An Internet of Things real-time human activity recognition system based on Inertial Motion Unit (IMU) and cloud machine learning

Steven Chang(s1992413)
Ke Shen(s1965695)
Yanchen Fan(s1918258)

PDIoT Coursework 3 (2022-23)
Group R Report
School of Informatics
University of Edinburgh

2023

Abstract

In this project, we developed an Internet of Things (IoT) real-time human activity recognition system based on inertial motion units (IMUs) and cloud machine learning. The system uses Thingy:52 or respack, two IoT sensor development kits, to collect motion data. We use these two devices to collect motion data and use these data to train a machine learning model on azure server, which achieved an accuracy of 98% in recognizing human motion, and users will also use these two devices(both or one of them) to collect their motion data. Beside, an Android application was created to connect to the IoT devices mentioned above in real-time and collect user‘s real-time motion data, which is then sent to the trained machine learning model on the Azure cloud server to determine the user's current motion. The Android app will displays the user's current motion and also allows users to view historical motion data and access multi-user functionality. Users can log in to the app with their Google account. The system has potential applications in health monitoring, fitness tracking, and other areas.

Table of Contents

Chapter 1

Introduction

1.1 Project aims

With the continuous development of Internet of Things technology and wireless communication technology, Internet of Things devices have played an increasingly important role in helping human life. Among them, the three aspects of sports monitoring, elderly care, and health detection are the three most important parts of the Internet of Things in human life. Our project aims to cover these three aspects at the same time, that is, to develop an IoT motion recognition system that can obtain user motion data in real time through two IoT sensors (Thingy and Respack), and build on the cloud (Microsoft Azure) A machine learning model to identify these data and return the user's current motion state, such as walking, sitting, lying down, etc. The goal of our plan is to enhance the application value of the Internet of Things in these three aspects, and to realize high-precision models, easy-to-use, and elderly-friendly real-time motion detection tools, which requires our models to have extremely high test accuracy, and our Android The software should be easy to use, provide a very friendly human-computer interaction interface (especially for the elderly), and develop available functions in the software based on the returned data from the model. We believe that covering these three aspects that are closely related to human health at the same time can bring considerable value to human health care, and has great potential application value in these aspects.

1.2 Brief description of the method adopted

As mentioned before, we chose to use Microsoft Azure as our online model server, and we used Google Cloud Firebase as the backend server to store our user data as well as the historical logs for the Android application.

1.3 List the physical activities used in the classification

In our project, we detected different types of sports, we selected the most representative 14 types of motions in daily life, the following is the list:

- Sitting (upright, bent forward, leaning backward)
- Standing
- Lying down (left side, right-side, on the back, on the front)
- Walking
- Running
- Ascending and descending stairs
- Desk work
- General movement (sudden turns, bending down, getting up from chairs, or anything else not listed above)

We think these 14 types of motions cover the classic rest scene, office scene and sports scenes, it can cover 80% of daily sports types. The last item, General Movement, also provides options for other motions that are not in the list.

1.4 Summary of results

Chapter 2

Literature survey

2.1 A review of the state-of-the-art for human activity recognition algorithms

Human activity recognition (HAR) is the task of identifying and classifying the physical actions and behaviors of a person or group of people from data. This is an important problem in many applications, including healthcare, sports, and smart homes. In recent years, there has been significant progress in the development of algorithms for HAR using sensor data, and a variety of approaches have been proposed to address this problem.

One common approach to HAR is to use machine learning algorithms to classify activities based on features extracted from the sensor data. This typically involves training a classifier on a dataset of labeled sensor data, such as data collected from wearable devices or sensors embedded in a smart home. Many different machine learning algorithms have been applied to this problem. For example, Xu et al.[1] introduced a way to classify human activities using a random forest (RF) model. They established a random forest model for user human activity recognition. Through the design and analysis of the algorithm, it was proved that the overall accuracy The rate can reach about 90%.

Another approach to HAR is to use deep learning algorithms, which are able to learn complex patterns in data automatically without the need to manually extract features. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are particularly well-suited for this task, as they are able to effectively process data from two-dimensional arrays and sequential data, respectively. In recent years, there has been a significant amount of research on the use of deep learning algorithms for HAR, and many state-of-the-art methods rely on this approach. For example, Pienaar et al.[2] outline the design of long short-term memory (LSTM) architecture models for human activity recognition applications. In the first 500 epochs of training, the accuracy rate is over 94%, and the loss rate is less than 30%. At the same time, Ronao et al.[3] also proposed a deep convolutional neural network (convnet) to perform efficient and effective HAR using smartphone sensors by exploiting the inherent characteristics of

activities and one-dimensional time series signals, while providing an automatic and data Adaptive methods extract robust features from raw data. An overall performance of 94.79% is achieved on the test set using raw sensor data. Zebin et al.[4] also proposed a feature learning method where the input is a multi-channel time-series signal acquired from a set of wearable inertial sensors and the output is a predefined human activity. The method deploys a convolutional neural network (CNN) to automatically perform feature learning from raw input in a systematic manner. They found that CNNs achieved significant speedups in computing and deciding on the final class and achieved marginal improvements in overall classification accuracy compared to baseline models such as support vector machines and multilayer perceptron networks.

Other approaches to HAR include using video, image data, and 3D skeletal features to identify human activities. For example, Uddin et al.[5] proposed a human activity recognition approach using robust translation and scale-invariant body silhouette features recurrent neural network. Their proposed method demonstrated superiority by achieving an average recognition rate greater than 98% on private and public datasets. advantage over the other methods, which can yield about 95% recognition. Meanwhile, Piyathilaka et al.[6] also proposed a human activity detection model using only 3-D skeletal features generated from an RGB-D sensor (Microsoft Kinect TM). They used Gaussian mixture modes based on hidden Markov models to judge human activities, and finally achieved 78% accuracy. Afsar et al.[7] also proposed an automatic human detection and action recognition system using hidden Markov models and bag-of-words. The algorithm is capable of robust detection in cluttered environments and severe occlusions.

On an additional level, a survey of human activity recognition using wearable sensors is done by Lara et al.[8], who first present a general architecture and a description of the main components of any HAR system. They also proposed a two-level taxonomy based on the learning method (supervised or semi-supervised) and response time (offline or online). Then, the main problems and challenges are discussed, along with the main solutions for each problem and challenge. Attal et al.[9] review different classification techniques for recognizing human activities from wearable inertial sensor data. Four supervised classification techniques, k-nearest neighbors (k-NN), support vector machines (SVM), Gaussian mixture models (GMM), and random forests (RF), and three unsupervised classification techniques, k- Performance of mean, Gaussian Mixture Model (GMM), and Hidden Markov Model (HMM) in terms of correct classification rate, F-measure, recall, precision, and specificity. They found that the k-NN classifier provided the best performance, while the HMM classifier was the classifier that gave the best results among unsupervised classification algorithms. Zeng et al.[10] developed a convolutional neural network (CNN) based approach that can capture the local dependence and scale invariance of signals, as it has been shown in the fields of speech recognition and image recognition. Furthermore, an improved weight sharing technique called partial weight sharing is proposed and applied to accelerometer signals for further improvement.

Overall, the state-of-the-art for human activity recognition algorithms using sensor data is constantly evolving, with new approaches and techniques being developed and refined on an ongoing basis.

Chapter 3

Methodology

3.1 Description of the system and its implementation

The proposed system is composed of hardware, software, and cloud-based components. During the initial stage of development, data was collected at a sampling rate of 25 Hz using the Respeck and Thingy sensors. This data was subsequently used to train a machine learning model for human activity recognition (HAR), which was packaged as a TFLite file using TensorFlow Lite, and integrated locally within the Android application.

In order to provide more flexibility for the users, the human activity recognition (HAR) functionality was migrated to Azure by packing the model into Open Neural Network Exchange (ONNX). In addition, the software features user authentication and historical log storage capabilities through integration with Google Cloud Firebase, resulting in the use of two cloud-based services as the backend for the mobile application.

The data flow within the system is as follows: raw data is collected by the Respeck and/or Thingy sensors, and streamed to the Android application via Bluetooth Low Energy (BLE) utilizing the `RXAndroidBLE` library. The software receives this raw data, converts it into a `List` of `Float` numbers, and stores it in a `MutableList`. Once the `MutableList` reaches a size of 50 frames, equivalent to 2 seconds of collected data, it is converted into a fixed-size `List` and packaged as a JSON object along with labels indicating the source sensor. The JSON is then sent through an HTTP request to Azure Functions, a serverless solution on the Azure cloud platform.

Azure Functions receives the JSON, runs the machine learning model in ONNX format, and returns an index in response to the software. This index ranges from 0 to 13, indicating the classified movement type. The software then uses this index to retrieve corresponding text and icon information for reflecting on the user interface. The retrieved text, timestamp, original data, and sensor labels are then packaged in a hash map and stored in Google Firestore as historical logs through an HTTP/POST request. Users can access their historical data through the ”Historical Data” page, by providing a specific time range, and the data is retrieved through an HTTP/GET request and visualized on the interface in the form of a pie chart.

The following sections of this report will provide more detailed information on the various aspects of the system, including the hardware components, wireless communication, machine learning algorithm, user interactions, software design, and testing conducted prior to deployment of the application.

3.2 Hardware and firmware

There are a total of three hardware components used in this project, namely a mobile phone and two IoT sensors. They will be introduced separately below.

For the mobile phone part, this project is developed for Android phones, and only Android phones are available, so you need a phone that supports NFC (Near-field communication) and BLE (Bluetooth Low Energy). Almost all phones these days have this feature, so if yours is less than 5 years old it should be fine. Regarding the firmware part, the minimum SDK limit we set is 23, and the target SDK is 31, which means you can run the software even if you are using Android 6.0. But please note that we don't recommend it, and we can't be sure that it can be used, because our test machines are Samsung phones and Huawei phones based on Android 10. So please try to make sure that you use a phone with a version higher than Android 10.

Regarding the IoT sensor part, the sensors we mainly use are Nordic Thingy:52 and Respack. Thingy:52 is a compact multi-sensor IoT platform that revolves around The nRF52832 System-on-Chip builds a versatile Bluetooth 5.3 SoC that supports Bluetooth Low Energy, Bluetooth mesh, and NFC. Respeak, also a Bluetooth Low Energy IoT sensor, was developed by the University of Edinburgh and was initially used to help nursing professionals diagnose a patient's health condition and also predict the deterioration of the condition. But this sensor can also get 3-axis gyroscope acceleration information, so we will use it as one of the sensors. As for the firmware part, both ThingY and respeck have built-in firmware, so you don't need to manually flash the firmware, just use our Android application and follow the prompts to connect. It's worth mentioning that using Thingy and respeck at the same time can get better detection results, but we also allow you to use only one sensor.

3.3 Wireless communication

Regarding the wireless communication part, as mentioned earlier, the main technologies we use to connect with sensors are NFC (Near Field Communication) and Bluetooth Low Energy (BLE). We use the NFC function to pair the sensor with the mobile phone, and use BLE to ensure the efficient transmission of information between the mobile phone and the sensor. In order to go online, wifi and cellular network technologies are also the main technologies applied in the connection process between the online Azure server(database) and mobile phone.

Bluetooth Low Energy (BLE) is a wireless communication technology that is designed to be energy-efficient and low-power, making it ideal for use in small, battery-powered devices such as fitness trackers, smartwatches, and other IoT devices. BLE uses the

same 2.4 GHz radio frequency as classic Bluetooth, but it uses a completely different protocol stack and a different set of commands for communication.BLE is based on a star topology, where a central device (such as a smartphone) communicates with one or more peripheral devices (such as a fitness tracker). The central device is responsible for managing the connection and sending commands to the peripheral devices, while the peripheral devices respond to commands and send data back to the central device. One of the key features of BLE is its ability to operate in low-energy mode, where it consumes very little power and can run for months or even years on a small coin cell battery. This is achieved by using a sleep mode where the device is only active for short periods of time to send and receive data, and by using a low data rate (1Mbps) for communication. BLE also supports a number of security features such as authentication and encryption to protect against unauthorized access and tampering. BLE can be used in various application such as Smart home devices, Wearable devices, Health care devices, Industrial Automation, etc.

Near Field Communication (NFC) is a short-range wireless communication technology that enables the exchange of data between devices when they are brought into close proximity, typically within a few centimeters of each other. NFC operates at 13.56 MHz and uses magnetic field induction to communicate between devices. NFC is based on a peer-to-peer communication model, where two devices can exchange data when they are brought into close proximity. One device acts as the initiator, while the other acts as the target. The initiator sends a command to the target, which responds with the requested data. In terms of data exchange, NFC can be used to share photos, videos, business cards, and other types of data between two devices. This is done by bringing the devices close together and then initiating the data transfer. NFC also supports a number of security features such as authentication and encryption to protect against unauthorized access and tampering.

Wi-Fi is a wireless networking technology that allows devices such as computers, smartphones, and tablets to connect to the internet or communicate with each other without the need for physical cables or wires. It uses radio waves to transmit and receive data, and operates on a specific frequency band (2.4GHz or 5GHz). The most common way to connect to a Wi-Fi network is through a router, which acts as a hub for the devices on the network. Wi-Fi has become a widely-used standard for internet connectivity in homes, offices, and public spaces.

Cellular technology is a method of communicating wirelessly over a wide area, typically through the use of radio waves. It is the backbone of modern mobile communication systems, such as mobile phones, tablets and other wireless-enabled devices. A cellular network consists of a number of base stations, each of which covers a specific geographic area, known as a cell. Mobile devices communicate with the closest base station, which then relays the information to other base stations as the device moves from one cell to another. This allows for seamless communication even as the device moves across the network.

3.4 Machine Learning methods for activity recognition

3.4.1 Overall

In this study, we aimed to recognize 14 types of human activities in real-time using data from the Nordic Thingy:52 and RESpeck sensors. To do this, we collected and processed the data, trained two random forest models, and tested the models using the leave-one-out cross-validation (LOSOXV) method.

3.4.2 Data Collection and Preprocessing

The data was collected using the Nordic Thingy:52 and RESpeck sensors, which are equipped with accelerometers and other sensors that can measure various types of data. The sensors were worn on the body and were used to collect data while the subjects performed a variety of activities.

Before the data could be used to train the models, it was preprocessed to remove any noise or artifacts and to ensure that it was in a suitable format. This may have involved cleaning the data, handling missing values, and scaling or normalizing the features.

3.4.3 Model Training

Once the data was prepared, we trained two random forest models using the collected data: one model using data from the Nordic Thingy:52 sensor, and one model using data from the RESpeck sensor. The models were trained using the leave-one-out cross-validation (LOSOXV) method, which involves training the model on all but one of the data points and then testing it on the remaining point. This process is repeated for each data point, and the performance of the model is evaluated based on the average performance across all of the data points.

During the training process, we also specified the hyperparameters of the models, such as the number of trees in the forest and the maximum depth of each tree.

3.4.4 Model Testing

After the models were trained, we tested them using the leave-one-out cross-validation (LOSOXV) method. This involved using the models to make predictions on the data points that were held out during the training process and comparing the predicted values to the true values.

We evaluated the performance of the models using four metrics: accuracy, precision, recall, and F-1 score. The accuracy metric measures the percentage of predictions that were correct, while the precision, recall, and F-1 score metrics measure the ability of the model to correctly identify positive cases (in this case, correctly identifying the activity being performed).

3.4.5 Results

The results of the evaluation showed that both models performed well, with the Nordic Thingy:52 model achieving an average testing accuracy of 0.98 and the RESpeck model achieving an average testing accuracy of 0.97. The precision, recall, and F-1 score metrics were also high for both models.

3.4.6 ONNX and Azure Function App

The Open Neural Network Exchange (ONNX) is a format for representing machine learning models that enables models to be transferred between different frameworks and tools. Exporting a model to the ONNX format allows you to use the model with a variety of different platforms and tools that support ONNX.

Once you have exported your model to the ONNX format, you can use it in an Azure Function App, which is a serverless computing platform that allows you to run code on demand without having to worry about managing infrastructure. Azure Function Apps are designed to be scalable, reliable, and easy to use, and they can be used to build a wide variety of applications, including machine learning models.

There are several advantages to using an Azure Function App to develop a machine learning model that has been exported to the ONNX format:

Serverless computing: An Azure Function App allows you to run code on demand without having to worry about managing infrastructure, which can save you time and resources.

Scalability: An Azure Function App can automatically scale up or down based on demand, which means you can handle large amounts of traffic without having to worry about capacity planning.

Integration with other Azure services: An Azure Function App can be easily integrated with other Azure services, such as Azure Storage, Azure Stream Analytics, and Azure Machine Learning, which can make it easier to build and deploy your machine learning model in a cloud environment.

Low cost: Azure Function Apps are billed based on the number of executions and the duration of each execution, which means you only pay for what you use. This can be a cost-effective way to deploy a machine learning model, especially if you don't need to run it all the time.

Ease of use: Azure Function Apps are designed to be easy to use and require minimal setup, which means you can focus on building and deploying your machine learning model rather than worrying about managing infrastructure.

Overall, using an Azure Function App to develop a machine learning model that has been exported to the ONNX format can be a convenient, scalable, and cost-effective way to deploy your model in a cloud environment. It allows you to take advantage of the benefits of serverless computing and to easily integrate your model with other Azure services.

3.4.7 Conclusion

Overall, these results demonstrate the effectiveness of using the Nordic Thingy:52 and RESpeck sensors for human activity recognition in real-time. The high accuracy and other performance metrics suggest that the models are able to accurately identify the 14 types of human activities with a high level of confidence.

It is worth noting that the specific results may vary depending on the specific data and model implementation used. Further studies could be conducted to explore the performance of the models under different conditions and with different types of data.

3.5 Mobile application

In this section, the design concept and user interaction flow of the application are presented. The design concept outlines the overall vision and approach for the app's visual and functional design. The user interaction flow describes the steps and actions taken by the user to interact with the app, including the navigation and functionality of various pages and features.

This information provides an overview of the app's user interface, and how it's designed to provide a smooth and intuitive experience for the users. These elements are essential to creating a user-friendly app that meets the needs of the target audience.

3.5.1 Design

The design concept for the current application was developed through a collaborative process, in which each team member contributed their preferred colors to serve as the primary palette for the app. Furthermore, to complement the three primary colors, two additional colors were chosen to be used as secondary hues for typography and graphic elements.

(a) Color Palette used in the mobile application

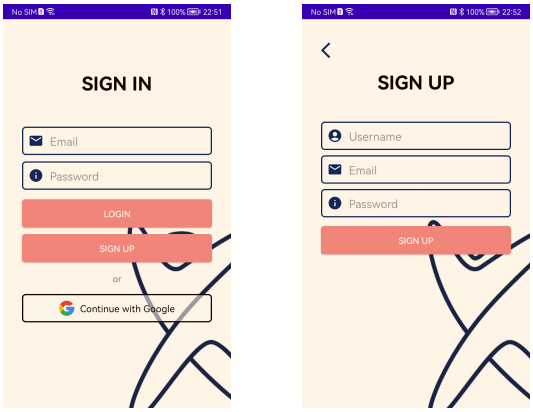
(b) Draft for the interface design

Figure 3.1: Application design materials

A draft of the interface design for the current application has then been developed. The design includes a layout that prioritizes user needs and ensures a seamless user experience. It also includes visual elements that are aligned with the brand's aesthetic and the user's expectations. The design is highly intuitive, providing users with an easy-to-use and efficient interface that is consistent throughout the application.

3.5.2 Registration

The sign-in page serves as the initial point of entry for users upon opening the application. The page presents users with the options to authenticate via Google sign-in or by providing an email and password. For users who do not have an account, a direct link to the registration page is provided, where they can create a new account by supplying a unique username, unique email address, and a password. This design ensures that the user's experience is streamlined and that the necessary information is easily accessible.



(a) Sign in page (b) Sign up page

Figure 3.2: Registration pages

3.5.3 Home Screen

Upon successful authentication, users are directed to the home page which serves as a central hub for accessing all of the application's functionalities. These functionalities include, but are not limited to, classification, historical data viewing, and sensor pairing. The design of the home page is intuitive, providing clear and easy access to all of the application's features.

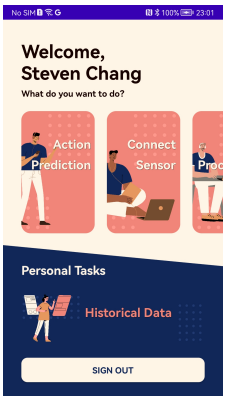


Figure 3.3: Home page of the mobile application

As the Live Data, Sensor Pairing, and Record Data pages have been provided to the students, and they have already fulfilled the desired functionalities, no modifications have been made to them. The following sections of this report will focus on the new

pages that have been designed and developed by our team. These pages have been created to enhance the user experience and provide additional functionality to the application.

3.5.4 Calibration Page

The sensor calibration page has been designed to assist users in accurately positioning the sensors on their bodies. This page also serves as an instructional resource to enhance the user's understanding of sensor placement and calibration. The user interface has been designed to be straightforward and easy to understand, providing clear instructions and visual aids to guide the user through the calibration process.

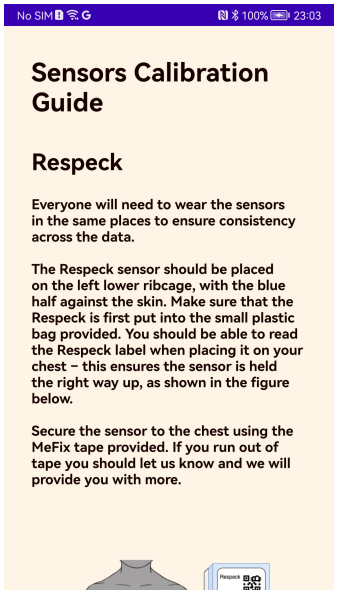


Figure 3.4: Calibration page of the mobile application

3.5.5 Classification Page

This page serves as the focal point of the application where the core functionalities have been deployed. The upper panel has been designated to display the classified results in the form of text and an icon. Additionally, two sensor status indicators have been placed in the top right corner of the page. The indicator uses a color-coding system, with green indicating that a sensor is streaming, amber indicating that it is connecting, and red indicating that it is disconnected. These design choices have been made to enhance the overall usability and functionality of the application, providing users with clear and concise information about the status of the sensors.

The lower panel allows users to select which sensors they wish to enable, through the use of toggle buttons. By default, the classification algorithm is executed on the cloud, but in the event of an internet connection interruption, the process will switch to an embedded local machine learning model. The design of this page has been carefully crafted to provide an intuitive and efficient user experience, while also providing users with full control over their sensor usage.

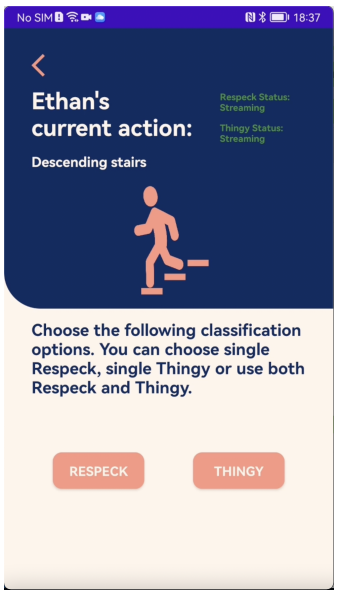


Figure 3.5: Classification page of the mobile application

3.5.6 Historical page

The historical data view page has been designed to provide users with a visual representation of their classified activities data in the form of a Pie Chart. The lower panel of the page includes a feature that allows users to specify a desired time range for the data they wish to view. This design choice allows users to easily filter and review their historical data, providing them with a clear and concise understanding of their activity patterns. The design is intuitive and user-friendly, allowing users to access their historical data in an efficient and effective manner.

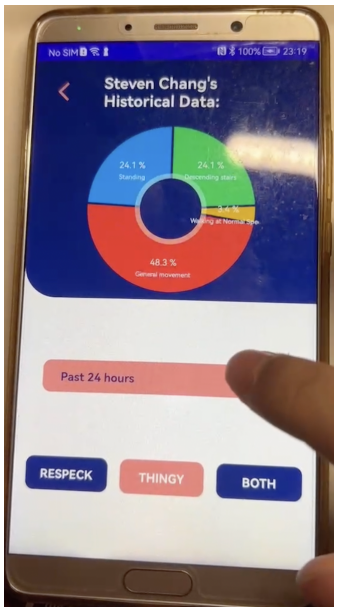


Figure 3.6: Historical page of the mobile application

3.6 Software organisation

The entire software architecture can be illustrated by the block diagram below:

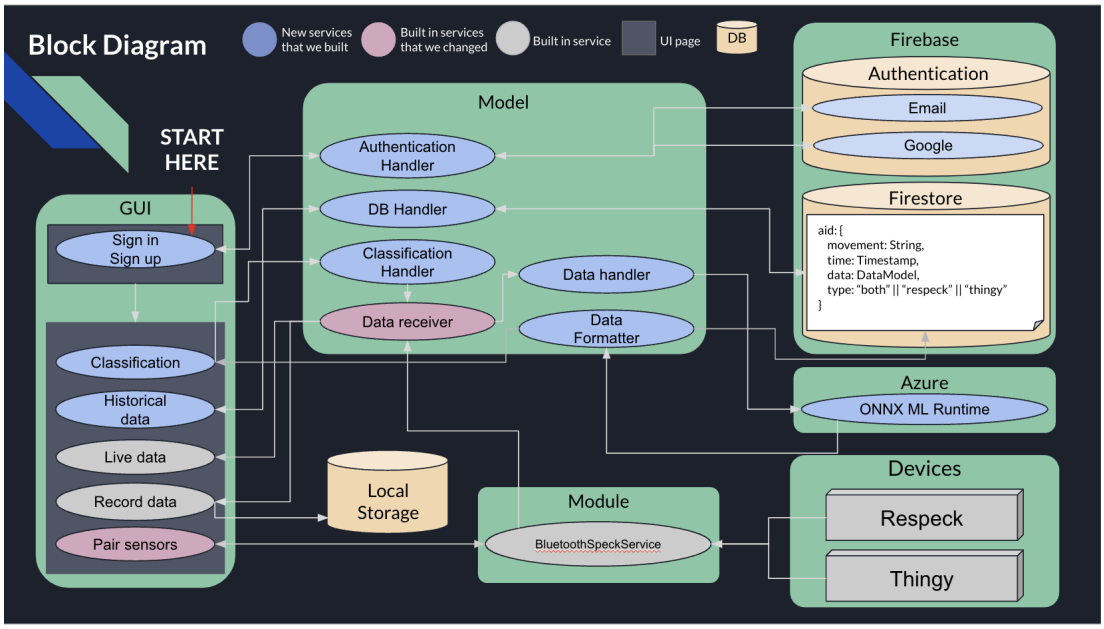


Figure 3.7: Block diagram for our system

Based on the diagram above, each part of the software organization will be introduced according to the order of the data flow.

3.6.1 Devices

This includes the IoT devices we used to detect and gather the raw data by placing them on the body. Two different types of sensor we used for the data collection and human activity recognization is Respeck and Thingy. They are connect with the software through `BluetoothSpeckService` module which will be introduced in the next section.

3.6.2 Module

The `BluetoothSpeckService` module is a critical component of our IoT infrastructure, responsible for enabling seamless communication between IoT devices and mobile devices. The module employs Bluetooth Low Energy (BLE) technology to scan for and connect to nearby IoT devices, and subsequently process the data packets transmitted by these devices. The implementation of BLE communication is achieved through the use of the `RXAndroidBLE` library, which offers a simple but reliable implementation of the BLE protocol. Furthermore, the module employs the publish-subscribe pattern for data transmission, ensuring real-time delivery and processing of data packets.

3.6.3 Local Storage

Local Storage represents the storage in the mobile devices, it is used to store the data collected from the sensors in the form of a csv file. The sensors, placed on the user's body, continuously detect and record data at a sampling frequency of 25 Hz. The collected data can be easily accessed via the file path:

`Android/app/data/com.specknet.pdiotapp/files/Filename.csv`

where the file name follows a standardized format as below:

`{sensorType}_{studentID}_{activityType}_{timestamp}.csv`

This ensures easy identification and organization of the collected data.

3.6.4 GUI

The section is the User Interface (UI) of the mobile application designed to facilitate user interactions. This section is comprised of six distinct pages, each tailored to provide a seamless user experience.

3.6.4.1 Sign in/Sign up

This is the initial page presented to the users once they have entered the app. Upon entering the application, users are provided with the options of using either a direct Google sign-in or traditional email/password authentication methods to create an account. Once an account is successfully established, the user's session remains active even upon exiting and reentering the application. The active session can only be terminated through manual logout within the application.

After the successful authentication, users are directed to the home page, which presents 5 distinct pages with each of these pages represents a specific functionality provided by the application, allowing users to effectively navigate and utilize the various features.

The implementation details of the user authentication are outlined in the *Authentication Handler* under the *Model* section of the report.

3.6.4.2 Pair Sensors

This is usually the first page that users will enter once they have landed on the home page. It allows users to establish connections with their sensors by using either the Respeck's direct scanning or Thingy's NFC pairing method. Once the connection is established, there is another instruction page provided for the users as guidance to accurately place and calibrate the sensors on the body.

In order to enhance the user experience, modifications have been made to allow pairing with only one sensor during the connection process. This is particularly beneficial in scenarios where users may only have the access to one sensor. The specific modification process is as follows:

- `BluetoothSpeckService.java`: Edited `scanForDevices` to enable sensor connection when Respeck is not paired up
- `ConnectingActivity.java`: Enable the sensor connection button when the Respeck is not paired up

3.6.4.3 Live Data

This is the page where the users can view the data collected from the sensors in the form of a line chart. In the early stage of development, this page was primarily used for monitoring sensor connections by inspecting data streaming from the sensors. However, in the final version of the application, the Classification page has been enhanced to provide data inspection capabilities, aimed at improving the user experience. This will be introduced in more detail in the Classification section.

3.6.4.4 Record Data

The Record Data page empowers users to record their activity data and save it on their mobile devices. During the data collection phase, the recording length is standardized at approximately 30 seconds. Additionally, this page includes a live data section, providing users with real-time data streaming from the sensors, ensuring the sensors are running as expected.

3.6.4.5 Classification

This is the crucial page of the application, designed for real-time human activity recognition capabilities with the ability to identify up to 14 distinct movements on both device and cloud with the use of single or both sensors. The on-device machine learning model is implemented as an embedded TFLite file via TensorFlow Lite, while the cloud-based model is hosted on Azure Functions App and formatted in Open Neural Network Exchange (ONNX) format.

This page includes indicators that inform users of the sensors' connectivity, eliminating the need for navigating between pages to check sensor status. To provide flexibility, the pages offers three options to the users: classification with Respeck, classification with Thingy, or classification with both sensors. Upon receiving the classified result, it is displayed on the user interface with a text message describing the movement type and an accompanying icon, making the results intuitive and easy to understand for the users.

The technical aspects of the classification and data receiver are detailed in the *Classification Handler* and *Data Receiver* section of the *Model* section in the report.

3.6.4.6 Historical Data

Once receiving the classified data, it is stored in the database and labelled with relevant information such as timestamp and sensor type. To view historical data, users can navigate to this page that visualizes the data in the form of a pie chart. Furthermore, users have the capability to specify a specific time range for past data, and the system will return a filtered result based on the user's customization.

A comprehensive explanation of the database implementation and data querying processes can be found in the *Database Handler* section of the *Model* section of the report.

3.6.5 Model

The Model section here does not mean the machine learning model, it means the model in the Model–view–controller (MVC) design pattern, which is a software architectural pattern used in software engineering to separate the concerns of the user interface, data management, and control flow in a software application. The Model represents the data and the business logic of the application. It is responsible for storing and manipulating the data, and for enforcing the rules of the application. In our application, we have seven components responsible for our data management.

3.6.5.1 Authentication Handler

The Authentication Handler is tasked with managing user authentication, supporting both Google sign-in and traditional email/password methods. Its primary function is to perform input validation during the authentication process, manage active user sessions, and facilitate the transfer of user information between pages."

The Sign-up and Sign-in pages employ the Authentication Handler to validate user input, such as username and password, for null values and to check for duplicate usernames against registered users. In the case that these conditions are not met, the actual authentication code will not be executed.

After successful authentication, the user's session will remain active until manually terminated by clicking the logout button on the home page. Therefore upon each subsequent application launch, two objects are instantiated during the creation phase: a `GoogleSignIn` client and a `Firebase` instance. The selection of the correct object is determined based on the user's chosen sign-in method, which includes either Google sign-in or traditional email/password sign-in. The selected object is then used to extract the current user session and retrieve the account's username, which is then passed to other pages by putting the account information in an `Intent`.

The entire authentication process as well as the user information is hosted and stored in Google Firebase, which will be further discussed in the *Firebase* section.

3.6.5.2 Classification Handler

The Classification Handler serves as a control panel for users to adjust the settings of the classification process, including the option to run the classification model locally or on the cloud, and the selection of sensors to be used for data transmission to the classification model. By default, the model is set to run on the cloud, and users have the option to toggle the buttons on the classification page to select their preferred sensors.

The settings configured through the Classification Handler are also used to create labels for the classified results such as the type of sensors. The newly formatted data is then

sent to Azure Functions App and stored in the Firestore, which will be explained in more detail in the *Azure* and *Firebase* section.

3.6.5.3 Data Receiver

The Data Receiver receives raw data streamed from the sensors through the use of `BroadcastReceiver`. Within the Receiver, the raw data, in the form of `ThingyLiveData` or `RESpeckLiveData` (depending on which sensors are in use), is converted to a `List` of `Float` numbers and added to a `MutableList`, resulting the collection of converted data being stored in the form of `MutableList<List<Float>>`. The use of `MutableList` instead of a `List` is to ensure that the size of `MutableList` does not exceed 50, representing 2 seconds of data, as required by the machine learning model. Once the size of `MutableList` has reached to 50, the collected data is then transferred to the *Data Handler* for the final process before being fed to the classification model.

3.6.5.4 Data Handler

By receiving the selected sensor source from the Classification Handler and the data sent from Data Receiver in the form of `MutableList<List<Float>>`, the Data Handler then performs a series of actions to ensure that the sensor data meets certain conditions, including generating labels based on the user's sensor selections, and verifying if the selected sensors function properly. Following these checks, the data, along with sensor type labels, is sent to Azure Function Apps for classification of the user's activity type via an HTTP/POST request. After the completion of the request, the Data Handler removes the first 25 items from the `MutableList`, which corresponds to approximately 1 second of data. This is to increase the frequency of requests to Azure Function Apps, thereby providing more up-to-date results and enhancing the user experience by avoiding stale results on the user interface.

3.6.5.5 Data Formatter

The Data Formatter is the first component to receive the classified result in the form of an integer from Azure Functions App. After the reception of the data, it is used as an index to retrieve the corresponding activity type from a preset `Enum` type. This process allows for efficient and accurate data formatting, ensuring that the correct activity type is displayed to the user. A complete `switch-case` statement between the index and `Enum` is shown as follows:

```
return when (idx) {
    0 -> ActionEnum.DESK_WORK
    1 -> ActionEnum.WALKING_AT_NORMAL_SPEED
    2 -> ActionEnum.ASCENDING_STAIRS
    3 -> ActionEnum.DESCENDING_STAIRS
    4 -> ActionEnum.SITTING_STRAIGHT
    5 -> ActionEnum.SITTING_BENT_FORWARD
    6 -> ActionEnum.SITTING_BENT_BACKWARD
    7 -> ActionEnum.STANDING
    8 -> ActionEnum.RUNNING
```

```
            9 -> ActionEnum.LYING_DOWN_ON_THE_LEFT_SIDE
            10 -> ActionEnum.LYING_DOWN_ON_THE_RIGHT_SIDE
            11 -> ActionEnum.LYING_DOWN_ON_THE_BACK
            12 -> ActionEnum.LYING_DOWN_ON_STOMACH
            13 -> ActionEnum.GENERAL_MOVEMENT
            else -> ActionEnum.GENERAL_MOVEMENT
        }
```

Once the activity type is selected from the mapping, it is used to retrieve the corresponding icons under the use of a `switch-case` statement similar to the one shown above. The resulting icons are then displayed on the Classification page with the activity type, providing a visual representation of the classified activity for the user.

The final step in the process is the storage of historical records. The sensor type (either ”respect”, ”thingy” or ”both”), activity type as a string, data from the Data Handler, and the timestamp are all combined and stored under the type of a `Hashmap`. This hashmap is then sent to Firestore via an HTTP/POST request, allowing for the preservation of historical records for future reference. This process ensures the integrity and accessibility of the data for future analysis and review.

3.6.5.6 Database Handler

The Database Handler serves as the primary means for retrieving historical data from Firestore and presenting it on the Historical Data page for the users. The `startDate` and `endDate` specified by the users on the Historical Data page are used to select the appropriate data from Firebase via an HTTP/GET request. The retrieved hashmap is then converted into a custom data type called `HistoricalDataModel`, which offers a more structured and organized approach to handle the data and define specific properties tailored to the requirements for data visualization to make the code more readable and clear. the `HistoricalDataModel` is then used to generate a Pie Chart by using `MPAndroidChart`. This chart is then presented on the user interface, allowing for easy visualization and analysis of the historical data.

3.6.6 Azure

Azure Function App is a serverless compute service provided by Microsoft Azure that allows developers to run their code in response to specific triggers without managing the underlying infrastructure. It is a suitable choice for an IoT motion recognition system. It allows for running machine learning models in a serverless environment, eliminating the need for provisioning or managing infrastructure. It also automatically scales as needed and provides a simple way to run code in response to events, making it easy to handle the logic for running inferences on the machine learning models and returning predictions to the Data Formatter component.

3.6.6.1 Implement

Our Azure Function App serves as the cloud backend for our IoT motion recognition system. It receives sensor data in the form of MutableList¡List¡Float¿¿ along with

sensor type labels via an HTTP/POST request from the Data Handler component. The data is then passed through the function that runs on the Azure Function App.

The function uses the ONNX Runtime library to perform inferences on the two machine learning models that are packaged in the ONNX format, one for the ”respeck” sensor data and one for the ”thingy” sensor data. The function then checks the type of sensor data that is received in the request, and runs the appropriate prediction function depending on the sensor type. If the sensor type is 'respeck' it runs the rep_predict () function, if it is 'thingy' it runs the thingy_predict () function, if it is 'both' it runs both functions, and compare the model outputs to decide which model has the higher prediction confidence.

The function then returns the prediction result in the form of an HTTP response, which is received by the Data Formatter component. The Data Formatter component then maps the prediction result to a specific activity type and uses this activity type to retrieve the corresponding icons and display them on the Classification page, providing a visual representation of the classified activity for the user.

3.6.7 Firebase

Firebase is a mobile and web application development platform provided by Google that offers a variety of tools and services such as real-time database, authentication, hosting, storage, and more to help developers build and scale their applications. In the current application, the authentication and storage components of Firebase have been utilized. These components will be further discussed in the sections below.

3.6.7.1 Authentication

The mobile application supports two forms of user authentication: Google sign-in and email/password. The Firebase authentication modules handle a significant portion of the authentication process, including the management of user sessions and the secure storage of user account information. The implementation of Firebase's authentication service ensures a high level of security, including the protection of users' privacy from even the developers of the application.

In order to utilize Firebase Authentication within the application, two instances are required: GoogleSignIn for users who choose to use Google sign-in, and Firebase for users who choose to use email/password sign-in. Once the user has been successfully authenticated, session information can also be retrieved from these instances. This ensures that the authentication process is easy to implement and maintain, which saves development time and resources.

3.6.7.2 Firestore

Firestore is a cloud-based NoSQL document database provided by Google's Firebase that allows developers to store, sync, and query data for their mobile applications. The decision to utilize a NoSQL database was based on the recognition of its advantages over traditional SQL databases, specifically its ability to handle large amounts of

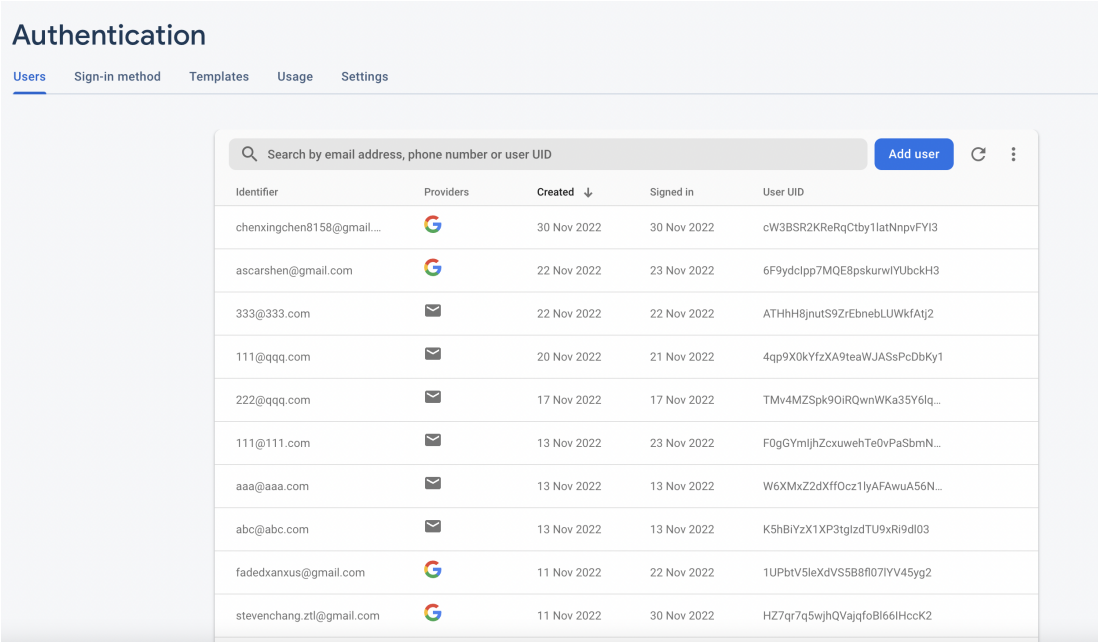


Figure 3.8: A screenshot of the developer's dashboard from Firebase Authentication

unstructured data, its horizontal scalability, and its efficiency. These characteristics make it well-suited for applications involving big data, real-time analytics, and user-generated content. Additionally, Firestore's offline synchronization feature makes it a suitable choice for mobile and web applications.

Since it is required during the registration phase that each username should be unique, each user is allocated one individual collection in the Firebase database, with the username serving as the collection id. Within each user's collection, each historical record is stored as an individual document, with a unique document id generated by Firestore. The format for these historical records is as follows:

```
aid: {
    movement: String,
    time: Timestamp,
    data: DataModel,
    type: "both" || "respect" || "thingy"
}
```

In order to ensure successful storage and retrieval of data, it is necessary to modify the rules within Firestore to allow for modification of the database by anyone with API access. This is accomplished by changing the rules to:

```
service cloud.firestore {
    match /databases/{database}/documents {
        match /{document=**} {
        allow read, write: if true;
        }
    }
```

```
}
```

In the current mobile application, the `FirebaseFirestore` object has been utilized to bridge the connection between the mobile application and the Firestore on the cloud. This approach allows for efficient and streamlined data management within the application.

3.7 Testing

3.7.1 Unit Testing

Unit testing is a method of testing individual units of code, such as methods or classes, to ensure they are working correctly. In the development of this application, unit testing was implemented as a crucial method for ensuring the reliability and functionality of the various helper functions. These models, such as Data Formatter and Database Handler, were regularly tested to verify their ability to properly clean and format data prior to transmission with the cloud-based service or presentation to the user interface. The utilization of unit testing throughout the development process greatly enhanced the overall quality and performance of the application.

3.7.2 Instrumentation Testing

Instrumentation testing is a methodology utilized for testing the application's UI and its interactions with the system. Due to constraints on the project timeline, it was decided to manually perform instrumentation testing in order to evaluate the functionalities of the application. This approach involved deploying the app on an emulator or physical device and manually interacting with each page and component to assess their performance and functionality. While this method of testing required a significant investment of time, it had the advantage of reducing the development time required for building and debugging instrumentation tests.

3.7.3 Performance Testing

After the application passes unit testing and instrumentation testing, it will then be subjected to performance testing. This phase of testing involved evaluating the app's behavior and capabilities under various conditions. The specific conditions that were tested include:

- Testing the app, classification, and user authentication with Wifi connection
- Testing the app, classification, and user authentication with poor Wifi connection
- Testing the app, classification, and user authentication with data roaming
- Testing the app, classification, and user active session without network connection
- Testing the app and classification with sensors placed on the body properly

- Testing the app and classification with sensors placed far away from the mobile device
- Switch the sensors when the app is receiving data
- Shut down the sensors when the app is receiving data
- Switch the sensors when the app is classifying user activities
- Shut down the sensors when the app is classifying user activities
- Testing the app, Azure Functions App, Firestore with classification frequency as 1 request in every 2 seconds
- Testing the app, Azure Functions App, Firestore with classification frequency as 1 request per second
- Testing the app, Azure Functions App, Firestore with classification frequency as 5 request per second

It's important to note that the performance testing is a crucial step in the development process as it helps to identify and address any potential bottlenecks or issues that may affect the app's overall performance and usability. The results of this testing phase were used to make any necessary adjustments to the app in order to optimize its performance.

3.7.4 Local Testing

In order to ensure the consistent performance and appearance of the application across various mobile devices, local testing was conducted. This testing phase involved evaluating the app on multiple mobile devices with different brands and models. The primary objective of this testing was to verify the consistency of the user interface across different device types and configurations.

This testing helps to identify and resolve any potential issues related to the compatibility and adaptability of the app across different devices, which is crucial for providing a seamless user experience. The results of local testing were used to make any necessary adjustments to the app to ensure consistent functionality and appearance across different devices.

3.7.5 User Acceptance Testing

In order to gather feedback and evaluate the overall user experience of the application, user acceptance testing was conducted. The primary objective of this testing phase was to determine if the app meets the expectations of its intended users and is user-friendly. The end-users were asked to perform specific tasks and provide feedback on the app's ease of use, functionality, and overall satisfaction.

This type of testing is essential in order to identify any potential issues related to the usability and user experience of the app, and to make any necessary adjustments to improve the overall quality and performance of the application. The results of the user

acceptance testing were analyzed and used to make any necessary changes to the app to ensure it met the needs and expectations of this project.

Chapter 4

Results

4.1 Critical analysis of the implementation using quantitative methods for accuracy of the classification both offline and in real-time.

4.2 Benchmarking of your implementation in terms of processing cycles, memory usage, power consumption, latency in generating results.

4.2.1 Test Environment

The test environment consisted of a Samsung Galaxy S10 smartphone running Android 10. The app was tested under 6 different scenarios. The sensors used were the Respeck and Thingy.

4.2.2 Test Scenarios

The app was tested under six test scenarios: no interaction, entering the app with the active session, signing in, signing out, fetching data from the Firestore, and finally sending data to Azure for classification and then storing it in the Firestore. In each test case, the CPU usage, memory usage, power consumption, and latency were measured.

4.2.3 Results and Analysis

The results of the benchmarking are shown in Figure 4.1. In conclusion, the benchmarking results show that the mobile app is able to perform real-time activity classification with an average latency of less than 800 ms for the data streamed. The app also demonstrated efficient use of CPU, power consumption, and relatively small memory usage. However, it's worth mentioning that this benchmarking is based on a specific device and Android version, it's good to test it on different scenarios as well.

	CPU	Memory Usage	Power comsumption	Latency
No interaction	App: 1% Others: 15% Threads: 45	Others: 10 MB Code: 31.6 MB Stack: 0.1 MB Graphic: 124MB Native: 166.3 MB Java: 12.8 MB	CPU: Light Network: None Location: None	N/A
Auto sign in (user session)	App: 13% Others: 36% Threads: 20	Others: 51.59 MB Code: 26 MB Stack: 0.1 MB Graphic: 143.6MB Native: 203.7 MB Java: 11.1 MB	CPU: Light Network: Light Location: None	N/A
Sign in	App: 18% Others: 32% Threads: 47	Others: 48.6 MB Code: 31.1 MB Stack: 0.05 MB Graphic: 20.9MB Native: 200 MB Java: 10.2 MB	CPU: Light Network: Medium Location: None	185 ms
Sign out	App: 19% Others: 29% Threads: 44	Others: 13 MB Code: 31.6 MB Stack: 0.1 MB Graphic: 142.7MB Native: 168.7 MB Java: 13.1 MB	CPU: Medium Network: Light Location: None	170 ms
Fetch Data	App: 15% Others: 28% Threads: 43	Others: 10.7 MB Code: 31.6 MB Stack: 0.01 MB Graphic: 168.7MB Native: 169.3 MB Java: 13 MB	CPU: Medium Network: Medium Location: None	461 ms
Classify Data/Store Data	App: 28% Others: 41% Threads: 44	Others: 10.3 MB Code: 33.2 MB Stack: 0.1 MB Graphic: 154.3MB Native: 166.3 MB Java: 13.2 MB	CPU: High Network: High Location: None	786 ms

Figure 4.1: Benchmarking Results

Chapter 5

Conclusions

5.1 Reflection on the project

5.2 How might you wish to extend the project and improve the implementation

If given additional time, the project could be extended to include the implementation of various system design techniques to enhance the scalability and robustness of the system. These techniques would be aimed at increasing the overall performance and capability of the system, making it suitable for industrial-level use. The specific system design techniques that would be adopted would be discussed in the following sections

5.2.1 Load balancer

A load balancer distributes network traffic among multiple servers, improving app performance and scalability by ensuring that no single server is overwhelmed with too many requests. Integration of reverse proxy like HAProxy or Nginx in Android Studio can be used to implement load balancing, improving app performance and availability.

5.2.2 Optimize data pipeline

Optimizing the data pipeline involves optimizing the flow of data from the IoT devices to the machine learning model, and then to the Firestore database. This can improve the performance and scalability of our android app by reducing data bottlenecks and latency. One effective approach is to use indexing and partitioning in Firestore to improve query performance and minimize data duplication. Additionally, using a database replicator or proxy can improve data availability and reduce the load on the main database. These techniques can be implemented on Firestore using Firebase's built-in tools and APIs, and can be integrated into the app through Firebase's SDK for Android Studio. By optimizing the data pipeline, we can ensure that our app can handle large volumes of data and high request loads.

5.2.3 Message queue

A message queue is a system that allows applications to send and receive messages in an asynchronous manner. This can greatly improve the performance and scalability of our Android app by decoupling different components and allowing them to communicate in a non-blocking way. Message queues can also be used to handle communication between our mobile app and other services such as the machine learning model hosted on Azure. Implementations such as RabbitMQ or Apache Kafka can be integrated into Android Studio to improve the scalability of the app.

5.2.4 Caching

Caching is a technique that temporarily stores frequently accessed data in a location that can be accessed faster than the original data source. This can greatly improve the performance of our Android app by reducing the number of network requests and database queries needed to access data. On Android Studio, caching can be implemented by using a library such as Retrofit or OkHttp, which allow for caching of network responses. Additionally, Firestore has built-in support for caching data locally, which can be enabled through the Firebase SDK. By implementing caching, we can improve the app's responsiveness, while also reducing the load on the network and database.

5.2.5 Optimize Machine Learning algorithm

Bibliography

[1] Lu Xu, Weidu Yang, Yueze Cao, and Quanlong Li. Human activity recognition based on random forests. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 548–553, 2017.

[2] Schalk Wilhelm Pienaar and Reza Malekian. Human activity recognition using lstm-rnn deep neural network architecture. In *2019 IEEE 2nd Wireless Africa Conference (WAC)*, pages 1–5, 2019.

[3] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert systems with applications*, 59:235–244, 2016.

[4] Tahmina Zebin, Patricia J Scully, and Krikor B. Ozanyan. Human activity recognition with inertial sensors using a deep learning approach. In *2016 IEEE SENSORS*, pages 1–3, 2016.

[5] Md. Zia Uddin, Weria Khaksar, and Jim Torresen. Activity recognition using deep recurrent neural network on translation and scale-invariant features. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 475–479, 2018.

[6] Lasitha Piyathilaka and Sarath Kodagoda. Gaussian mixture based hmm for human daily activity recognition using 3d skeleton features. In *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, pages 567–572, 2013.

[7] Palwasha Afsar, Paulo Cortez, and Henrique Santos. Automatic human action recognition from video using hidden markov model. In *2015 IEEE 18th International Conference on Computational Science and Engineering*, pages 105–109, 2015.

[8] Oscar D. Lara and Miguel A. Labrador. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys Tutorials*, 15(3):1192–1209, 2013.

[9] Ferhat Attal, Samer Mohammed, Mariam Dedabrishvili, Faicel Chamroukhi, Latifa Oukhellou, and Yacine Amirat. Physical human activity recognition using wearable sensors. *Sensors*, 15(12):31314–31338, 2015.

[10] Ming Zeng, Le T. Nguyen, Bo Yu, Ole J. Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *6th International Conference on Mobile Computing, Applications and Services*, pages 197–205, 2014.