PDIoT Coursework 3 (2024-25)

Human Activity Recognition using the Respeck Monitor

Douglas Torrance s1948873
Zhongfan Wang s2102597
Guoyu Zhang s1808795

Abstract

We built a user friendly app which connects to the RESpeck Bluetooth device to classify user activity in real time using machine learning models. The user can use their RESpeck device to measure and classify physical activity data, which is logged in a local database and visualised through interactive charts. This analysis allows users to track their overall distribution of activities or observe trends in a specific activity over time. The app also supports live sleep analysis, continuously classifying sleep states until manually stopped. It tracks key sleep metrics, including restlessness, sleep efficiency, and sleep duration, to assess overall sleep quality.

Table of Contents

Chapter 1

Introduction

1.1 Project Aims and Objectives

The aim of this project is to develop an Internet of Things (IoT) enabled mobile application that leverages machine learning to analyse physical activity, social signals and sleep quality using sensor data. The app connects to a Bluetooth device (Respeck) which is attached to the user's lower left ribcage. Machine learning models run inferences on this data to infer user activities or sleeping patterns. These analyses can then be visualised using interactive graphs.

The objectives of the app are:

- Provide functionality for supporting different users on the same app.
- Provide live, physical activity classification using the users movement data and save it locally.
- Provide statistics on the users sleeping patterns and use these to create a metric measuring sleep quality.
- Allow the user to view activity history and provide them with meaningful analysis.
- Evaluate the application's usability and practicality in real-world scenarios.

The objectives of our machine learning models are:

- Obtain a mean classification accuracy of 90% for static activities – static activities include sitting/standing, lying down on your left side, lying down on your right side, lying down on your back, and lying down on your stomach.
- Obtain a mean classification accuracy of 90% for dynamic activities – dynamic activities include walking, ascending stairs, descending stairs, running/jogging, shuffle walking, and miscellaneous movements
- Obtain a mean classification accuracy of 80% for each social signal class - normal breathing, coughing, hyperventilation and other social signals.
- Able to classify the sleep-wake cycle.

- Able to classify and count for positional change while the user is sleeping.

1.2 Brief Description of the Method Adopted and Summary of Results

The mobile application was developed for the Android platform using Kotlin. The UI has a modern style and is accessible and intuitive. The app allows multiple users to store and access their activity and sleep data separately. Authentication is handled by Firebase, using its cloud-based database to handle registrations and identity verification. The app can classify 11 different physical activities and stores these locally in a "Room" database. The user can visualise their activity distribution with a pie chart or track trends within a specific activity over time using a line graph. The app analyses the user's sleep data to generate an overall "sleep quality rating." However, sleep data is only viewed for a single session, and users cannot access past records as it isn't saved. Whilst our app provides all the features we aimed for it still has issues around practicality. The high battery consumption means that older phones would not be able to run it all day, we discuss the potential fixes to this problem in the "Android App Performance" section 3.8.3.

This project uses a hybrid deep learning model that integrates Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) layers for time-series classification of accelerometer data. This model effectively extracts multi-scale spatial features and captures temporal dependencies in sequential data. It also incorporates Inception modules with parallel convolutional filters (sizes 1, 3, and 5), batch normalization, and max-pooling layers to process features at varying scales. These features are further refined by bidirectional LSTM layers, which capture temporal relationships in both forward and backward directions.

The output layer classifies the data into activity labels across four domains:

- **Physical activities:** 11 labels, including actions like walking, running, and lying in various positions.
- **Social signals:** 4 labels, including normal breathing, coughing, hyperventilating, and others.
- **Sleep-wake cycle:** 2 labels, sleep and awake.
- **Positional change:** 4 labels, including lying on back, lying on stomach, lying left, and lying right.

To ensure robustness, the training pipeline employs:

- A 5-fold cross-validation strategy,
- Dynamic class weighting to address imbalanced datasets, and
- Adaptive learning rate adjustments.

Our model achieves an overall accuracy of 95% for classifying 11 distinct physical activities and 83.8% for classifying 4 social signal categories.

Chapter 2

Literature Survey

2.1 Human Activity Recognition

Human Activity Recognition (HAR) involves identifying and classifying actions performed by humans using data collected from sensors. The goal is to develop techniques and algorithms that can accurately perform this classification. With the advancements in wearable devices, mobile sensors, machine learning, and the broader Internet of Things (IoT) ecosystem, HAR has become increasingly important and useful, leading to a significant rise in development in this field. Accurate identification and classification of human activities are crucial for better understanding human behaviours across diverse applications.

HAR has numerous applications, including healthcare monitoring, sports analytics, and IoT environments such as smart homes. In healthcare monitoring, HAR enables remote tracking of patients with medical conditions, facilitating early detection and prevention of health issues (Chandramouli et al., 2024). In sports analytics, HAR helps monitor athletes' activity data, allowing them to identify problems, refine techniques, and enhance performance (Adel et al., 2022). Lastly, in smart homes, HAR can optimize home automation systems, improving convenience, security, and energy efficiency (Bouchabou et al., 2021).

While HAR offers many benefits, it also faces significant challenges, One major challenge is the lack of standardised data collection and annotation, which complicates comparison and reproducibility across studies. Furthermore, many datasets focus on a particular niche of human activity, underscoring the need for broader datasets that capture diverse activities across varied contexts. To address these issues, researchers are exploring solutions like sensor data fusion, which integrates data from multiple modalities to enhance the robustness and accuracy of HAR systems (Qiu et al., 2022).

2.2 Framework for HAR System

The typical framework for a HAR system involves data collection, data preprocessing, machine learning and classification, as seen in Figure 2.1.

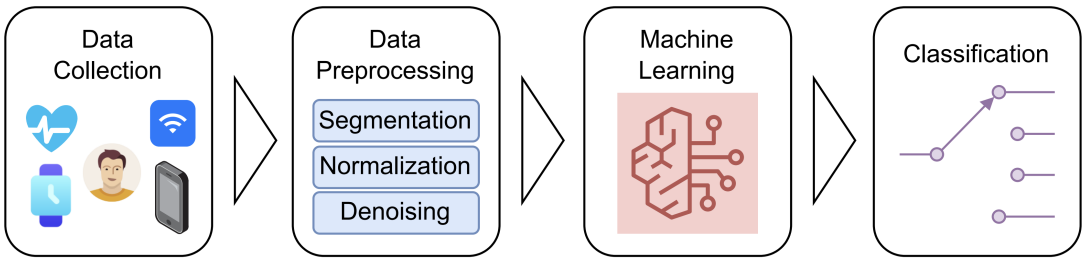


Figure 2.1: Generic framework of processes in HAR system.

Data collection involves capturing raw signals from sensors such as accelerometers, gyroscopes, or cameras, depending on the specific application (our project uses accelerometers and thus in our review we will mainly focus on acceleration-based sensing methods, as opposed to vision-based methods such as cameras). The raw data is then cleaned and structured using techniques like filtering, normalisation, and segmentation, ensuring consistency and quality. The preprocessed data is fed into machine learning algorithms, where models such as decision trees, support vector machines, or neural networks are trained to identify patterns and learn activity representations. Finally, the trained models are used to analyze new data, accurately classifying human activities for real-time or offline applications.

2.3 Deep Learning for HAR

Among the various machine learning approaches, deep learning has emerged as a transformative technology in Human Activity Recognition (HAR), driven by its ability to automatically learn complex features from raw or minimally preprocessed data. Unlike traditional machine learning models, which often require manual feature extraction, deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) excel at capturing spatial and temporal patterns directly from the input data, making them particularly well-suited for HAR applications.

Of the machine learning approaches, Convolutional Neural Networks (CNNs) have proven to be effective in extracting features for time-series classification tasks like HAR. CNNs offer advantages such as local dependency, which captures correlations between nearby signals, and scale invariance, allowing the model to handle variations in frequency effectively. For example, Maurya et al. (2022) demonstrated the efficacy of 1D CNNs for recognising activities such as studying, playing games, and mobile scrolling. Their approach utilised accelerometer data from a smartwatch to train a 1D CNN, achieving an accuracy of 98.28% while maintaining computational efficiency. Another approach leveraging CNNs was introduced by Cheng et al. (2022), who proposed a computationally efficient CNN model using conditionally parameterised convolutions (CondConv) for real-time HAR on mobile and wearable devices. This method demonstrated the accuracy on benchmark datasets while maintaining low computational costs. By dynamically adjusting the convolutional parameters based on the input data, the CondConv model balanced the trade-off between model capacity and computational efficiency, making it ideal for deployment on resource-constrained

devices.

Another popular approach is the use of RNNs, particularly with Long Short-Term Memory (LSTM) cells, which are effective in capturing long-term dependencies in Human Activity Recognition (HAR). However, a significant drawback of LSTM-based methods in HAR is their slower learning speed, largely due to their sequential nature (Edel and Köppe, 2016; Inoue et al., 2018). Edel and Köppe (2016) aimed to introduce resource-efficient RNN-based models by proposing a Binarised-BLSTM-RNN architecture. This approach binarised the weights and activations within the network, significantly reducing memory consumption and computational complexity by replacing high-precision arithmetic with bitwise operations. Despite these simplifications, the model still achieved an accuracy of approximately 90%, which was only slightly lower than full-precision networks, while being better suited for power-constrained environments such as mobile or embedded devices. Another LSTM-based approach by Chung et al. (2019) optimises sensor data acquisition and multimodal fusion for HAR. Using sensors on the wrists, waist, and ankle, their framework employs LSTM networks to model temporal dependencies and classify activities of daily living, including eating and driving. They found that low-frequency sampling rates (10 Hz) suffice for accurate recognition, reducing computational and energy demands. Their two-level ensemble model combines accelerometer, gyroscope, and magnetometer data, assigning customised weights to each sensor type based on activity type. The study demonstrates the practicality of LSTM models for efficient, scalable HAR systems in mobile and wearable applications.

2.4 Hardware Approaches to HAR

To address the limitations of wrist-worn devices, Yen et al. (2020) developed a waist-mounted wearable device aimed at improving HAR for individuals with medical constraints such as artificial blood vessels in their arms. This device integrates an inertial sensor comprising a three-axis accelerometer and gyroscope, alongside a Raspberry Pi-based microcontroller for data acquisition and processing. The accompanying activity recognition algorithm employs a 1D CNN. The system was validated using data from both the UCI HAR dataset (Reyes-Ortiz and Parra, 2013) and experimental data collected from 21 participants performing six common activities (walking, walking upstairs, walking downstairs, sitting, standing, and lying). The results demonstrated high recognition accuracy, achieving 95.99% on the UCI dataset and 93.77% on the experimental data.

Similarly, Chowdhury et al. (2020) leverages the integration of smartphone accelerometers and gyroscopes and IoT to propose a deep learning-based approach using the ”Human Activity Recognition with Smartphones” dataset, which consists of sensor data collected from 30 participants performing six common activities. The study used a neural network model optimized for HAR. By preprocessing the data and applying techniques such as Principal Component Analysis (PCA) for feature reduction, their five-layer network achieved a high testing accuracy of 96.47%.

Chapter 3

Methodology

3.1 Description of the System and Its Implementation

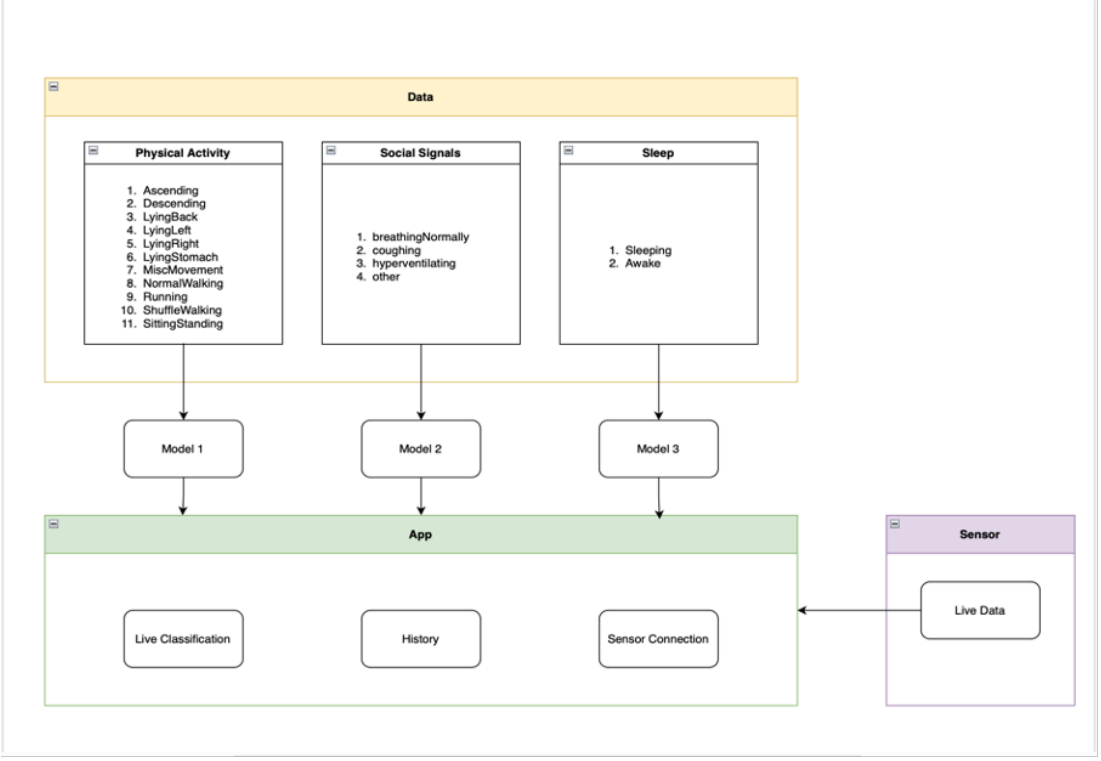


Figure 3.1: Architecture of the System Implemented

The system is a comprehensive real-time Human Activity Recognition (HAR) IoT solution designed to monitor physical activities, social signals, and sleep states using wireless Inertial Motion Unit (IMU) sensors and three machine learning models. It integrates data from the RESpeck and Thingy sensors and processes it through a user-friendly mobile application to provide live classification and historical insights.

The system supports three distinct categories of features:

1. **Physical Activity Recognition:** Identifying activities such as walking, shuffle walking, running, sitting, standing, lying down, ascending stairs, and descending stairs.
2. **Social Signal Classification:** Monitoring breathing patterns, normal breathing, coughing, hyperventilating, and other physiological signals.
3. **Sleep State Detection:** Determining whether the user is awake or sleeping.

This system allows users to use RESpeck or Thingy devices, or both sensor devices at the same time, to view their current activity, social signal, or sleep state in real-time. The mobile application serves as an interface that enables users to log in, track live data, and view their activity history. Historical data is visualized using graphs and activity summaries, allowing users to monitor trends over time.

3.2 Hardware

RESpeck

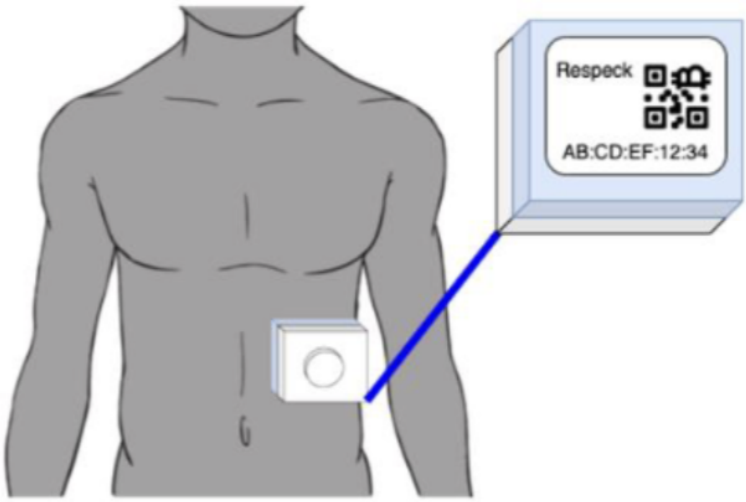


Figure 3.2: RESpeck device and how should it be set.

The RESpeck is a wearable device consisting of a tri-axial accelerometer and gyroscope, which allow it to record accelerations and angular velocities in three dimensions (x, y, z). The collected data is wirelessly transmitted to an Android App through Bluetooth Low Energy (BLE) technology, which operates at a sampling rate of 25 Hz. The device maintains continuous activity, automatically transmitting data to the connected mobile device.

The RESpeck sensor should be placed on the left lower ribcage—just below the ribs—with the blue half against the skin. You should be able to read the *RESpeck*

label when placing it on your chest; this ensures the sensor is held the right way up, as shown in Figure 3.2.

Thingy

Figure 3.3: Thingy device and how should it be set

The Thingy is a compact multi-sensor prototyping platform based around the NRF52 SoC, developed by Nordic Semiconductor. Similar to the RESpeck device, it is capable of recording linear accelerations in three dimensions (x, y, z).

The Thingy sensor should be placed in the front right pocket of your trousers, with the circle placed in the upper right corner and the USB port facing downwards, as shown in Figure 3.3.

3.3 Algorithms and Methods Used for Activity Recognition

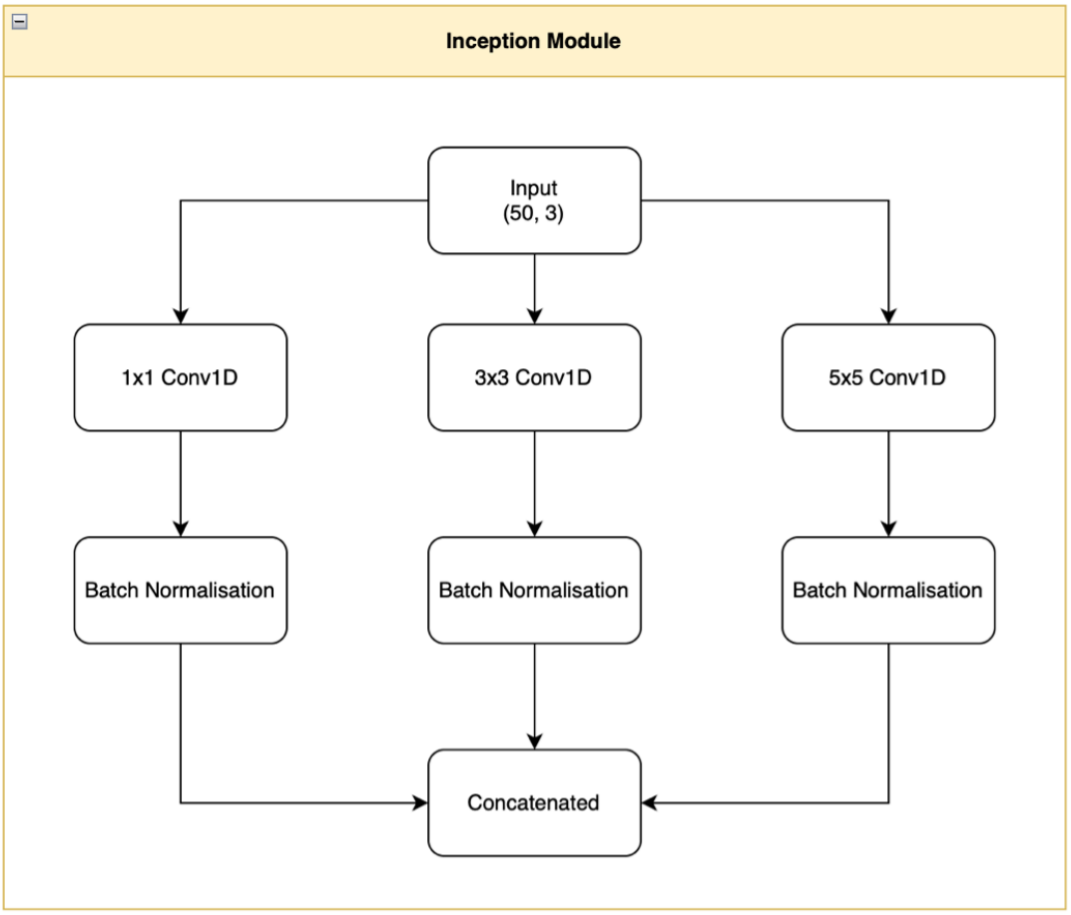


Figure 3.4: Inception Module

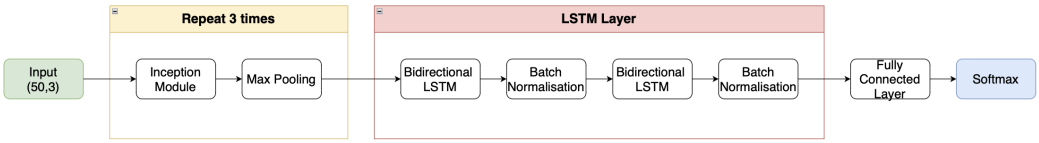


Figure 3.5: Architecture of our CNN-LSTM model

We designed a hybrid deep learning model combining Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) layers for time-series classification of accelerometer data. This architecture effectively extracts multi-scale spatial features while capturing temporal dependencies in sequential data.

The input layer accepts 3-channel accelerometer data with a fixed shape of $(50, 3)$, representing 50 timesteps across three axes (x, y, z).

We also designed an Inception modules to extract multi-scale features which uses parallel convolutional filters of sizes 1, 3, and 5. Each branch is followed by batch normalization to stabilize training and improve convergence. The outputs of these branches are concatenated along the feature axis. The model employs three Inception modules with 32, 64, and 128 filters, respectively. After each Inception module, a `MaxPooling1D` layer is used to reduce the spatial dimensions, ensuring efficient feature extraction.

The LSTM layers process the feature sequences generated by the Inception modules. A bidirectional LSTM layer with 128 units captures temporal dependencies in both forward and backward directions, returning sequences for further temporal analysis. This is followed by a second bidirectional LSTM layer with 64 units, which outputs a fixed-length vector. Batch normalization is applied after each LSTM layer to enhance training stability.

The fully connected layers condense the learned features into a dense layer with 256 units and ReLU activation. To prevent overfitting, a dropout layer with a rate of 0.5 is included.

Finally, the output layer maps the extracted features to N activity classes, where N is the number of activity labels. This layer uses a dense configuration with a softmax activation function to produce probabilistic predictions across all activity classes.

3.4 Training

The data was initially divided into 90% for training and 10% used for testing.

The training process employed a 5-fold cross-validation strategy to comprehensively evaluate the model's performance and ensure its generalizability. In this approach, the dataset was divided into five folds, with one fold serving as the validation set and the remaining four folds used for training during each iteration. This method ensured that the model was trained and validated on different subsets of the data, providing a robust assessment of its capabilities.

To address class imbalance within the dataset, class weights were computed and applied during training. These weights ensured that all classes were equally represented in the learning process, regardless of their frequency in the data, thereby preventing the model from favoring majority classes. The weights were dynamically computed for each fold based on the distribution of labels in the training set.

The model was trained using the Adam optimizer with a learning rate of 0.0001, which allowed for efficient gradient updates while minimizing overfitting. A batch size of 32 was used, ensuring a balance between computational efficiency and the stability of gradient updates. The training was conducted for a maximum of 100 epochs for each fold, with the option to terminate early if the validation loss failed to improve for 15 consecutive epochs.

To further optimize the training process, the learning rate was reduced by a factor of 0.5 if the validation loss did not improve for five consecutive epochs. This dynamic

adjustment allowed the model to make finer weight updates in later stages of training, leading to improved performance. Validation performance was monitored at the end of each epoch, and the model parameters that resulted in the best validation loss were saved to ensure optimal performance.

The training history, including metrics such as loss and accuracy for both training and validation datasets, was recorded for each fold. This data provided insights into the model's learning progress and helped identify any trends or issues during training.

By combining cross-validation, class balancing, adaptive learning rate adjustment, and performance monitoring, this training strategy facilitated the development of a robust and well-generalized model capable of accurately classifying time-series data.

3.4.1 The Physical Activity Model

The data are labeled in 11 different labels, Ascending, Descending, Lyingback, LyingLeft, LyingRight, LyingStomach, MiscMovement, NormalWalking, Running, ShuffleWalking and Sitting Standing. Than, the physical activity model is trained on these data.

3.4.2 The Social Signal Model

The data are labeled in 4 different labels, breathingNormally, coughing, hyperventilating and other. Than, the Social Signal model is trained on these data.

3.5 Algorithms and Methods Used for Sleep Analysis

For the sleep analysis, we trained two models utilising data from 45 users, including questionnaire responses detailing timestamps for when users went to bed, fell asleep, woke up, and got out of bed. The analysis aims to classify sleep-wake cycles, count positional changes during sleep, calculate sleep efficiency, and use this information to determine a sleep quality index.

The general architecture and training process for the models follow the framework used for activity and social signal recognition. However, modifications were made to the input data and the classification layer to develop two new models: one to classify sleep-wake cycles and another to classify sleep positions (back, stomach, left side, or right side), which aids in counting positional changes.

3.5.1 Initial Data Analysis

To begin, we analysed the sleep data and corresponding questionnaire responses. Out of the 45 users' sleep data, only 34 had matching entries in the sleep questionnaire. This discrepancy meant that data from 11 users, lacking annotated timestamps, could not be used for training the sleep-wake classification model. The absence of timestamps prevents the generation of sleep and awake labels necessary for supervised learning. Additionally, within the 34 entries with questionnaire data, some annotations did not

align with the recorded timestamps. For instance, as shown in Figure 3.6, User 20's data ends prematurely, before the wake-up timestamp, making it incomplete. These inconsistencies reduced the usable dataset to fewer than 34 complete entries.

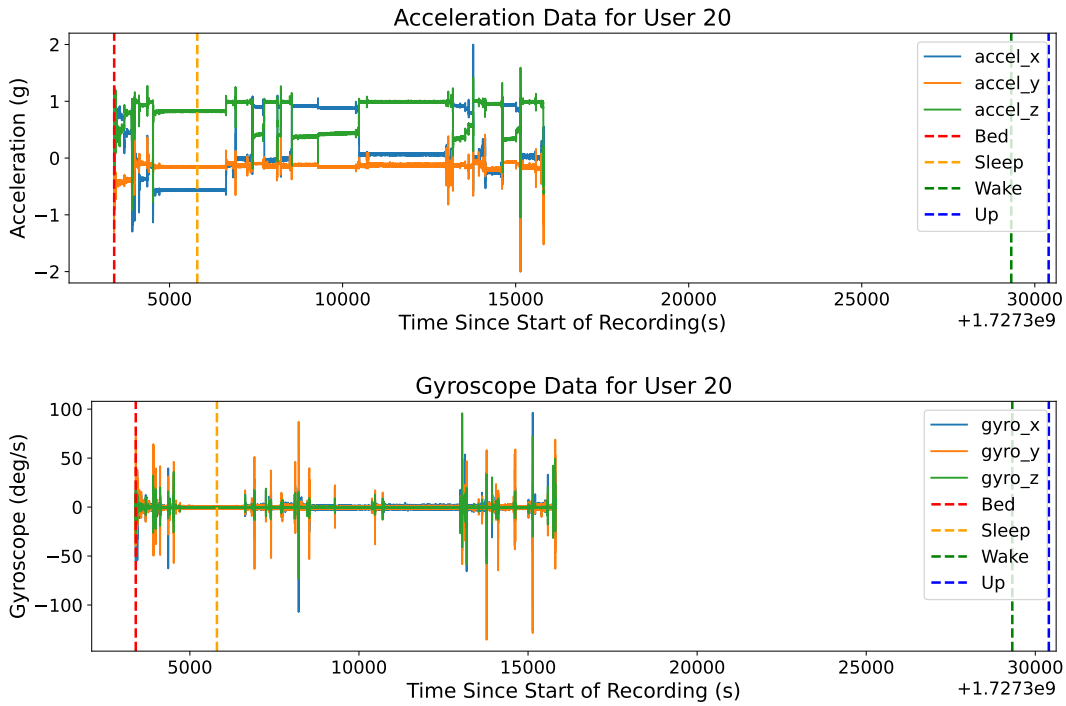


Figure 3.6: Accelerometer and Gyroscope Data for User 20

3.5.2 Data Preprocessing

Given these challenges, we preprocessed the data to prepare it for training the sleep-wake classification model. For the 34 users with corresponding questionnaire entries, we divided their data into three intervals based on timestamps:

1. From the start of the recording to the timestamp when the user fell asleep.
2. From the time the user fell asleep to the time they woke up (classified as sleep data).
3. From the time the user woke up to the end of the recording.

Data from before and after the sleep interval, along with activity and social signal data from 2023–2025, were labelled as awake data. To conduct a sleep analysis, we set aside data from six users, selecting two random users from each sleep quality category (poor, average, and good). The remaining data was used for training. Since the sleep data was more than double the awake data, we downsampled the sleep data to balance the class distribution. Finally, the data was split into 90% for training and 10% for testing.

3.5.3 Sleep-Wake Cycle Classification

To classify sleep-wake cycles and calculate sleep efficiency, we trained a binary classification model using the same architecture, training configurations, and hyperparameters as the physical activity and social signal models. The input for this model included gyroscope data in addition to accelerometer data, resulting in six-channel input with a shape of $(50, 6)$. Here, 50 represents a window size of 2 seconds at 25 Hz, and the six channels correspond to three axes of accelerometer and gyroscope data. The model was trained to predict two labels: sleep or awake.

Using the sleep-wake classification model, we calculated sleep efficiency for the test users. Sleep efficiency is defined as the percentage of time spent asleep compared to the total time spent in bed. This was calculated by dividing the number of windows classified as sleep by the total number of windows and multiplying by 100.

3.5.4 Position Classification for Positional Changes

To count positional changes, we trained a separate model to classify sleeping positions. While the physical activity model already included classes for lying on the back, stomach, left side, and right side, it also contained unrelated labels, increasing the potential for misclassification. Thus, a dedicated model was trained using only data from these four position classes, collected during 2023–2025.

This model used accelerometer data with an input shape of $(50, 3)$, representing a 2-second window of three-axis accelerometer data. The architecture and training process were similar to the activity recognition model, but the output classification included only four labels: lying on the back, stomach, left, or right.

To count positional changes, we used the sequence of classified positions for each user. A positional change was counted only if a new position persisted for at least three consecutive windows, mitigating the effects of brief, minor movements or misclassification. For instance, if a sequence transitions from lying on the back to lying on the right for several windows, it would count as one positional change.

3.5.5 Sleep Quality Index

Using sleep efficiency and the number of positional changes, we conducted further analysis to explore relationships, such as whether we could come up with a sleep quality index that could be modelled as a function of sleep efficiency and positional changes:

$$\text{Sleep Quality} = x \times \text{Sleep Efficiency} + y \times \text{Number of Positional Changes}.$$

3.6 Mobile Application Implementation

We extended the sample Kotlin PDIOT app to provide the required functionality. We developed this app in Android Studio version 2024.1.2. It uses the Android SDK compile version 31, Android Gradle Plugin version 7.4.2 and Kotlin version 1.6.21. The minimum SDK version required is 23 and the target version is 31.

3.6.1 Tools Used to Develop the App

We developed this app in Android Studio which allowed us to use a range of integrated tools:

1. **Phone Emulation:** This allows us to examine the appearance of our UI on many different screen sizes and orientations. This was crucial for ensuring that our app provided a consistent user experience across a wide range of Android devices, reducing the need for extensive physical device testing.
2. **XML Layout Editor:** This drag-and-drop visual editor, with integrated layout constraints, allowed us to quickly design dynamic UIs which adapted to different screen sizes.
3. **Theme Editor:** This tool allowed us to define UI elements according to inclusive "Material Design Principles." This is discussed further in the UI section of this report.
4. **Android Profiler:** This tool was vital to provide application runtime performance statistics that include CPU usage, energy consumption, memory usage, etc. This was particularly important for optimizing the app's responsiveness, ensuring smooth real-time data processing from the Bluetooth sensor, and improving battery efficiency.
5. **Gradle Build Tool:** We used Gradle as our build tool as it has built-in integration with Android Studio. This tool was vital as we used many third-party libraries complex and interdependent dependencies. We could easily resolve dependency conflicts using the automatic dependency conflict flagging mechanism.
6. **Version Control Integration:** Android Studio integrates with GitHub, which allowed us to easily keep track of each other's branches and provided an easy UI to help resolve merge conflicts and manage branches.

Firebase: We used the Firebase service suite to enable user authentication and account management functionality in our app. The Firebase console provided an intuitive web-based interface for configuring security and session management rules, significantly simplifying what would otherwise have been a complex manual process. The console's GUI allowed us to easily test and validate the App's connection to our online database. The Firebase Kotlin SDK allowed us to directly interact with Firebase services directly in our code, providing a type-safe way to manage our connections.

3.6.2 Functions and Features of the App

3.6.2.1 Login and Sign Up

Feature: Provides essential user authentication and registration functionality, ensuring secure access to each user's data.

1. **User Login:** When first entering the app the user is prompted to enter their username and password. The app sends these credentials to a remote Firebase server for verification. If the user logs in successfully the server returns a secure authentication token to verify the user's identity for subsequent actions in the app. If the log-in attempt fails, Firebase returns an error and the user is prompted to log in again.

2. **User Registration:** New users are required to register a personal account by clicking the Sign Up button on the Login page. This action redirects them to the registration form, where they input their name, email, and password. The app sends a message to the end point createUserWithEmailAndPassword, which creates a new entry in the database. Once the user has done this they are returned to the Login page and can log on using their new username and password.

3.6.2.2 Home Page

Feature: Easy navigation throughout the app to access other features.

After logging in, the app opens the home page. The home page passes the authentication token to any subsequent page that the user requests. The home page is the main navigation portal, the users can access the app's features from here:

1. Activity Classification
2. Sleep Analysis
3. Viewing Activity History
4. Connecting Sensors

3.6.2.3 Activity Classification and Recording

Feature: Classifies the sensor data readings it receives from the Respeck Bluetooth device and stores them in a local Room database.

It provides live classification for the user and also displays the raw data, allowing the user to test if the classification model is working and the RESpeck device is returning the correct data.

This feature requires the RESpeck device to be connected. This component receives the live data in the form of (x,y,z) coordinates from the RespeckPacketParser. It operates a sliding window and waits until it has 50 fresh readings from the sensor. It runs a machine learning model and classifies the action by returning a string. It stores each classification as a triple (userid, activity, timestamp) in a Room Database.

3.6.2.4 Sleep Monitoring

Feature: This features also provides live classification using a machine learning model. It requires a RESpeck device connection and uses the RespeckPacketParser to parse packets. However it doesn't save the results to a database. It updates the statistics in real time whilst the user is recording and when they stop recording it freezes the final results so they can view them. It provides statistics for:

- Positional Changes: the number of times the user has rolled over.
- Sleep Efficiency: the percent of time the user has spent sleeping since starting the recording.

- Quality Rating: a weighted heuristic to measure the overall quality of the users sleep. It is calculated as:

$$\text{Sleep Efficiency} = \frac{\text{Sleeping Readings}}{\text{Total Readings}}$$

$$\text{Duration Factor} = 1 - \frac{|\text{Time Slept} - 8\ \text{Hours}|}{\text{Ideal Sleep Duration}}$$

$$\text{Turn Factor:} \quad \text{Turn Factor} = 1 - \min\left(\frac{\text{No. Turns}}{40}, 1.0\right)$$

$$\text{Sleep Quality Index } = 0.4 \cdot \text{Efficiency} + 0.3 \cdot \text{Duration Factor} + 0.3 \cdot \text{Turn Factor}$$

Here we assume that the best amount of sleep for a person is 8 hours and turning more than 40 times in one night is an indication of poor sleep Skarpsno et al. (2017)

3.6.2.5 View Activity History

Feature: Allows the user to view statistics about their previous activity history.

This component receives the user authentication token from the Home Page. It uses this unique identifier to query a locally stored Room Database to access the users data. This feature allows the user to see two types of statistics about their previous activities:

1. Overall Activity Distribution: After the user selects a date, the app uses the user ID and selected date to query the Room Database. It receives the data for that day in the form (userid, activity, timestamp). The app calculates the proportion of time spent on each activity by summing the number of readings for each activity and dividing by the total number of readings for that day. This data is displayed as a pie chart, showing the relative time spent on each activity.
2. Specific Activity Intensity: When the user selects a specific activity, the app generates a line chart showing the relative intensity of that activity throughout the day. The app processes activity classification readings, typically taken every 2 seconds, and groups them into 5-minute intervals. For each interval, it calculates the percentage of time spent on the selected activity and plots this data on the chart with ”Relative Intensity” on the y-axis and Time on the x-axis.

3.6.2.6 Connecting Sensors

Feature: Allows the user to connect the sensors to the app. The user can connect the RESpeck device by:

1. Manually inputting MAC address written on the RESpeck
2. NFC connection

3. Scanning the QR code on the RESpeck

The RESpeck's ID (MAC Address) is saved between user sessions, so the user doesn't have to do this again (unless they change the specific RESpeck device).

3.6.3 App User Interface

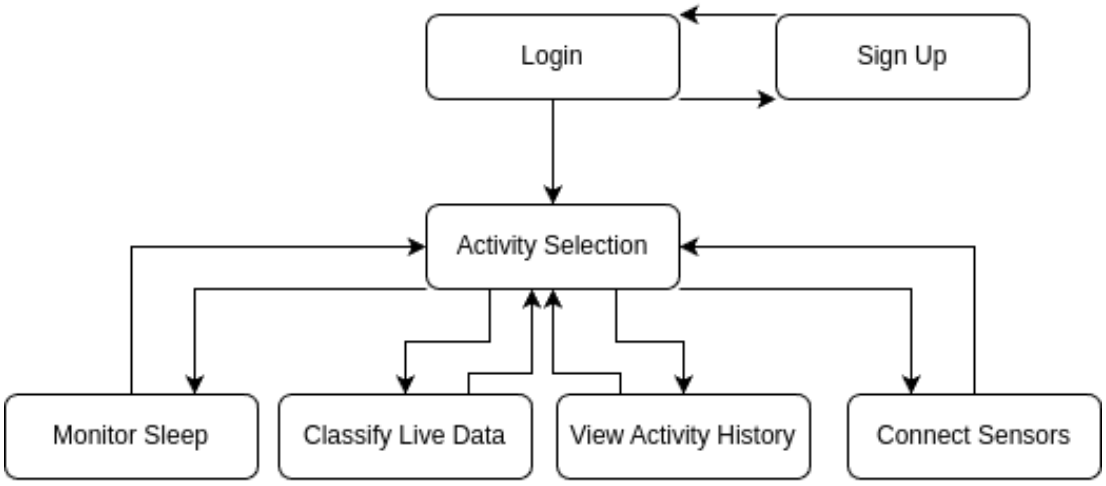


Figure 3.7: Flow Diagram describing page transitions

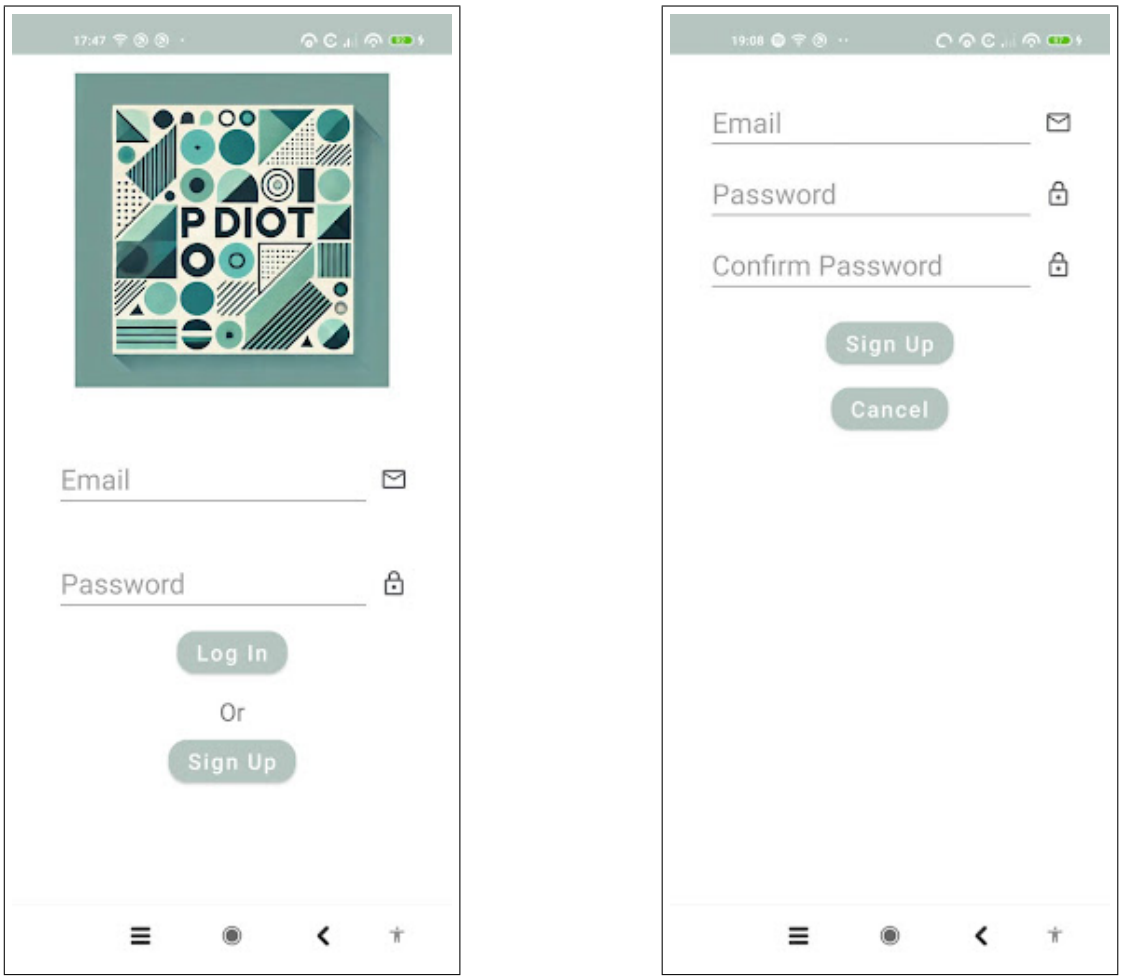
3.6.3.1 Design Philosophy

We have designed this App with a focus on usability, intuitiveness and accessibility. To achieve this we chose to design our UI based on Google's Material Design System, a comprehensive framework which provides design guidelines and customisable UI elements such as buttons, icons and fonts. Adhering to these guidelines helped us maintain a professional and user-friendly design throughout the app.

The UX elements we used are consistent with those found in Google's applications, providing users with a sense of familiarity. For example, we used the same icons and widgets as Google because users will already be familiar with what they mean and how they work. Consistency is a key principle; it keeps interactions intuitive and predictable. Using the style.xml file we enforced a consistent appearance for all our UX elements e.g button shapes, text fonts and theme colours.

Material Design also takes great care to ensure its designs are accessible. It encourages the use of adaptive layouts, responsive design, and accessible colour contrasts to support users with varying needs, including low vision, cognitive impairments, and motor disabilities. For example, we followed recommendations for larger touch targets and well-contrasted colour schemes to improve usability for individuals with vision impairments. Another strength of Material Design is its support for older versions of Android and its adaptability to different screen orientations and device sizes. This flexibility ensured that our app delivered a consistent experience across a wide range of devices, from small smartphones to large tablets.

3.6.3.2 Login & Sign Up Screens



(a) Sign in page

(b) Registration page

Figure 3.8: User authentication pages

We ensured that the same visual language, button styles, and iconography are used across all screens to provide a cohesive and seamless user experience. Consistency reduces cognitive load and helps users navigate the app with confidence, as familiar elements reinforce intuitive interaction patterns.

The muted green tones chosen for buttons and icons align with the app's overall aesthetic and branding. This color palette is modern and professional, while also providing enough contrast to guide users' attention to key interactive elements.

3.6.3.3 Activity Selection

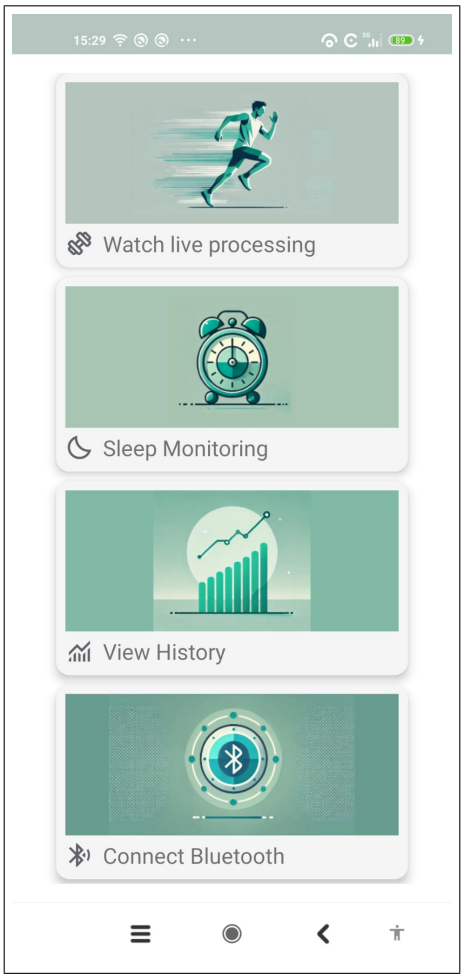


Figure 3.9: Home page

We used Google Material's Cards to allow the users to select between different options. This is a clean and modern approach favoured by Google. Android users will be familiar with their appearance. Their size and ability to store multimedia content makes them very versatile. We have used images, text and icons to allow the user to quickly understand what each option does at a glance. By providing three distinct modes of communication, we improved the app's accessibility and inclusiveness, catering to a diverse range of users with different preferences or needs. The combination of large fonts, large buttons, high contrast from elements to their backgrounds make this intuitive and accessible.

3.6.3.4 Classifying Live Data

Figure 3.10: Activity classification page

This feature is primarily provided by background activity, so there isn't much need for user interactivity. Nevertheless this provides clearly organised labels each with intuitive icons. We added graphs of the raw live data to allow the user to tell if their devices were transmitting data correctly and had the right orientation.

3.6.3.5 Monitoring Sleep Quality

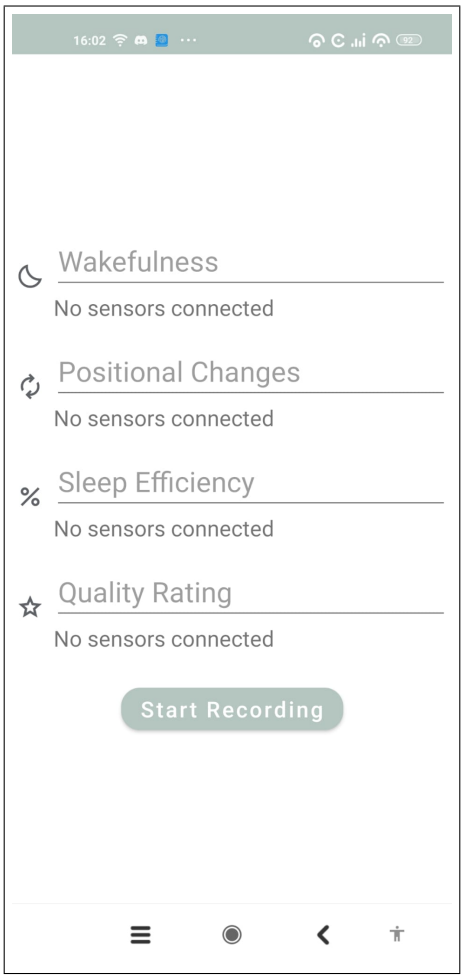


Figure 3.11: Sleep classification page

We kept the style similar to the Activity Classification to maintain consistent design. When the user clicks the ”Start Recording” button it turns red and the text changes to ”Stop Recording”. When the user presses ”Stop Recording” it goes back to green.

3.6.3.6 Viewing Activity History

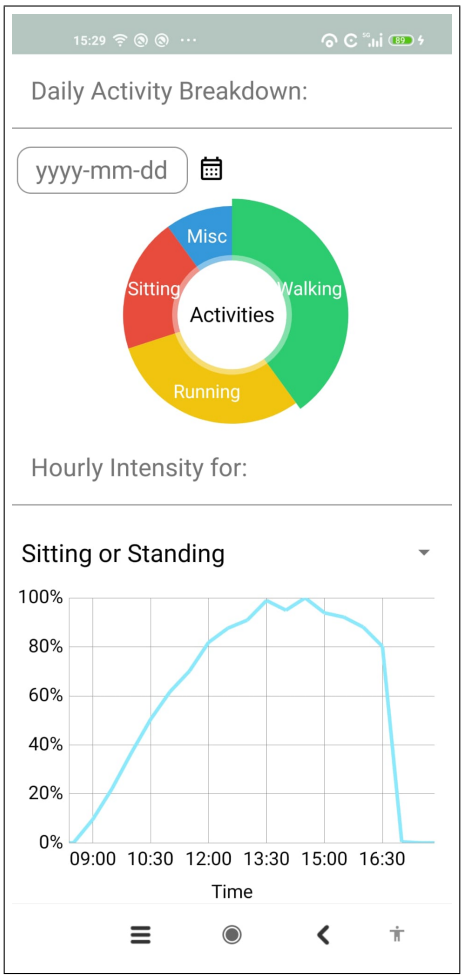


Figure 3.12: Previous activity statistics

We aimed to display the historic data in an informative and intuitive way. The top label indicates that the data is broken down into individual days. The user first selects the date by clicking the text box labeled with the calendar icon. This opens a commonly used calendar widget, making it easy for the user to select a date.

After selecting a date, a dynamic pie chart displays the distribution of activities performed on that day. The user can then select a specific activity for that day and a line graph will display when they were performing that activity throughout the day allowing the user to track trends in their behaviour throughout the day.

3.6.3.7 Bluetooth Connection

Figure 3.13: RESpeck connection page

This page retains the original design provided by the sample app to connect Bluetooth devices, as its straightforward interface effectively delivers clear instructions on multiple connection methods for each sensor. The only modifications we implemented involved adjusting the shape and colour of the buttons to align with the app's pre-existing style, maintaining visual consistency across all screens.

3.7 Software Organization

3.7.1 Design Pattern Selection

We followed the Model-View-Controller (MVC) design pattern during the development of this app. Although we considered following an MVVM design pattern we decided against it. MVC allowed us to have a far faster implementation time. This was critical for us to meet the tight deadlines we had.

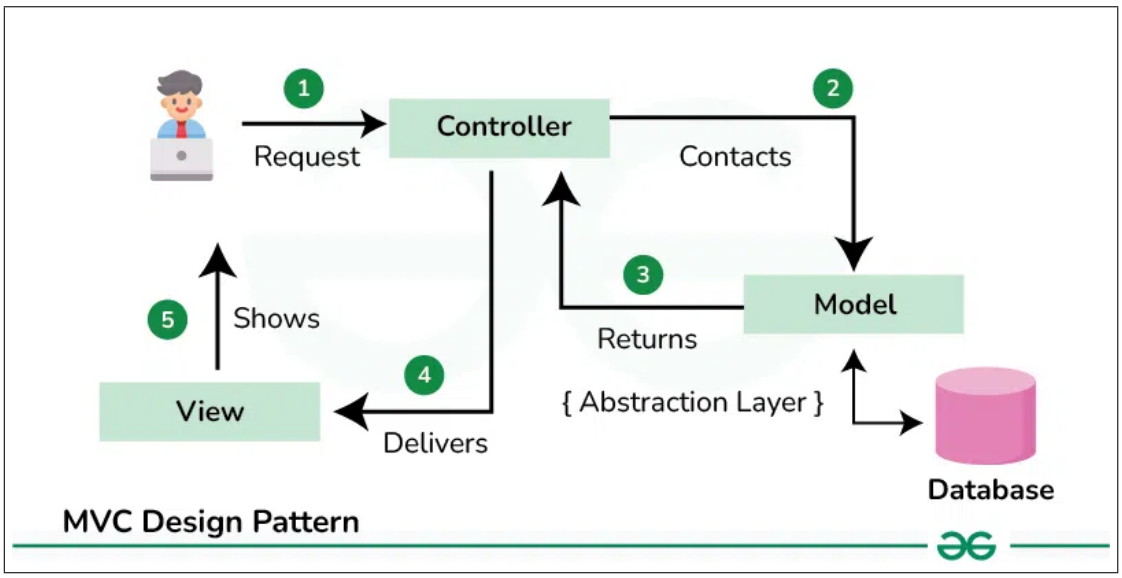


Figure 3.14: MVC Design Pattern. Source: GeeksforGeeks (n.d.)

It is a simpler design pattern as it avoids the new layers of abstraction that MVVM entails. Given the few components, limited features and limited future plans for our project, we decided to keep it simple and decided the extra layer of abstraction would add more complexity than it would save. If this app was developed further we would switch to an MVVM approach. There were also performance concerns as adding this extra layer would add memory and latency delays to our app. MVC allowed us to directly update the UI from the controller and made it easier to work together as a team as it was obvious where the logic for each component resides.

3.7.2 Feature Driven Design

The organisation of our components is done primarily by feature. We split the work based on features:

1. Authentication
2. Live Data Classification & Sleep Analysis
3. Activity History

This made it easy to divide the tasks between ourselves. It provided modularity as each team member worked on their feature independently without being constrained by the progress of another teammate. It was also an effective way to engage in agile practices as we could quickly develop basic functionalities. As a team, we had limited experience developing Mobile Apps and no experience with Kotlin. So it was hard for us to predict how long it would take us to develop features, making it difficult to develop accurate long term plans.

This design methodology allowed us to quickly develop a minimum viable product with very basic features. We could iteratively and incrementally improve the product. This was very effective for us as we quickly had a working product which we could

improve upon for demo day, instead of having an app with a wider scope, which was only partially developed.

3.7.3 Data Flow Visualisations

Understanding the data flows of our application is key to understanding the organisation of our software.

There are three main flows to consider based on our primary features.

Classifying Live Data

Flow: RESpekLiveData - Classifier - LiveData (- RoomDatabase)

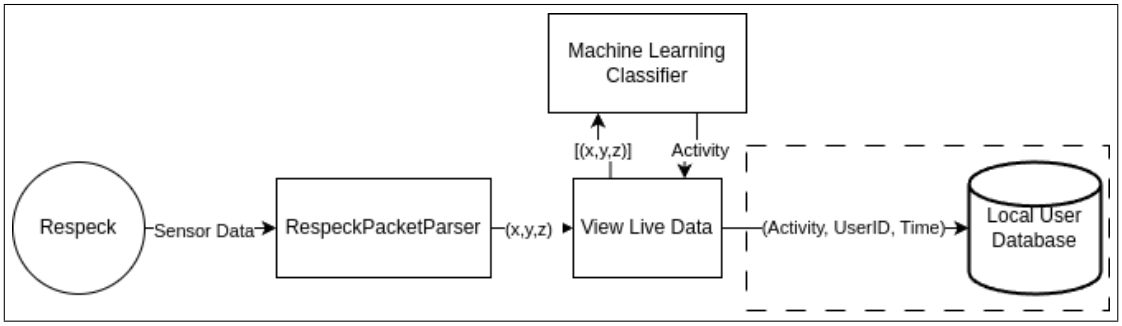


Figure 3.15: Data classification flows

This flow describes how we take raw sensor data from the RESpeck device and turn it into an activity represented as a string.

The data is transported over bluetooth and received by the RespeckLiveData.kt component. This component interprets the incoming byte stream from the Respeck device and converts it into a format usable by the app. The LiveDataActivity component transforms this stream of coordinates into a sliding window array of size 50. Each entry is an array of size three containing the (x,y,z) orientation of the Respeck Device. This transformation is necessary to put the data into the correct shape for our machine learning model. This process happens every 2 seconds to allow for a completely new window to be filled for the classifier. Our classifier component takes this shaped data and analyses it using our TFlite model to return the predicted result as a string. This string is fed into the LiveData component where it is displayed to the UI to provide the user instant feedback.

In the physical activity classification feature, the prediction is also associated with the user's ID and is timestamped and saved to a local RoomDatabase.

Displaying Historic Data

Flow: RoomDatabase - Display

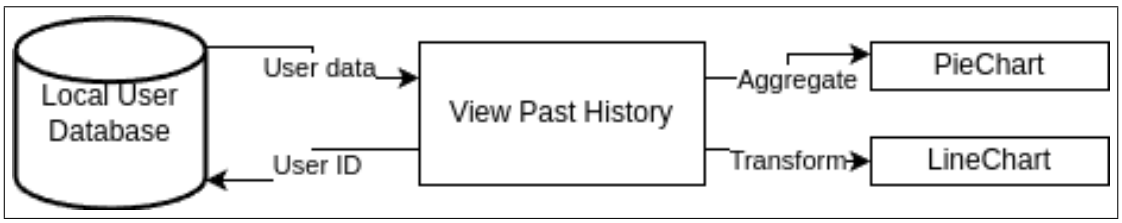


Figure 3.16: Visualising past activity flow

This flow outlines how stored data is retrieved and transformed for visualization.

Data saved in the RoomDatabase is queried when the user requests historic activity statistics. Because the app follows an MVC design pattern, the same component is responsible for both processing the queried data and rendering it for display. This streamlined approach minimizes complexity.

User Authentication

Flow: Authentication - Firebase Service - Authentication - HomePage

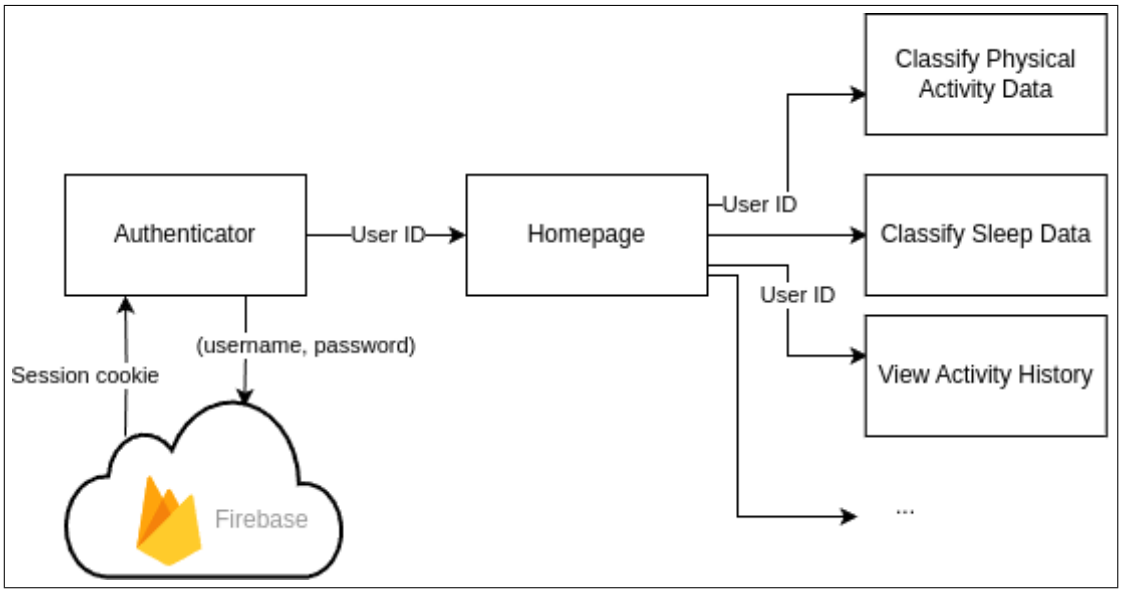


Figure 3.17: Authentication flow

This flow secures the app by verifying user credentials and managing sessions.

The Authentication component processes the username and password entered by the user, formatting them into a POST request compatible with Firebase's authentication API. If the user's details were valid, it returns a session cookie, if the user's details were invalid it throws a checked exception. The session cookie is passed onto the rest of the program to ensure the user's identity. This identity is important as it ensures that writes to the database are correctly tagged with the user's identity and it ensures that each user can only access their own data.

3.8 Testing

We have devised a thorough testing plan to ensure all our required functionalities work as expected. Given the difficulty of mocking web services, databases and sensor input, these tests are performed manually.

The following servers as a checklist to perform after refactoring.

3.8.1 Unit Testing

Unit tests will test each component in isolation. We test their correctness under normal inputs (including edge cases) and robustness under exceptional or invalid cases.

Authentication: You must login to our Firebase console here: J1 Users Database. From this console you can view registered users and login attempts. Verify that the following features work correctly:

- Login.kt
  - Valid Login Attempt: Logs the user in and redirects them to the homepage.
  - Invalid Login Attempt: Rejects the input and asks them to input the credentials again.
- SignUp.kt
  - Valid Sign Up: Verify this creates a new user using the Firebase dashboard.
  - Invalid Sign Up Attempt: Verify this doesn't create any new users.

Bluetooth Functionality: This requires the tester to provide the requested permissions, turn on their Bluetooth and have a working RESpeck device available:

- ConnectingActivity.kt
  - Valid Connection: once you have connection navigate to the live activity classification page to ensure data is being received.
  - Invalid MAC Address: ensure this does not crash the app.
  - No Available Devices: ensure this does not crash the app.

Live Data Classification and Data Visualisation

- ActivityHistoryManager.kt
  - Verify database reading and writing operations: we have included helper functions which write synthetic data to the database (using the current date) and then retrieve the data using SQL queries.
  - Visualise data: additional helper functions can feed the data collected into the graph writing functions allowing us to test our visualisations.
- LiveData.kt and SleepClassification.kt

  - Verify machine learning model through manual testing: connect the respeck and verify that the model is returning the correct predictions given your current movement.

3.8.2 Integration Testing

- UI Testing: Test the control flow of UI pages. It should only allow transitions in accordance with the Flow Diagram described in the UI section: 3.7
- Authentication: Test that each page has access to the users session token as parameter. This can be done using LogCat, filtering for (PAGE, user-token)
- Classification: Manually test on LiveData.kt that each classifiable action is classified correctly by the app. Then navigate to ViewHistoryActivity.kt to ensure that the data is being visualised properly.

3.8.3 Performance Testing

Here we use Android Profiler to obtain performance metrics for our app.

- CPU usage
- Memory usage
- Battery consumption

The results of these tests are listed in the 'Performance' section.

Chapter 4

Results and Discussion

4.1 Evaluation Metrics of Machine Learning Models Implemented

Precision

Precision assesses the veracity of the model's positive predictions. It evaluates the fraction of base pairs predicted by the model that are correct in the ground truth structure. Precision is mathematically defined as:

$$\text{Precision}_i = \frac{\text{True Positives (TP)}_i}{\text{True Positives (TP)}_i + \text{False Positives (FP)}_i} \tag{4.1}$$

Recall

Recall measures the model's proficiency in correctly detecting all pertinent base pairs. It reflects the proportion of actual base pairs that are successfully identified by the model. Recall is mathematically expressed as:

$$\text{Recall}_i = \frac{\text{True Positives (TP)}_i}{\text{True Positives (TP)}_i + \text{False Negatives (FN)}_i} \tag{4.2}$$

F1-Score

The F1 score is a composite metric that synergistically combines precision and recall to provide a single, balanced measure of the model's predictive power. The F1 score is formulated as:

$$\text{F1-Score}_i = 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4.3}$$

A superior F1 score signals a model's capacity for high accuracy in both detecting correct base pairs and avoiding incorrect predictions.

Accuracy

$$\text{Accuracy} = \frac{\sum_i \text{True Positives (TP)}_i + \text{True Negatives (TN)}_i}{\text{Total Samples}} \tag{4.4}$$

4.2 Daily Physical Activity Classification

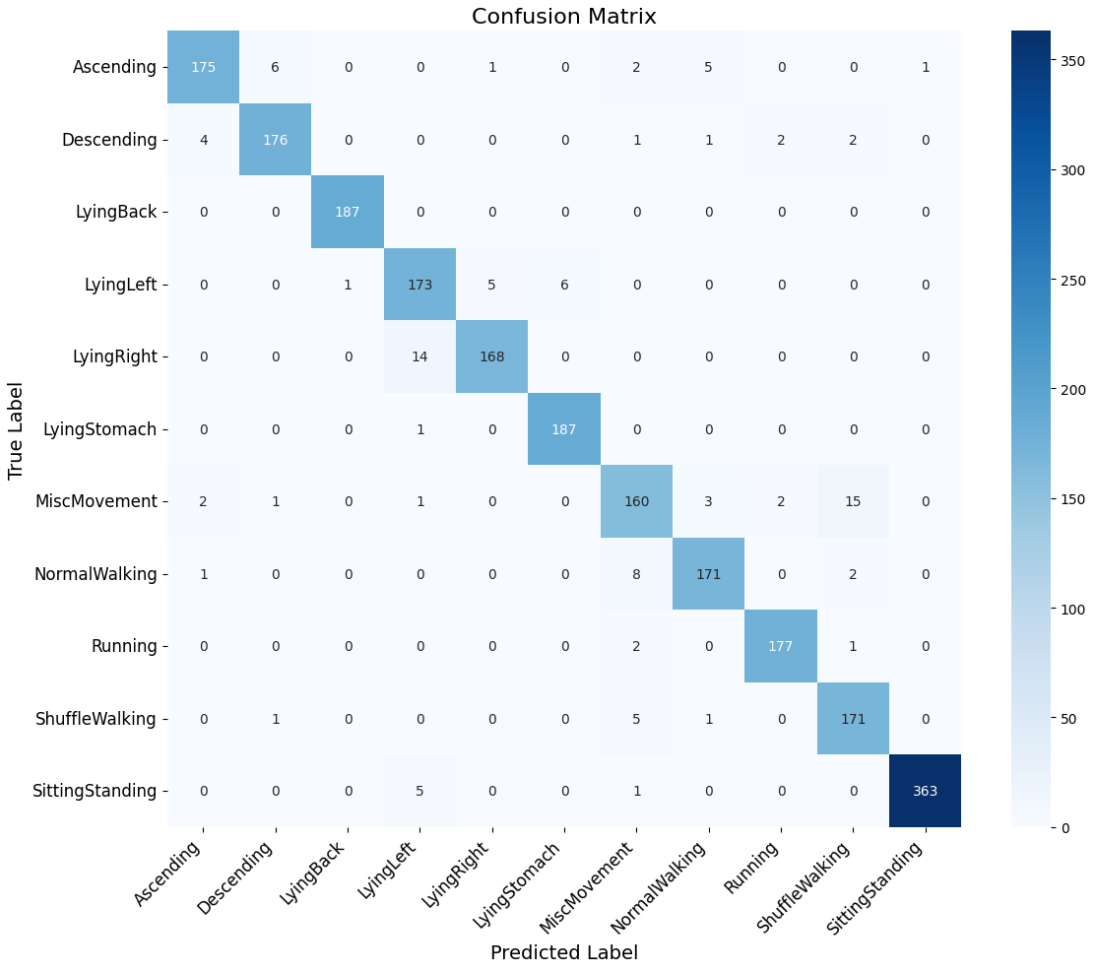


Figure 4.1: Confusion matrix for Physical Activities

Table 4.1: Classification Metrics for Each Class

Class	**Precision**	**Recall**	**F1-Score**
Ascending	0.96	0.92	0.94
Descending	0.96	0.95	0.95
LyingBack	0.99	1.00	1.00
LyingLeft	0.89	0.94	0.91
LyingRight	0.97	0.92	0.94
LyingStomach	0.97	0.99	0.98
MiscMovement	0.89	0.87	0.88
NormalWalking	0.94	0.94	0.94
Running	0.98	0.98	0.98
ShuffleWalking	0.90	0.96	0.93
SittingStanding	1.00	0.98	0.99
Mean	0.95	0.95	0.95
Accuracy (Overall)		0.95	

The overall performance of the model is notable. The model achieved an overall accuracy of 95% for this multi-class classification problem with 11 distinct classes.

Table 4.1 shows that the model performs exceptionally well for static activities such as *LyingBack*, *SittingStanding*, and *LyingStomach*, achieving a Precision of more than 0.97, Recall of more than 0.98, and F1-score of more than 0.98. For instance, *LyingLeft* and *LyingRight* have lower results, but they still achieve a Precision of 0.89 and 0.97, a Recall of 0.94 and 0.92, and an F1-score of 0.91 and 0.94, respectively. The confusion matrix 4.1 provides further insights into the model's behaviour. Misclassifications are more prominent in classes like *LyingLeft*, which is sometimes misclassified as *LyingRight* or *LyingStomach*. This is likely due to overlapping features or postures shared by these activities and due to not using gyro information in the training process.

Table 4.1 also shows that the model performs well for dynamic activities, such as *Running*, *Ascending*, and *Descending stairs*. For instance, *MiscMovement* has the lowest F1-score of 0.88, which suggests difficulty in correctly identifying this activity. Similarly, *ShuffleWalking* shows a slightly lower F1-score of 0.93, indicating some feature overlap with other activities. These results highlight areas where the model struggles to differentiate between similar activities or those with diverse patterns. Again, the confusion matrix 4.1 provides further insights into the model's behaviour. We can see that *MiscMovement* is often confused with other classes, likely because it encompasses a wide variety of motions that are harder to distinguish.

4.3 Social Signals Classification

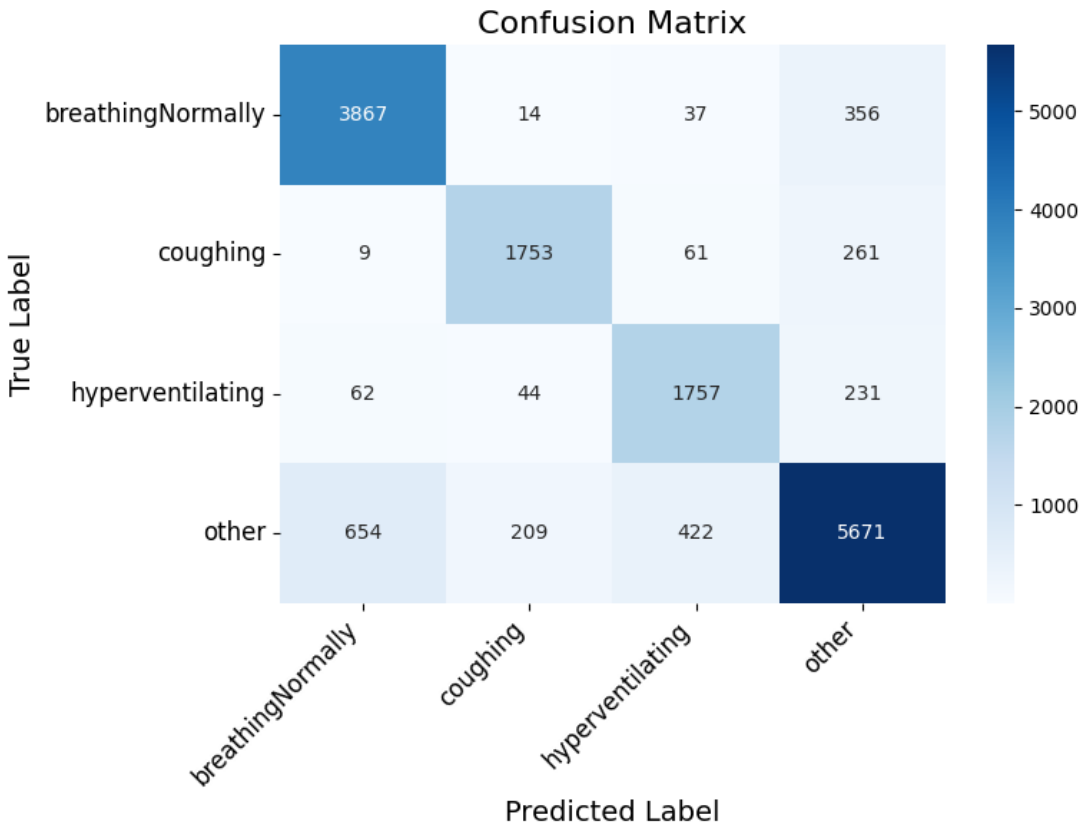


Figure 4.2: Confusion matrix for Social Signals

Table 4.2: Classification Metrics for Each Class

Class	Precision	Recall	F1-Score
breathingNormally	0.842	0.905	0.872
coughing	0.868	0.841	0.854
hyperventilating	0.772	0.839	0.804
other	0.870	0.815	0.842
Mean	0.838	0.850	0.843
Accuracy (Overall)		0.838	

The overall performance of the model is notable. The model achieved an overall accuracy of 83.8% for this multi-class classification problem with 4 distinct classes.

Table 4.2 shows that the model performs well for the class *breathingNormally*, achieving a precision of 0.842, a recall of 0.905, and an F1-score of 0.872. This indicates the model's strong ability to identify this class with minimal misclassification. Similarly,

the *coughing* class shows good performance, with a precision of 0.868, recall of 0.841, and F1-score of 0.854, demonstrating the model's balanced behaviour for this activity.

The confusion matrix 4.2 provides further insights into the model's behaviour. Misclassification are more prominent for the *hyperventilating* class, which achieves the lowest F1 score of 0.804. This is likely due to significant overlaps with the classes *breathingNormally* and *other*, as seen in the confusion matrix, where 62 samples of *hyperventilating* are misclassified as *breathingNormally* and 231 samples as *other*. These misclassification highlight the challenge of distinguishing between activities with overlapping respiratory patterns.

Table 4.2 also shows that the *other* class performs moderately well, with a precision of 0.870, recall of 0.815, and F1-score of 0.842. However, this class encompasses a wide variety of activities, which leads to frequent misclassification, as evidenced by 356 samples of *breathingNormally* and 261 samples of *coughing* being misclassified as *other* in the confusion matrix. This suggests that the broad nature of the "other" category contributes to its confusion with other classes.

4.4 Results of Sleep Analysis

4.4.1 Sleep-Wake Cycle Classification

The sleep-wake cycle classification model achieves an overall accuracy of 82% for the binary classification task of distinguishing between sleep and awake states, as shown in Table 4.3. While this performance is solid, there are notable differences in how the model handles each class.

For the "Sleep" class, the model achieves a high recall of 0.95, indicating that the vast majority of actual sleep instances are correctly identified. However, the precision for the "Sleep" class is lower at 0.75, meaning that some awake segments are misclassified as sleep. On the other hand, the "Awake" class demonstrates a high precision of 0.93, ensuring that most awake predictions are accurate, but its recall of 0.69 indicates that a significant number of awake instances are misclassified as sleep.

The F1-score for both classes is 0.84, reflecting a balanced trade-off between precision and recall for the overall task. This balance suggests that the model performs reliably across the dataset, though the lower recall for the "Awake" class highlights a specific area where the model could improve.

These results imply that the model effectively identifies sleep states but has a tendency to misclassify some awake instances. Indeed this is demonstrated in Figure 4.3, , where there are 3,691 cases of awake instances being misclassified as sleep and 619 cases of sleep instances being misclassified as awake, despite the dataset containing a similar total number of awake and sleep instances.

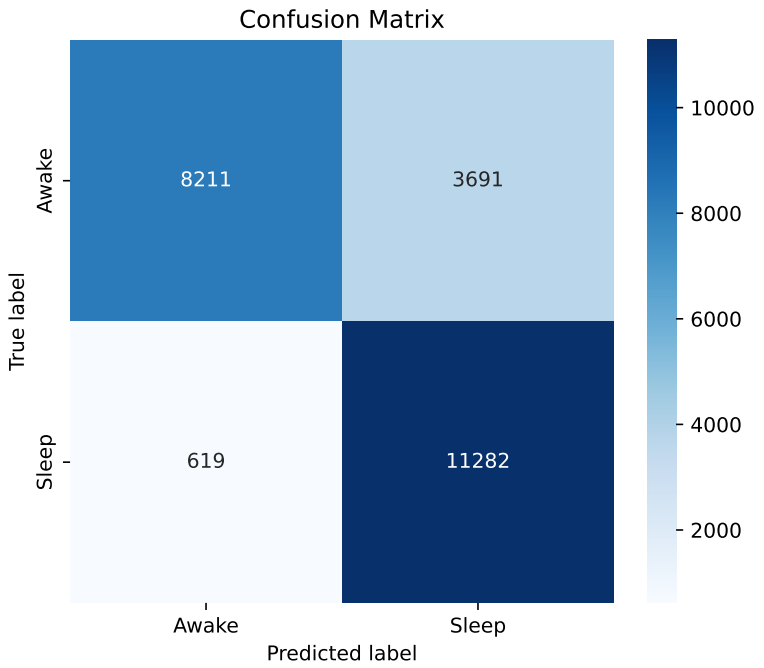


Figure 4.3: Confusion matrix for sleep-wake cycle classification

Class	**Precision**	**Recall**	**F1-Score**
Sleep	0.75	0.95	0.84
Awake	0.93	0.69	0.84
Mean	0.84	0.82	0.84
Accuracy (Overall)		0.82	

Table 4.3: Classification Metrics for sleep-wake cycle classification

Using the sleep-wake classification model, we analysed the sleep efficiency of six users who were initially set aside as the test set from the 34 users available in the sleep questionnaire dataset. Recall that we selected two users from each category of sleep quality: poor, average, and good. The results, summarized in Table 4.7, show that most users have a sleep efficiency between 75% and 90%, indicating that users spent a majority of their time in bed asleep. However, one user was awake for approximately a quarter of the night.

These results are visualised in Figure 4.4, where red shading indicates periods of sleep and green shading indicates periods of wakefulness. From the figure, we observe numerous small windows classified as awake by the model. This classification is likely erroneous, as it seems improbable that all six users experienced frequent brief awakenings throughout the night. This issue may stem from several factors, including the quality of the training data and annotations.

The training data were not collected in a controlled environment, raising concerns about the consistency and accuracy of the dataset. Additionally, the annotations, based on user-reported times for falling asleep and waking up, may lack precision, introducing further uncertainty. Another significant limitation is the small dataset size: with only 34 users available and 6 used for testing, the training set of 28 users is insufficient for the model to learn generalized patterns effectively.

A notable observation is the extended period of wakefulness predicted for user 35, as seen in Figure 4.4. The model predicts with high confidence that the user was awake for a prolonged duration between timesteps 200,000 and 300,000. This could plausibly indicate that the user got up during the night; however, without further details from the sleep questionnaire, this cannot be confirmed.

To improve future analyses, it is essential to acquire better-annotated data, a larger dataset, and more detailed sleep questionnaire responses. These enhancements would likely address the current model's limitations and improve its predictive accuracy.

User	**Sleep Efficiency (%)**
27	86.32
14	79.76
42	87.75
35	76.76
17	82.91
36	86.72

Table 4.4: Sleep Efficiency for Each User

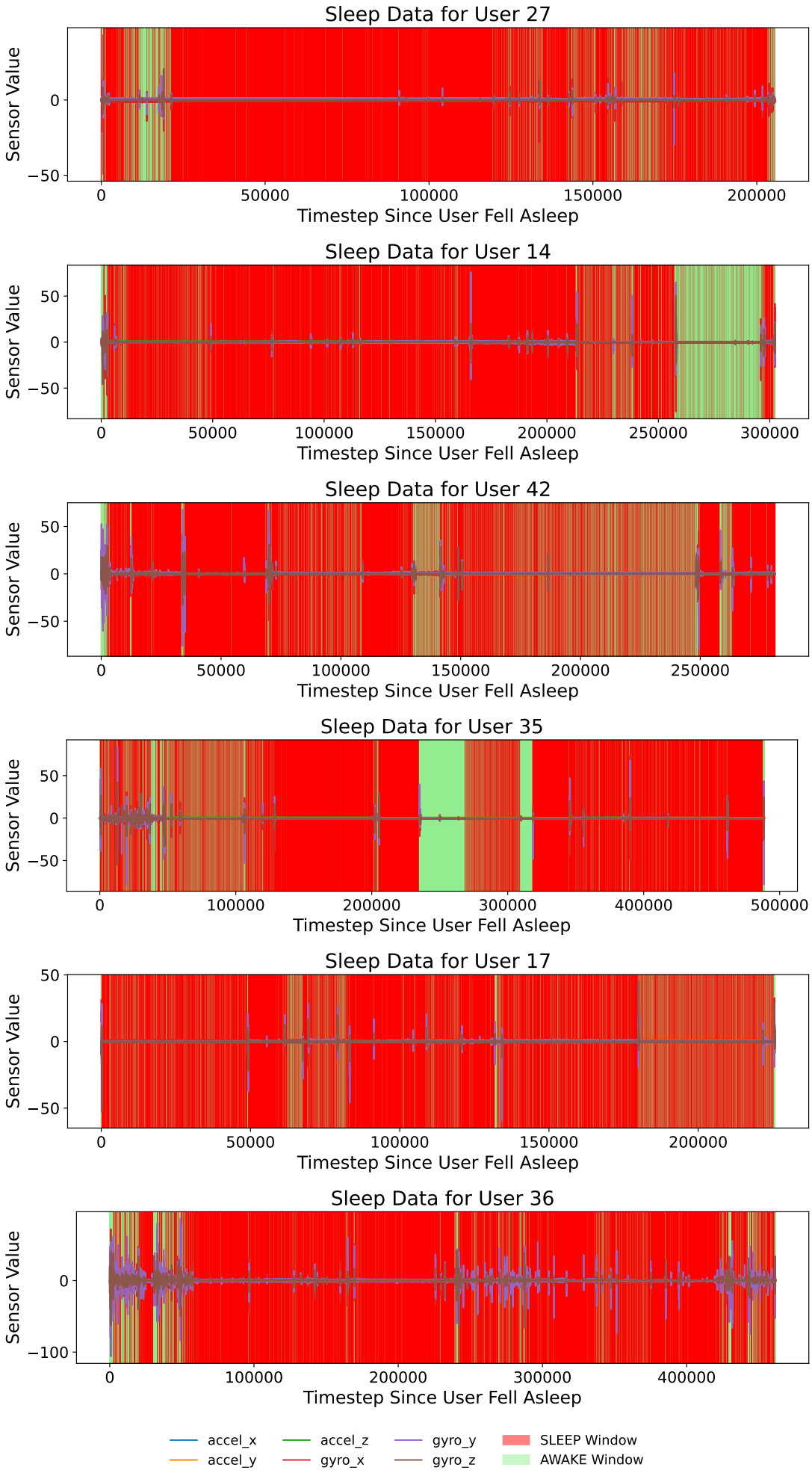


Figure 4.4: Sleep data for 6 users with sleep-awake analysis

4.4.2 Position Classification for Positional Changes

The position classification model achieves an overall accuracy of 99% for distinguishing between different lying positions: on the back, stomach, left, and right. This high performance is expected, as the accelerometer data (x, y, z axes) provides distinctive patterns for each position. Furthermore, as shown in Table 4.5, the precision and recall for all classes are consistently around 99%, demonstrating the model's capability to accurately classify each position. Figure 4.5 further supports this, highlighting that misclassification are rare across all classes.

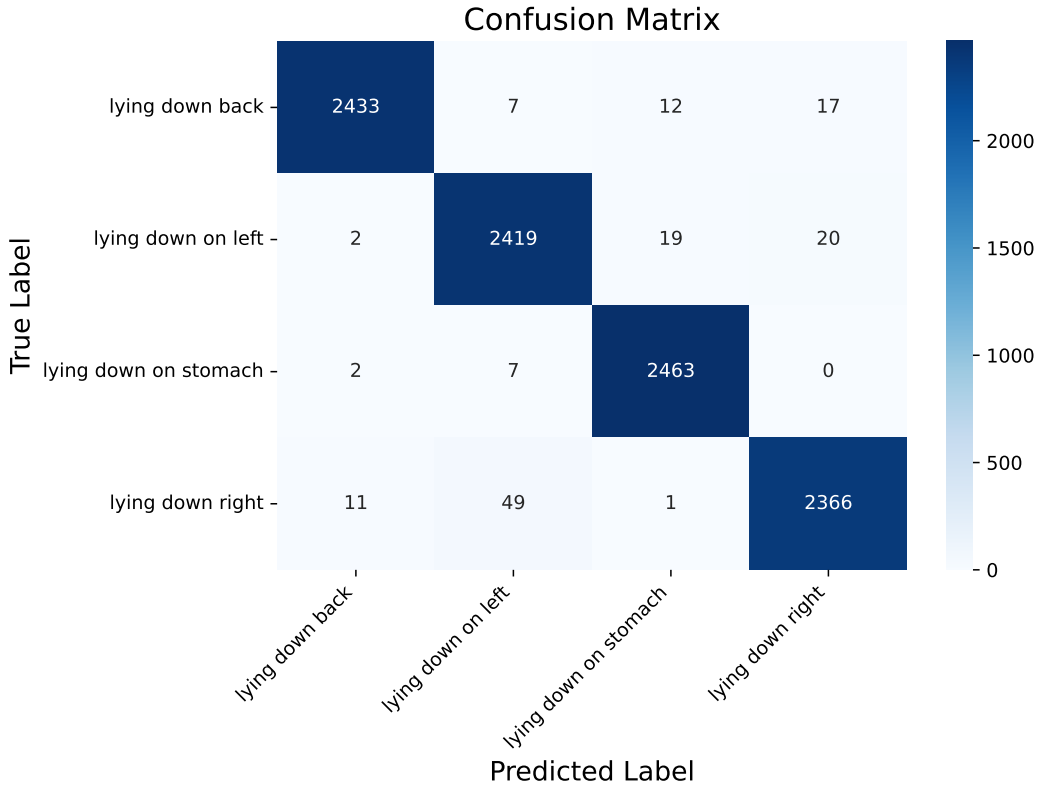


Figure 4.5: Confusion matrix for position classification

Class	Precision	Recall	F1-Score
Lying down back	0.99	0.99	0.99
Lying down on left	0.97	0.98	0.98
Lying down on stomach	0.99	1.00	0.99
Lying down right	0.98	0.97	0.98
Mean	0.98	0.99	0.99
Accuracy (Overall)		0.99	

Table 4.5: Classification Metrics for position classification

Using the position classification model, we counted the number of position changes

for the six users in the test set. Recall that there are two users from each sleep quality category: poor, average, and good. The results, as shown in Table 4.6, indicate that users generally experienced a single-digit number of position changes during the night. However, two users exceeded this, with 10 and 12 position changes.

The results are visualised in Figure 4.6, where each shading represents a different sleep position. From the figure, we observe that users tend to remain in one position for extended periods before changing positions. Manual inspection of the plots suggests that position changes align with points where the patterns in the three axes of accelerometer readings shift noticeably, which corresponds to how the model identifies these changes.

However, this is not always the case. For instance, for User 42, the algorithm reports 12 position changes, but visual inspection of the figure suggests there may only be three distinct changes. One possible explanation is that the algorithm detected subtle movements that are not easily discernible in the coarse-grained visualization, and these might represent actual position changes. Alternatively, the algorithm may have incorrectly identified changes due to noise in the sensor data, leading the model to predict a few windows of position change even when the user's position did not fully change.

The model for detecting the specific position a user is lying in performs well, achieving 99% accuracy, but the algorithm for counting position changes could be further refined. Future improvements could include stricter thresholds for identifying sustained position changes, incorporating additional sensor modalities such as gyroscope data to validate changes, or using post-processing techniques like sequence smoothing or aggregation over multiple frames to reduce noise and eliminate false positives. These refinements would enhance the algorithm's ability to accurately distinguish between significant and minor movements, leading to more reliable results.

User	**Position Changes**
27	4
14	6
42	12
35	10
17	5
36	3

Table 4.6: Position Changes by User

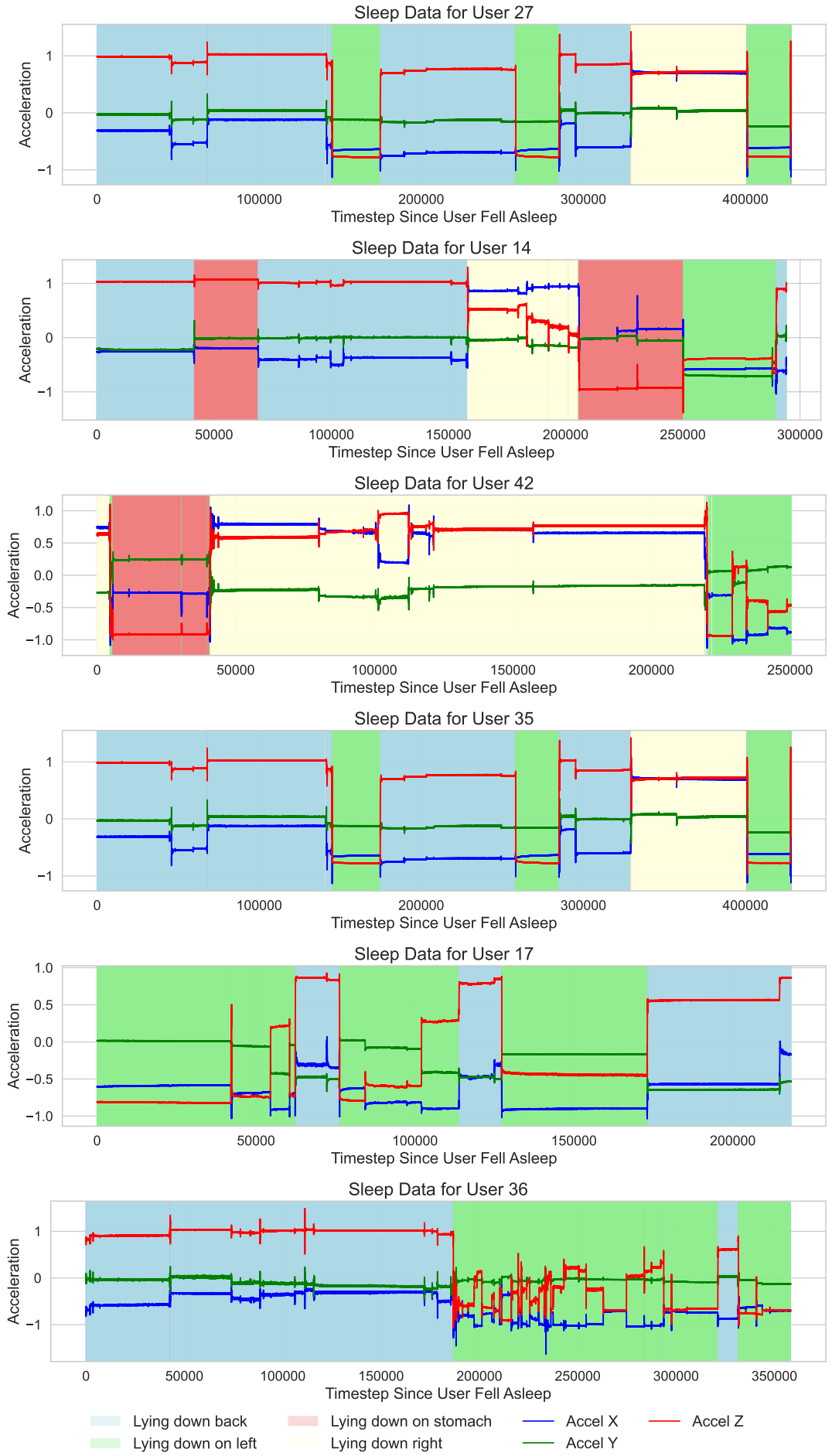


Figure 4.6: Sleep data for 6 users with position change analysis

4.4.3 Sleep Quality Index

Using the sleep efficiency and number of position changes obtained, we combined this data with the sleep quality ratings provided by the corresponding users. These results are presented in Table 4.7. From this data, we created Figure 4.7, which explores potential relationships. From the figure, we observe no discernible relationship between sleep quality and sleep efficiency. However, for sleep quality versus the number of positional changes, there appears to be a positive relationship, where an increase in positional changes correlates with higher sleep quality.

This observation is counterintuitive and contradicts findings from studies such as Zhang et al. (2022), which show that an increase in positional changes during sleep is associated with poorer sleep quality, characterized by increased morning and evening sleepiness and greater fatigue.

The discrepancy between our analysis and theirs may be attributed to differences in the data. Zhang et al. (2022) analysed 13 subjects over 8 hours of sleep for 15 nights, whereas our study uses data from 6 subjects over a single night, with varying durations. This limited dataset means that irregularities in a single night's sleep for our participants could skew results.

Additionally, there are inherent limitations in our analysis. The sleep quality ratings provided by users are subjective and lack consistency, as users were not given specific constraints or criteria for their evaluations. For example, Zhang et al. (2022) measured specific factors such as morning and evening sleepiness and fatigue, while our ratings are based on broad categories of "poor," "average," or "good." Furthermore, the quantity and quality of our data are insufficient. Users estimated when they fell asleep and woke up, which introduces inaccuracies in annotations and further affects the analysis.

To improve this analysis in the future, data should be collected over multiple nights to account for irregularities. The sleep questionnaire should be expanded to include more detailed questions, such as whether the user woke up during the night and for how long, or more nuanced scales beyond the current "poor," "average," and "good" ratings. These improvements would enable better annotations for training a more accurate sleep efficiency model and increase the dataset size, ensuring more representative results.

Due to these limitations and the lack of meaningful relationships in our analysis, the sleep analysis algorithm in our app relies on a custom formula to determine sleep quality. This formula is further described in Section 3.6.2.3.

User	Sleep Quality	Sleep Efficiency (%)	Position Changes
27	average	86.32	4
14	average	79.76	6
42	good	87.75	12
35	good	76.76	10
17	poor	82.91	5
36	poor	86.72	3

Table 4.7: Sleep Efficiency and Position Changes by User Rating

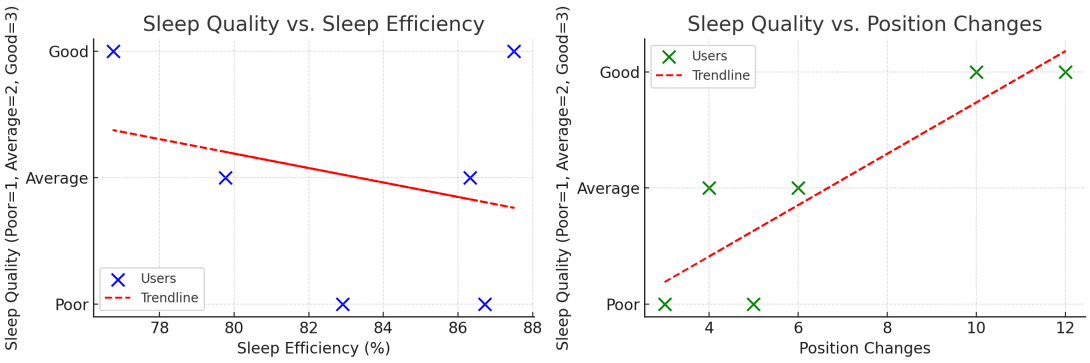


Figure 4.7: Trend between sleep quality, sleep efficiency and position changes

4.5 Android App Performance

4.5.1 Latency

Our app experiences several sources of latency, with the largest contributor being the sliding window implementation required for activity classification. The machine learning model requires 50 data points before it can process and classify an activity. Given the Respeck sensor's data collection rate of 25 Hz, this introduces a minimum delay of 50/25=2 seconds to fill the sliding window.

However, this is only the base latency. If a user starts performing an activity partway through a window, the initial contaminated window is likely to result in an incorrect classification. The user would then need to wait for another 2 seconds for a new window to fill and generate an accurate result. This worst-case scenario creates a delay of up to 4 seconds for classifying an activity.

The time taken to execute the machine learning model itself is negligible in comparison to the delays caused by the sliding window process. Despite this, the app was not designed for real-time feedback but rather for tracking user behavior throughout the day. Users are already aware of the activities they are performing, so immediate classification is less critical for the intended use case.

4.5.2 Power Consumption

After running the app for an hour, we measured an average power consumption of 360mAh. This is typical for apps that perform frequent machine learning inferences. However, given that the app is designed for continuous daily use, the current consumption rate poses a significant challenge. A typical smartphone battery has a capacity of approximately 4500mAh, meaning the app would use around 8% of the battery per hour. Over several hours, this would deplete the majority of the user's battery, leaving insufficient capacity for other applications. To make the app practical for daily use, power consumption must be reduced to around 150mAh. This would require significant optimizations, including:

- Reducing TensorFlow Model Complexity: By lowering the precision of weights

and activations from 32-bit to 16-bit or 8-bit floats and pruning insignificant weights, we can decrease computational demands.

- Reducing Inference and Communication Frequency: Processing data in real time increases CPU usage, memory bandwidth, and power consumption. Lowering the frequency of data transmission would alleviate these issues but may increase latency. Running the inference less frequently could further reduce power consumption at the cost of classification granularity.
- Using Cloud-based Machine Learning Models:

Since the app must continuously sample sensor data, the CPU cannot enter low-power states. Batch processing could mitigate this but would increase latency further, presenting a trade-off that needs careful consideration.

4.5.3 Memory Usage

The app's average memory consumption was 255MB, which is broken down into the following categories:

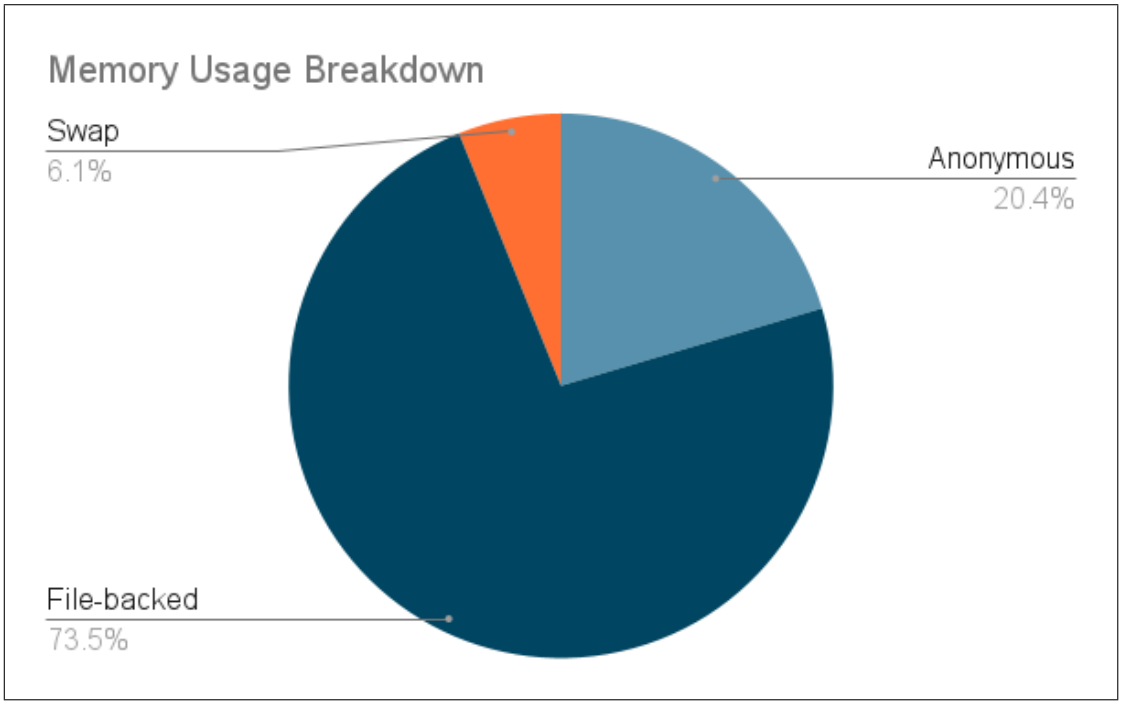


Figure 4.8: Memory usage pie chart

- 60MB Anonymous Memory: This includes stack and heap memory and is relatively low for an app of this type.
- 180MB File-Backed Memory: This memory is used for memory-mapped files and other assets. It is higher than expected, and reducing this number could not only improve memory efficiency but also decrease battery consumption by minimizing active RAM usage.

- 15MB Swap Memory: This occurs when insufficient RAM forces memory to be stored on disk. Ideally, swap memory usage should be 0MB, as frequent read/writes to long-term storage are inefficient and increase both processing time and battery consumption. Reducing file-backed memory requirements, such as by decreasing the model size, would help eliminate the need for swap memory.

Optimizing memory and model size, along with fine-tuning resource utilization, would address these performance bottlenecks and make the app more efficient for long-term use.

Chapter 5

Conclusions and Future Work

5.1 Summary of Your Project and Reflection

Reflections: Although we worked effectively as a team and had a good development process, there are still a few areas of improvement. We only started performance testing relatively late in the development process. This meant we didn't have time to explore other ways of improving our apps heavy battery consumption.

The app and machine learning teams worked well together, however there were often problems integrating machine learning "Tflite" models with the Kotlin code due to the very specific data types they wanted for their input. We could have agreed before hand to a common interface between app and model because it led to lengthy delays fixing the integration errors each time we tried a new model.

5.2 Areas for Future Works

5.2.1 Improvements:

5.2.1.1 Model Improvements

Data Quality: The dataset suffers from poor quality and limited volume, with significant noise in the recorded accelerometer signals. This adversely impacts the reliability of model training and validation, emphasizing the need for cleaner and more representative datasets.

Class Overlap and Imbalance: Some activities, such as *LyingLeft* and *LyingRight*, or *hyperventilating* and other social signals, showed notable misclassifications due to overlapping features. Additionally, imbalanced class distributions affected the model's ability to generalise to under represented activities.

5.2.1.2 App Improvements

Branding: Although the minimalist design makes it easy to navigate and interpret, if we were to release it, we would want to develop a more distinctive brand.

Performance: At the moment it would take a phone with a relatively strong battery to run our app all day. Using more power efficient models could improve our battery performance. However using cloud machine learning models would massively improve power efficiency, but would require the user to maintain an internet connection. A hybrid system which allows the user to use the cloud when connected to the internet and use local models when not would be optimal. However this was too complex to implement in our limited timescale.

5.2.2 Extensions:

5.2.2.1 Future work on data

Improving Data Collection: Future efforts should focus on collecting larger, more diverse, and cleaner datasets to mitigate the impact of noise and improve the model's robustness.

Incorporating Gyroscope Data and physiological signals: Adding gyroscope data and combining accelerometer data with other physiological signals could help distinguish overlapping activities more effectively.

5.2.2.2 Additional App Features

Multi-Series Line Graph: Users may find it informative to allow for direct comparison of multiple activities over the same day. It could allow the user to more easily identify events in their day i.e. an increase in sitting and a decrease in walking may help the user identify this is when they reached the library etc.

Further Sleep Analysis Visualisation: At the moment the data for sleep analysis is not saved between sessions. It would a great feature to save the sleep data and allow the user to visualise it as a barchart so they could track their sleep quality overtime. This would be an excellent way to add more health tracking functionality to our app.

Add "Forgot Password" feature: Currently there is no way for the user to recover/change their password other than asking an administrator to do this for them. This could be automated by adding a forgot password feature.

User Selected Targets: We could allow the user to add personalised targets for activity levels or sleep quality. This is a fantastic feature most health apps include as it improves interactivity and encourages the user to improve themselves over time.

Bibliography

Basant Adel, Asmaa Badran, Nada E Elshami, Ahmad Salah, Ahmed Fathalla, and Mahmoud Bekhit. A survey on deep learning architectures in human activities recognition application in sports science, healthcare, and security. In *The International Conference on Innovations in Computing Research*, pages 121–134. Springer, 2022.

Damien Bouchabou, Sao Mai Nguyen, Christophe Lohr, Benoit LeDuc, and Ioannis Kanellos. A survey of human activity recognition in smart homes based on iot sensors algorithms: Taxonomies, challenges, and opportunities with deep learning. *Sensors*, 21(18):6037, 2021.

Nishanth Adithya Chandramouli, Sivaramakrishnan Natarajan, Amal H Alharbi, Subhash Kannan, Doaa Sami Khafaga, Sekar Kidambi Raju, Marwa M Eid, and El-Sayed M El-Kenawy. Enhanced human activity recognition in medical emergencies using a hybrid deep cnn and bi-directional lstm model with wearable sensors. *Scientific Reports*, 14(1):30979, 2024.

Xin Cheng, Lei Zhang, Yin Tang, Yue Liu, Hao Wu, and Jun He. Real-time human activity recognition using conditionally parametrized convolutions on mobile and wearable devices. *IEEE Sensors Journal*, 22(6):5889–5901, 2022.

Atiqul Islam Chowdhury, Mohsena Ashraf, Ashraful Islam, Eshtiak Ahmed, Md Saroar Jaman, and Mohammad Masudur Rahman. Hactnet: an improved neural network based method in recognizing human activities. In *2020 4th international symposium on multidisciplinary studies and innovative technologies (ISMSIT)*, pages 1–6. IEEE, 2020.

Seungeun Chung, Jiyoun Lim, Kyoung Ju Noh, Gague Kim, and Hyuntae Jeong. Sensor data acquisition and multimodal sensor fusion for human activity recognition using deep learning. *Sensors*, 19(7):1716, 2019.

Marcus Edel and Enrico Köppe. Binarized-blstm-rnn based human activity recognition. In *2016 International conference on indoor positioning and indoor navigation (IPIN)*, pages 1–7. IEEE, 2016.

GeeksforGeeks. Mvc design pattern. https://www.geeksforgeeks.org/mvc-design-pattern/, n.d. Accessed: 2025-01-14.

Masaya Inoue, Sozo Inoue, and Takeshi Nishida. Deep recurrent neural network for mobile human activity recognition with high throughput. *Artificial Life and Robotics*, 23:173–185, 2018.

Raman Maurya, T Hui Teo, Shi Hui Chua, Hwang-Cherng Chow, and I-Chyn Wey. Complex human activities recognition based on high performance 1d cnn model. In *2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 330–336. IEEE, 2022.

Sen Qiu, Hongkai Zhao, Nan Jiang, Zhelong Wang, Long Liu, Yi An, Hongyu Zhao, Xin Miao, Ruichen Liu, and Giancarlo Fortino. Multi-sensor information fusion based on machine learning for real applications in human activity recognition: State-of-the-art and research challenges. *Information Fusion*, 80:241–265, 2022.

Anguita Davide Ghio Alessandro Oneto Luca Reyes-Ortiz, Jorge and Xavier Parra. Human Activity Recognition Using Smartphones. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C54S4K.

Eirik S Skarpsno, Paul J Mork, Tom IL Nilsen, and Andreas Holtermann. Sleep positions and nocturnal body movements based on free-living accelerometer recordings: association with demographics, lifestyle, and insomnia symptoms. *Nature and Science of Sleep*, 9:267–275, 2017. doi: 10.2147/NSS.S145777.

Chih-Ta Yen, Jia-Xian Liao, and Yi-Kai Huang. Human daily activity recognition performed using wearable inertial sensors combined with deep learning algorithms. *Ieee Access*, 8:174105–174114, 2020.

Yuan Zhang, Aiping Xiao, Tianhao Zheng, Huafei Xiao, and Ruiyan Huang. The relationship between sleeping position and sleep quality: A flexible sensor-based study. *Sensors*, 22(16):6220, 2022.