Dissertation:

Journaling to Insight: Empowering People with Dementia using AI and Social Robotics.

ID:

University of Nottingham

Computer Science G400 – COMP3003

Word Count: 14,569

Figures & Extras Word Count: 1,779

Pages: 38

I hereby declare that this dissertation is all my own work, except as indicated in the text.

Signed: T.S

1 Abstract

Dementia is a prevalent neurological condition marked by a progressive decline in cognitive function (NHS, 2023), impacting both diagnosed individuals and their loved ones. Effective treatments remain rare, expensive, and difficult to implement. Furthermore, the diverse symptomatology across cases poses additional challenges, many of which modern solutions fail to address.

This dissertation presents a mobile application designed to enhance the daily lives of people with dementia (PwD). The application, available for free for Android devices, integrates state-of-the-art artificial intelligence (AI) to simplify everyday tasks and increase accessibility. By applying advances in natural language processing (NLP) and social robotics, this project explores how AI-driven interventions can be both cost-effective and widely accessible.

A key component of the project is an integration with a MiRo social robot, enhancing accessibility and engagement. NLP tools are used to extract meaningful insights from user entries, enabling features such as mood monitoring, medication tracking, and event organisation. These insights are shared with registered caregivers, allowing for non-intrusive, remote support alongside automatic generation of reminders to aid memory and help with daily planning.

We investigated the use of AI and social robotics within gerontology, focusing on their impact on older PwD. This research contributes to the field by identifying areas of greatest need through patient and public involvement (PPI) and user testing, thus informing the development of more targeted and personalised support tools.

The project concludes with a successful implementation that demonstrates potential to improve the quality of life for PwD and their caregivers. Positive user feedback, and valuable stakeholder input, praising the accessible app design, beneficial use of social robotics, and usefulness of the reminder system, underlines a desire for help with daily organisation and easy to use supportive technology. This supports the system's potential and lays the foundation for future innovation in digital dementia care, highlighting the importance of assistive technologies in the lives of PwD.

Key Words: *Natural Language Processing, Artificial Intelligence, Social Robotics, Dementia, Naïve Bayes, TF-IDF, Logistic Regression.*

2 Acknowledgements

The work for this project has been supported by and influenced by many people, to whom I would like to express my sincere appreciation and gratitude.

To my supervisor Dr Armaghan Moemeni, for her guidance, support and valuable feedback throughout this project.

To Dr Neil Chadborn and Dr Laura Edwards for their expertise, kindness, and for playing a crucial part in organising meetings with Public and Patient Involvement.

To Dominic Price for his technical knowhow and introduction to the Cobot Maker Space.

To my friends and family for their unwavering support, confidence, and kind words.

Dedicated to Great-Grandma, Sylvia Lofthouse.

(1928 - 2025)

---

3 Contents

4 Introduction

Dementia is one of the most common and recognisable neurological conditions affecting millions of people worldwide. The challenges associated with this condition extend beyond the individuals diagnosed, significantly impacting caregivers and family members. This dissertation describes a system to improve the quality of life for people with dementia (PwD), while enabling caregivers to provide remote, non-intrusive support. This is achieved by automatically picking out meaningful 'insights' written to a journal using artificial intelligence (AI) detection features to categorise mood, events and medications. Understanding the limitation of technological proficiency among older users was a key factor, and in response we integrated a social robot coupled with a voice user interface (VUI), designed to support PwD using spoken inputs. By designing for inclusivity and accessibility, this system has a lower bound to entry, offering support to a wider audience.

5 Motivation

"People fear dementia more than any other disease." (CPE, 2022).

Dementia is a relentless syndrome "associated with an ongoing decline in brain functioning" (NHS, 2023). Despite advances in technology, dementia remains incurable. Effective and affordable treatments are rare, and fewer aim to cater for the diagnosed individuals alongside their caregivers. Through personal experience with dementia, I have firsthand knowledge of how this violent condition can reshape relationships and introduce new difficulties and challenges.

Current technological solutions are outdated, insufficient, and poorly designed. The lack of accessible technology prevents older PwD from developing the skills necessary to engage with more advanced technologies that could improve their quality of life in the future. By not designing for PwD, we risk reinforcing the perception that technology is inaccessible or even useless (Petr Balog et al., 2023). This suggests a new approach to digital care is needed, one that is easy to use and accessible to a wide range of people of differing technological ability.

Solutions based on chatbots have been proven effective in some cases, however, research suggests that "chatbots [are] not […] practical for every patient; they can't replicate human emotion" (Müller et al., 2022). It can then be said these applications may be unintentionally sacrificing the emotional connection that is crucial to dementia care. For this reason, a more personalised solution, using both AI and social robotics, may be more appropriate. The benefit of this approach is its seamless integration into daily life, working alongside a digital diary to give daily structure and combat memory loss. Additionally, using social robotics has the potential to provide companionship, promoting out loud speech as a form of interacting with the application while the robot responds with physical, human like emotion. Social robots have been shown to provide companionship (Faisal *et al.*, 2024), used as a tool to tackle the loneliness often associated with dementia, while speaking out loud is beneficial for slowing the symptoms of dementia (Quinlan and Taylor, 2013).

6 Aims and Objectives

6.1.1 Key Aim

The overarching aim of this project is to develop a mobile application with AI-powered diary analysis and organisational features using advanced natural language processing techniques, and to integrate a social robot and voice user interface, to facilitate user interaction and provide companionship and timely care.

6.1.2 Key Objectives

1. *Identify appropriate classification techniques*: Conduct research into classification techniques and identify appropriate techniques to design a system capable of analysing large bodies of text in the form of diary entries.
2. *Android application:* Design and implement an Android application that allows users to create and view a digital diary, as well as view automatically scheduled events and analysed notes. This application will feature methods for carers to access this information through an admin tab.
3. *Python classification algorithm*: Design and implement a Python back-end (PyB) for text classification and intent matching, using the techniques identified in research.
4. *RESTful API:* Design and implement a RESTful API to enable communication between the mobile application and the PyB algorithm using HTTP calls.
5. *Identify social robot communication techniques and principles*: Conduct research into relevant social robot techniques, specifically MiRo robots, to create an effective companionship support appropriate for use with PwD.
6. *Voice user interface*: Design and implement a VUI using a social robot and Google Speech-to-Text as a communication medium, which can access all the features available in the mobile application.

7 Related Work

Technology in dementia is an emerging field. This section discusses works that are relevant to this project and that have been used to influence its direction.

7.1.1 Care: A Chatbot for dementia care

Care is a dementia friendly chatbot that aims to use accessibility features to improve care. From this work, there are three main takeaways useful for this project. Firstly, easy handling is a must; one participant stated, “big picture, big texts, easy handling”, suggesting that PwD struggle with complex systems. In response, our application is designed with simplicity in mind. Secondly, the application must be clear and readable; users noted that low contrast text was hard to read. Consequently, this dissertation will implement high contrasting text and widgets, as well as larger areas for touch interactions. Finally, highly abstracted usage; interviewees had various troubles using the application, as they found that 3-4 responses were too many, instead suggesting a “this or this” approach. This suggests a linear approach design would be most effective in our work. The researchers behind Care also mention that they “did not use […] natural language processing” in their chatbot, which we will use to differentiate this work by implementing NLP elements.

(Müller, Paluch and Hasanat, 2022)

7.1.2 Robot-based solution for helping Alzheimer patients

This study focuses on the use of a Pepper humanoid social robot, and its usage to try to “refresh patients’ memory”. The Pepper robot is designed to interface through its touch screen, and, like a MiRo robot, can record speech using a microphone. In an article raised by this study, it was found that people “enjoyed the robotic interaction” and “were highly engaged” (Carros *et al.*, 2020). It is then appropriate to extrapolate that a MiRo robot would illicit a similar response, sharing many design features with Pepper. This engagement is important in getting PwD to use and enjoy MiRo and the diary system, providing an extra element of companionship while also taking advantage of the applications other features for improved daily structure. (Faisal et al., 2024)

7.1.3 Adaptive User Interface for Healthcare Application for People with Dementia

The Indoor and outdoor NITICSplus solution for dementia challenges (IONIS) interface is a “user-centred modular […] platform” supporting a “wide range” of solutions designed for helping the elderly and people with dementia. The main aspects of this work to focus on relate to the design and accessibility features. Firstly, the authors give 6 proposed accessibility features for IONIS, the most relevant being intuitive symbols and navigation, and voice communication. IONIS attempts to create a more dementia friendly interface by using clear symbols and obvious navigation structures. These ideas fold nicely into the design of our mobile application, where we implement simple navigation structures as well as easy to understand symbols to aid use. Our application will also include voice communication using MiRo to improve accessibility and add an element of companionship. By building on the components proposed for IONIS, this application will be able to create a more dementia friendly environment.

(Awada et al., 2018)

Figure 7.1: Labelled IONIS Interface

7.1.4 Lotsa Helping Hands

Lotsa Helping Hands is an application designed to improve communication between families through a variety of social tools. “Care Calendar” is a group calendar allowing users in a “community” to manage events, which can then be seen by everyone in that community. The idea behind this is to take away some of the planning responsibilities of PwD and instead allow relatives to plan events for them and provide a central place to view those events. Our mobile application will implement a reminder system, taking inspiration from Lotsa Helping Hands and expanding this to automatically send reminders to registered admin devices, allowing a similar shared calendar experience.

(Lotsa Helping Hands LLC, 2025)

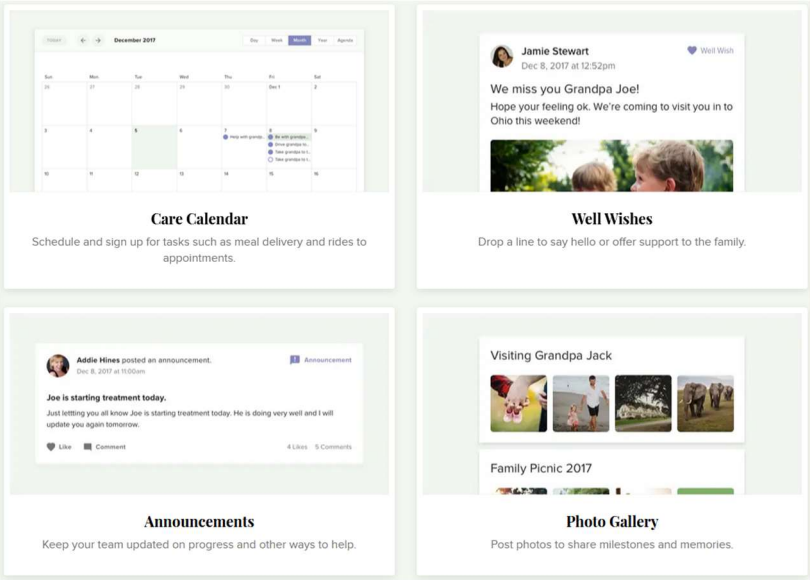

Figure 7.1: Lotsa Helping Hands Features (Lotsa Helping Hands LLC, 2025)

7.1.5 Medisafe

Medisafe is a “Digital Drug Companion” deigned to offer personalised support with pill reminders and medication tracking. Medisafe is unique in the way it uses a proprietary “predictive machine learning engine” to “drive personalised patient support”, with the goal of improving adherence to taking treatments on time. Where Medisafe falls short is its reliance on manual entry of medications. This is a disadvantage for use with PwD, as they may rely on carers to input this information for them. Our approach is to passively track medications by parsing diary entries that may mention them, automatically adding them to a tracking system without any additional intervention from users of the application. This has the potential to be better for reducing the stress involved in tracking medications manually.

(Medisafe® Medication Adherence Platform, 2025)

8 Legal, Social, Ethical and Professional Considerations

This section provides reflections on various relevant Legal, Social, Ethical and Professional issues that are relevant to this dissertation.

8.1.1.1 Copyright, Designs and Patents Act (1988)

This project uses several open-source libraries across multiple programming languages. These libraries are commonly used and distributed, however there are no intentions for commercial gain or distribution. It is to be kept as strictly research.

This is consistent with the use of a MiRo social robot, a copyright of Consequential Robotics Ltd. While MiRo is used as a communications medium, we do not distribute or sell any of the proprietary technologies associated with MiRo or Consequential Robotics Ltd.

8.1.1.2 Research Ethics

As this project serves to be an aid for a vulnerable group, we had to be especially diligent when defining our ethical requirements. We use human participants in this project; kept to strictly non-vulnerable groups to test basic functionality of the system, in line with ethics policies set out by the University of Nottingham. Data retrieved from this testing followed the standards set out in our ethical application, which can be summarised as being anonymised, kept private, and securely stored. It was ultimately decided to not use any vulnerable people in our testing, as we could not guarantee that they would be comfortable in a new environment, using experimental technology and social robotics, and we did not want to risk any undue harm in such a fragile situation.

8.1.1.3 Data Protection Act (2018)

Data is handled in line with the policies set out in the Data Management Plan (DMP). Participants are anonymised and data is kept secure and on university servers where possible. To protect the privacy of users, additional steps were taken to ensure compliance with the DMP. Namely, any entries and insights created and generated for test users were deleted completely from the system after the tests were concluded. Any audio recordings from the MiRo robot are automatically deleted immediately after processing by the backend system, so they are not kept for any meaningful amount of time. Any user accounts created during testing are also removed, and temporary email addresses were created for test participants so that they did not have to use personal accounts.

As required by Article 6 of the GDPR, consent was explicitly received in the form of consent forms, signed and dated by participants before any user testing commenced.

8.1.1.4 United Nations Sustainable Development

One important development goal set by the UN is goal 3: Good Health and Wellbeing. This dissertation attempts to contribute to this goal by promoting systems to aid with improved quality of life for PwD.

8.1.1.5 Broader Ethical Considerations

Bias is a growing issue in the world of AI, where issues such as racism, sexism and ageism are unintentionally baked into AI models due to the origin and demographic of their training data. This is important for this dissertation as it implements an existing VUI model, in the form of Google Cloud Speech-to-Text. To minimise the effects of bias, we chose GCST specifically due to its vast range of training data, spanning many countries, languages and demographics. Using a varied model like this one will reduce the inherent bias in many AI models, promoting inclusivity in this applications design.

9 Description of the Work

This dissertation presents a software-oriented system aiming to improve the quality of life for PwD. To do this, it implements an Android application allowing users to write to a journal, manipulate each entry, and provide access to AI generated insights. A Python system is developed using NLP techniques to extract relevant information from user entries and return a readable insight. As an additional accessibility element, a MiRo social robot, implemented in Python and ROS, gives an extra dimension of communication and companionship.

9.1.1.1 Functional Requirements

1. The system saves journal entries when a user enters them into the mobile application.
2. A user can view, edit, and delete journal entries from the mobile application.
3. The application should allow users to securely login and sign up using personalised credentials.
4. Users should receive notifications of scheduled events from the mobile application.
5. Users should receive personalised AI Insights produced from their journal entries.
6. Users should be able to view their own insights from a page in the application.
7. Administrative users should be able to view assigned users' journal insights from the application.
8. The system should be capable of using a social robot for inputs.
9. The social robot should be able to listen to user speech and translate it to text.
10. The social robot should be able to respond to user speech with physical gestures.

9.1.1.2 Non-Functional Requirements

1. The system should be able to handle multiple users on multiple devices.
2. The mobile application should employ secure login features.
3. Recorded speech from the social robot should be deleted immediately after translated into text.
4. Stored data should comply with data protection regulations.
5. The application should be stable and correctly use the Android lifecycle to prevent unexpected behaviour.

10 Methodology

This section discusses the methodology and systems used in the production of the system and introduces some high-level structures used to influence the final design.

10.1.1 Mobile Application

10.1.1.1 Application Platform

Android has been chosen as the development platform for the mobile application. This was a clear choice due to its popularity, encompassing a majority stake of the smartphone market, combined with its ease of use. This opens the possibility of reaching more PwD, especially in disadvantaged areas where Android phones show a clear lead in affordability over alternatives like iOS devices. Android also offers a fully documented developer program and simulation environment in Android Studio, allowing easier testing of applications.

10.1.1.2 Development Language

Java was the natural choice for the development language for the mobile application. Java is fully supported for Android development and will be used to build the key components of the application in an object-oriented manner.

10.1.1.3 Development Tools

Android Studio is used as it supports Java development with useful tools such as XML generation with an interactive GUI. This expedites development of Android applications by abstracting down to basic visual coding. Android Studio also offers a built-in emulator, allowing testing of apps before they are completed for more robust development, as well as tools to improve code quality and maintainability.

10.1.1.4 User Centred Design

User-centred design (UCD) is a design approach that focuses on the needs, behaviours, and preferences of end users at every stage of the development process. The first step of developing an application using UCD is understanding the user base. In this case, it is PwD and their caregivers. To understand the needs of these people, user personas were used (Appendix A) to simulate different users of the system, suggesting different stages of dementia, different cognitive abilities and effects of the condition, as well as different types of family members.

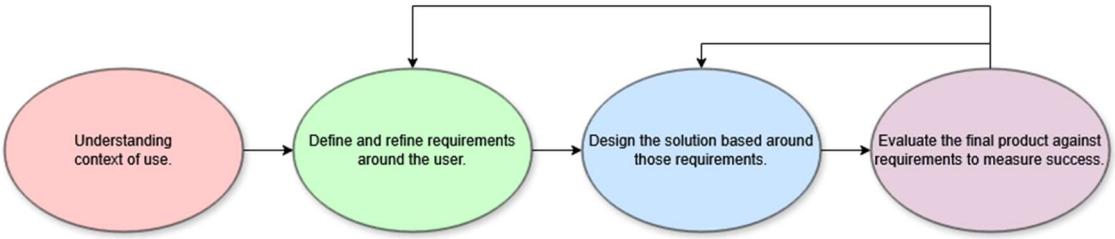

Figure 10.1: Diagram representing the flow of User Centred Design.

10.1.2 Google Firebase

Firebase is a mobile and web development program designed to support application development as a backend-as-a-service platform, providing a suite of tools to do so. Specifically for this project, we use Cloud Firestore as well as Firebase Authentication.

Cloud Firestore is a flexible database allowing synchronous cloud data storage. Firestore is used to hold user data in a No-SQL format using Collections and Documents. A unique quirk of a Firestore is that you may have nested Collections allowing you to layer data while maintaining efficient access. This is useful when handling many types of data with many users, as we are in this project.

Firebase Authentication is used for the log-in system, allowing for secure email-password logins stored in one central, trusted service for access across devices. This means the application can remain secure without incurring the time overhead of developing more advanced log-in systems manually.

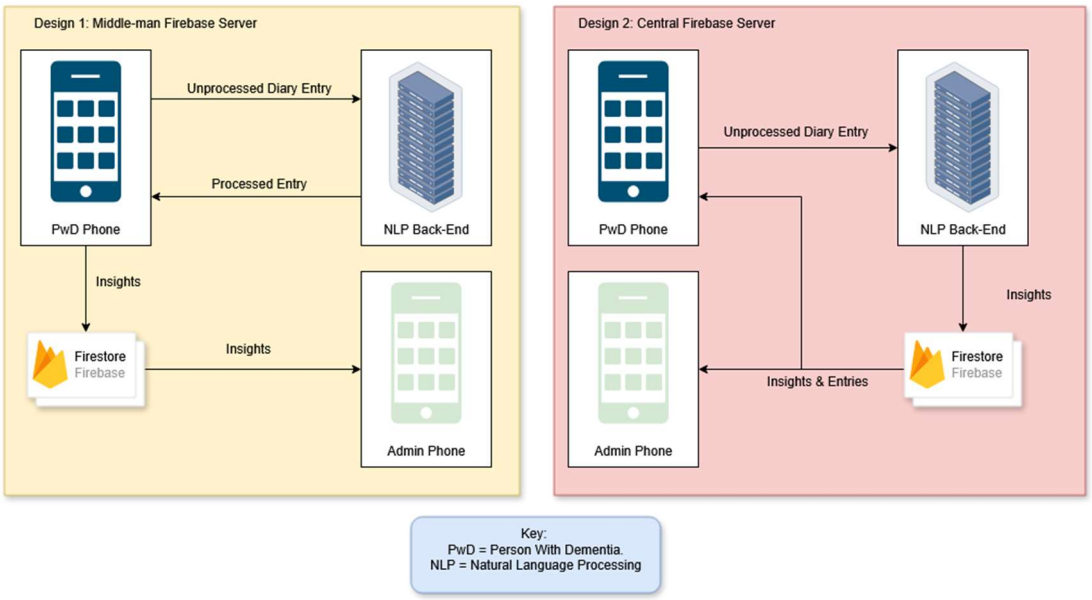

Figure 10.2: Potential System Designs using Firestore.

10.1.3 Natural Language Processing

10.1.3.1 Python

Python is a scripting language popular for its lightweight nature, and powerful mathematical libraries, used widely in NLP. Specifically, Python is used to implement the back end, which includes all NLP elements, as well as the implementation of MiRo software.

10.1.3.2 Scikit-Learn

Scikit provides "state-of-the-art machine learning algorithms" (Pedregosa *et al.*, 2011) and is built atop of Python, using libraries such as NumPy and SciPy. In this project, Scikit is used for its text classification tools, enabling processing of journal entries. The benefits are many, but the standout features are its ease of use, with pre-packaged classification tools "bringing machine learning to non-specialists". Scikit is also open source, with a large community of developers and support, as well as useful documentation and existing works.

10.1.3.3 Natural Language Toolkit

Natural Language Toolkit (NLTK) is an open-source Python library "including extensive software, data and documentation" (Bird, Loper and Klien, 2009) for working with text and text processing. Its main features are text tokenization, tagging, and parsing via lemmatization or stemming. For this project, NLTK is used for pre-processing steps, taking in the journal inputs and transforming them into machine readable text before classification. The main benefits being the large suite of tools available, and the interoperability between components for seamless text processing, making it much easier to correctly prepare text for processing using tools like Scikit.

10.1.3.4 Bag-Of-Words & Term Frequency-Inverse Document Frequency

A bag-of-words (BoW) model is a "feature extraction technique [for] modelling text data" (Murel and Kavlakoglu, 2024) as part of a text processing pipeline. Commonly used in NLP, BoW models can effectively represent large bodies of text using quantitative representation of the data, to make predictions based on patterns that appear in the data. The main benefit of BoW is this synergy with classification tasks, which play a significant role in this dissertation. It allows you to look at a body of text statistically to extrapolate documents with "similar content [and] type". This is highly useful when trying to predict user

mood from a given text, and can be done by training BoW model with a large dataset of existing values, then comparing those values against the user text.

Term Frequency-Inverse Document Frequency (TF-IDF) builds on top of BoW, by improving the representation using word weights based on the level of importance of each word. TF-IDF is "widely used in information retrieval [...] to identify the most relevant terms in a document" (Salton and Buckley, 1988), which helps to give higher weights to words that are frequent in a document but rare in a corpus, identifying words that may be distinctive to a specific document and is more likely to provide useful meaning. This has the potential to improve classification performance and leads to more accurate predictions.

10.1.3.5 Supervised Learning

Supervised Learning (SL) is a "machine learning technique that uses labelled data to train an AI model to identify the underlying patterns and relationships" (Belcic and Stryker, 2024). We used SL with a large corpus of tweets with sentiment labels, either positive, negative, or neutral, to train a BoW model to identify mood in each text. The benefit of this is that we can use existing, proven, datasets in the prediction model, guaranteeing it will produce expected results for the targeted purpose e.g., mood detection from a tried and tested sentiment analysis dataset.

Figure 10.3: Example use of NLP Tools.

10.1.4 Social Robot

10.1.4.1 MiRo Social Robot

A MiRo social robot is "a fully programmable […] platform for companion and social robotics" (Prescott et al., 2018) that can be controlled remotely over Wi-Fi using an instance Robot Operating System on its onboard controller. MiRo is used to interface with users, able to take in voice inputs and send them to the back end to be processed. The benefits of this are many. Studies have shown that social robots have positive effects for PwD (Faisal et al., 2024), specifically with companionship and loneliness. Additionally, the fine motor skills required to interact with a mobile application can be limited by dementia and age, proving difficult to use. MiRo replaces these interactions with speech, which has multiple advantages. Firstly, this encourages PwD to talk out loud, stimulating areas of the brain that may not be commonly used e.g., if they live alone. This has been shown to improve cognitive ability (Quinlan and Taylor, 2013) and even potentially reduce the progress of dementia symptoms. Secondly, it removes the need to use the touch screen for most interactions. Entries created using MiRo automatically appear on the application, allowing Admins to view insights amongst other usual functionality, improving the accessibility of the system.

10.1.4.2 Robot Operating System

Robot Operating System (ROS) is a "set of software libraries [for building] robot applications" (*ROS.org,* 2024). It provides operating system tools for connecting and interacting with ROS powered robots, such as MiRo. This dissertation uses ROS tools to implement features using MiRo to enable communication with users using a VUI. Specifically, MiRo can communicate with an external computer over a Wi-Fi network, allowing remote access to its sensors and motors for complete control. ROS also supports ROS-Bridge, an extension allowing ROS commands to be implemented in Python. This is very useful as it allows us to define behaviours for MiRo with scripts, making working with MiRo more maintainable and predictable, while also giving access to NLP tools via HTTP requests to the back end.

10.1.4.3 Ubuntu & VirtualBox

Ubuntu is a popular Linux distribution with an emphasis on user experience and ease of use. Linux is needed for this project to interact with MiRo and use ROS (only supported on Linux). This means that Linux is a

must have for MiRo to work correctly. ROS is supported on multiple Linux distributions; however, Ubuntu was chosen due to its clear interfaces that make it easier to use out of the box.

We use Ubuntu as a Virtual Machine with VirtualBox (a software for emulation). This has multiple benefits over natively installing Ubuntu, the main one being the option for Snapshots, a feature offered by VirtualBox that allows us to save the state of the operating system before performing dangerous actions. This is especially useful when working with ROS, which requires complex configuration and has many dependencies, so being able to save a working state is very useful. Additionally, to enable detailed integration testing, the project requires both Ubuntu and Windows to run simultaneously, which is only possible with a Virtual Machine.

10.1.5 Postman

Postman is software that provides an interface and tools to quickly simulate API requests, as well as the option to save and edit previous requests for repeated testing. Postman is used extensively in this project to test HTTP connections between the Android application and PyB, as well as test back-end responses. This is useful for testing NLP pipelines by sending simulated user entries and checking the responses for correctness, and for testing the HTTP systems themselves, ensuring they are working and connected.

10.1.6 Google Cloud Speech-To-Text

Google Cloud Speech-To-Text (GCST) is a cloud-based Speech-to-Text service that converts audio of spoken words into text using advanced machine learning models. Powered by Google Cloud, it uses external servers to process data. It uses a speech model trained on “millions of hours of audio data and billions of text sentences” (Google LLC, 2024). GCST was chosen due to its external API model, completing the work on Google servers and removing the need for a heavy processing overhead, as well as its state-of-the-art deep learning model for high accuracy.

10.1.7 Dataset Gathering

10.1.7.1 Kaggle

Kaggle is an online community for data scientists and provides a library of thousands of datasets and machine learning models. For this project, it is used to find supervised datasets for training a sentiment detection model for the NLP pipeline, specifically the dataset of tweets.

10.1.7.2 Data Collection and Labelling

This project uses data processing techniques to remove unnecessary datapoints to improve relevancy. Data Collection (DC) and Data Labelling (DL) are used for this scenario, in the form of Python scripts. As this project requires specific data, it is required to process any datasets that are acquired from external sources. Here DL is used to filter through results and give more relevant labels to each data point, for improved model training. Furthermore, DC is used to form simple datasets, made up of static data such as time detection. By manually choosing what goes in these datasets, we achieve a finer control over each model, allowing more intricate model tuning.

11 Design

This section discusses the design of the system components, including accessibility considerations and high-level design and algorithm decisions.

11.1.1 Android Application

11.1.1.1 Usability Considerations

As this application is targeting PwD, it is important to think about the different types of accessibility it should include (visual, fine motor, cognitive, etc) and approach this with a user centred design philosophy. To do this, a comprehensive set of user stories were produced (Appendix A), encompassing various age groups and life situations. This gives us a broader view of the potential user motivations we can expect, and

identifies areas of weakness with user-application interactions, allowing us to design with this in mind. The following section highlights some of the key areas we identified from this planning.

A critical component of user interface (UI) design are the on-screen components, such as buttons and text fields. These components are used for all interactions with the mobile application, and are tied directly to application functionality. A problem area that was identified were the tremors and reduced motor skills found frequently with PwD, affecting how these users can interact with smaller UI components. To account for this, we increase the size of all buttons to follow the standard set by Googles Material Design (MD) guidelines, with a minimum of 48dp in area. This increased size gives the user a larger margin of error when attempting to press a component, leading to easier interactions. Similarly, we increased the size of text to address any visual impairments, adhering to MD readability guidelines, making the text significantly easier to read.

Another area we identified was colour blindness, and ability to differentiate elements on the screen. To address this, we adhered to a strict colour palette throughout the application, increasing learnability with use over time, as each component has a recognisable colour that is consistent throughout each page (for example all buttons are purple with white text). Picking the colour scheme also posed the issue of finding contrasting colours to improve readability for each type of colour blindness, the solution we chose was to ensure all text is highly contrasting against lighter backgrounds, and that buttons are easily recognisable by shape and location, with labels for functionality.

Intuitive navigation is another key component when designing for older PwD, and we achieved this by implementing a distinct navigational structure. For each activity, we provided a task focused UI to minimize confusion and stress during use, being able to easily identify the purpose of each page. We did this by including consistently placed labels and buttons while keeping sub-menus to a minimum to avoid getting lost in confusing structures. This way of designing uses linear pathways to navigate the system, with all pages leading back to a central hub, reducing the number of steps required to complete tasks. Planning this required the use of a structure diagram (Figure 11.7), which aided in picking where each component should go, and how a user can access it.

11.1.1.2 Navigation & Internal Structure

When users launch the application, they are greeted by a login page (Figure 11.1). This page provides clear and identifiable data input fields to enter an email address and password, ensuring accessibility through labelling. The system keyboard is used to enter data into these fields, allowing the application to work with any custom keyboard presets the user may have. For new users, an intuitive sign-up button is highlighted at the bottom of the page, directing the user to a separate screen with fields to create an account. The sign-up process is given with step-by-step fields, and error messages to indicate what is wrong with a current input in the form of Toast popups. Upon successful sign-up, the user is redirected back to the login page where they can use their new details to gain access to the application. This is done to maintain a consistent and predictable application flow.

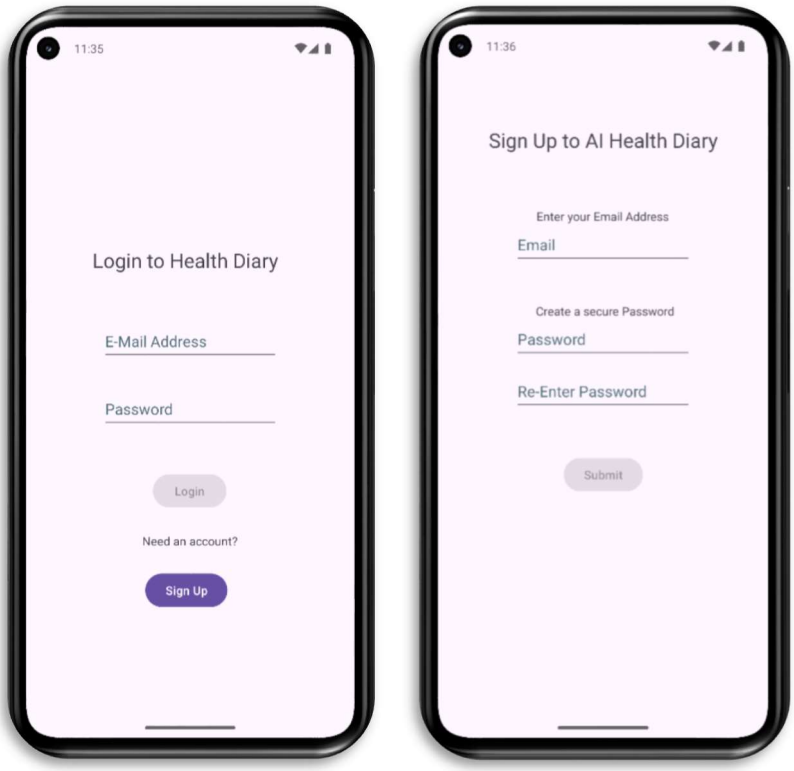

Figure 11.1: Login and Sign-up Pages.

The Hub page (Figure 11.2) provides navigation to all other application features. Navigation is facilitated by intentionally large, labelled buttons to address the needs of those with limited motor skills. To note is the consistent colour scheme throughout the application, made up of contrasting text and clearly defined interactive objects. This is done for improved visibility for those with visual impairments. Similarly, the use of recognisable icons and descriptive text labels that indicate button functionality improve accessibility for a more diverse range of users.

Another feature of the Hub is its responsibility in starting the reminder notification (Figure 11.3) for the application. This notification is used to support the daily lives of PwD by providing medication and event prompts to help organise the day ahead. Each reminder is structured with a clear title summarising the insight and a brief text body containing quick, essential information at a glance. To allow for user control, the notification includes an easily accessible button to dismiss it, which prompts the application to move onto the next reminder in the sequence. Reminders are designed to be linear, meaning only one is shown at a time in a queue system, with the head of the queue needing to be dismissed before moving onto the next item. This is done to simplify the reminder system, reducing stress caused by multiple notifications with differing dates and times appearing at once. By reducing this down to the closest due reminder, we remove this stress, allowing the user to focus on one task at a time.

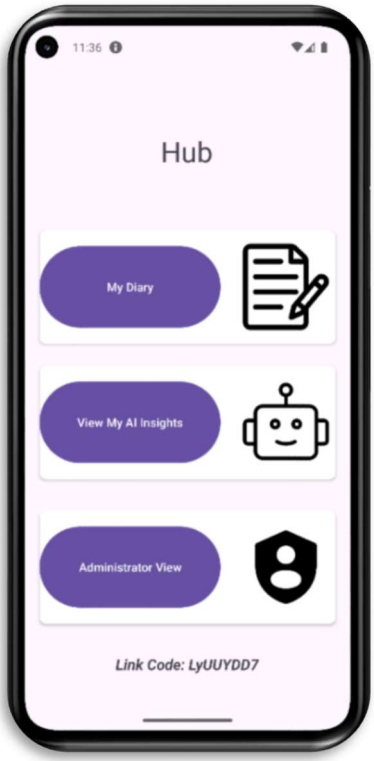

Figure 11.2: Hub Page.

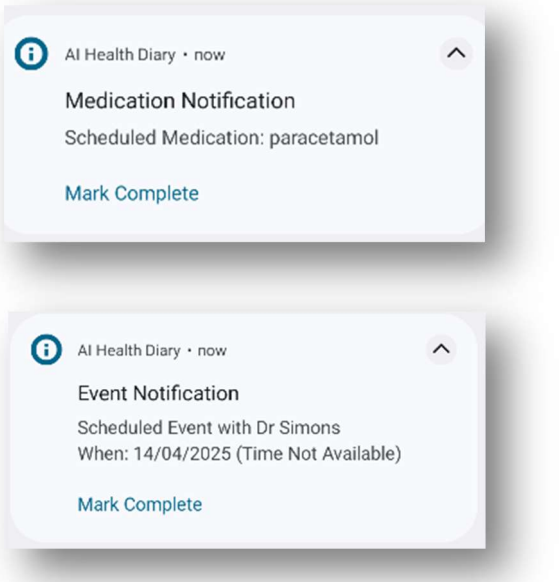

Figure 11.3: Example Notifications.

Leading on from the Hub page is the Diary Entry page (Figure 11.4). This page includes clear headings for each entry to clearly identify entries apart, as well as a smaller sub-heading for the last edited date. This was done to make finding the correct entry as easy as possible. The list of entries is scrollable, allowing more entries to fit onto a static screen for simplicity, while keeping a consistent format for each individual list item. There are several buttons on this page, each following the same colour coding as previous pages for consistency, and each being clearly labelled with their function in a contrasting colour. The largest button creates a new entry and redirects the user to a creation page.

On this page are large fields that the user can fill in with their new entry details using the system keyboard. The fields are hinted, meaning they contain example text of what should go in each field. We keep the UI simple with only two fields, instead automatically detecting things like date and time behind the scenes. Colour is also used to indicate when an operation is unavailable, as seen with the greyed out delete button, clearly showing the user that they cannot delete an entry that is not yet created. The benefit of this being that the delete button will stay in the same place for an entry that can be deleted, so the user will already know where to find it if they need that functionality in future.

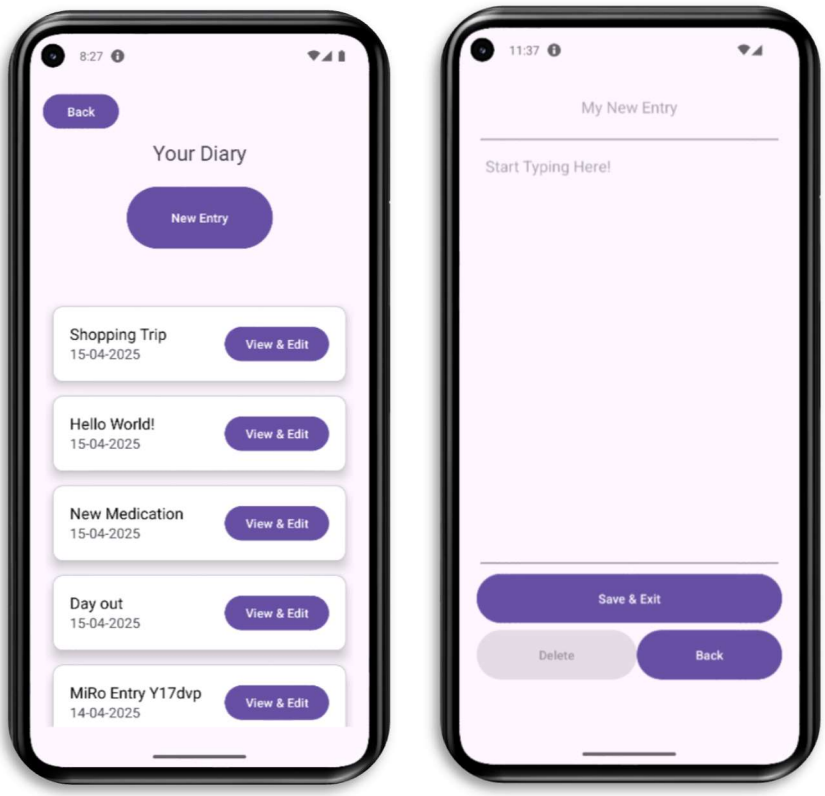

Figure 11.4: Diary Pages.

The AI Insight page (Figure 11.5) displays a list of insights produced by the NLP system. We define an insight as a useful piece of information extracted from a user entry. Insights fall into three categories: Sentiment, Event, and Medication. Sentiment insights contain information about predicted user mood in a given entry. They display this mood with the sentence that prompted that prediction, as well as the corresponding diary entry. Event insights contain information about predicted events, such as location, date and attendees amongst other things. Finaly, medication insights contain extracted medications names, suggested dosages, and side effects categorised from common to very rare.

The Insight page, like the Diary page, populates a scrollable list with each insight, clearly labelling each with various details. The design of this list promotes simplicity by abstracting details away from each entry, instead opting to house these extra details in a sub-menu, accessible via a button on each entry. This was done to ensure the UI is not cluttered by keeping only the most useful information in immediate view.

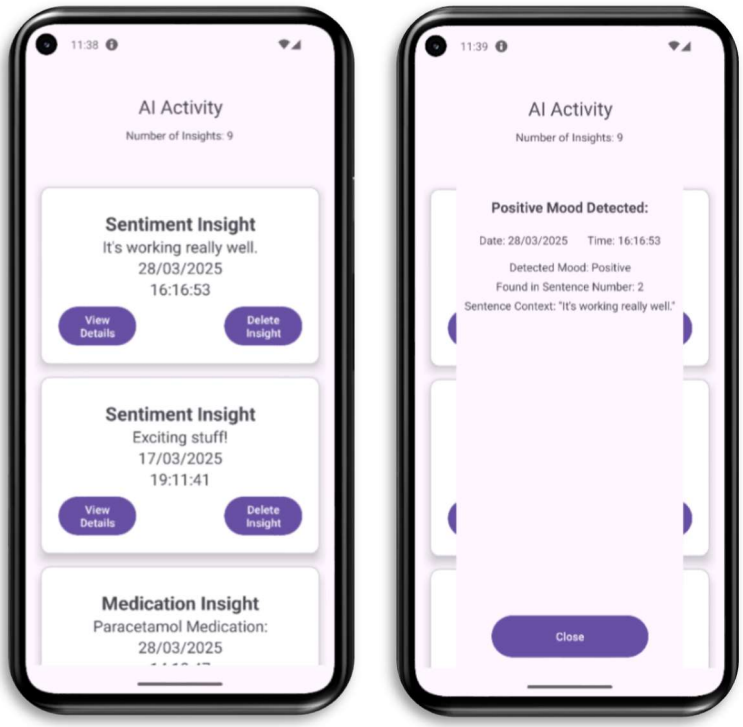

Figure 11.5: AI Insight Page.

Finally, the Admin page (Figure 11.6) is responsible for admin interactions, allowing the connection of accounts for remote monitoring, as well as viewing any insights for connected accounts. For improved ease of use, the Admin page has been split into two distinct sections. The simplest being the registration of a new user. This page is used to register a new member to the admin interface by entering their unique code, found on that users Hub page. It can be accessed via a clearly labelled button on the Admin page, and shows the user a simple UI asking for two text fields. Each text field is labelled to clearly indicate what belongs there and maintains the same theming as the rest of the application. The Admin Insight page follows the same structure as the AI Insight page above, with the main differentiating features being each list item also contains the name of the person that that insight belongs to, done to make it easier to find specific items from the list, as well as replacing the delete button with a button that will set a reminder for that insight on the admins device. This method of setting admin reminders was chosen to make it easier for carers to organise which users and insights they would like to keep track of. We did this in response to feedback from user testing, where participants were worried about being given too much information, unintentionally causing more stress.

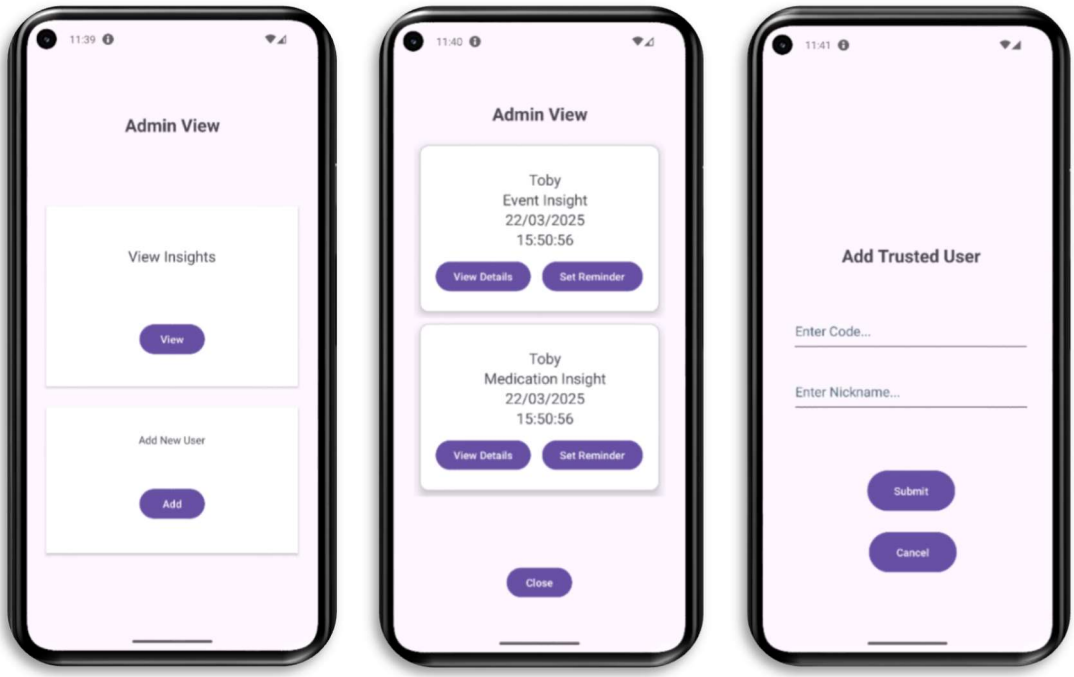

Figure 11.6: Admin Pages.

The following diagram can be used to visualize the overall navigational structure of the mobile application, showing the linear nature of each component, each linking back to the Hub.

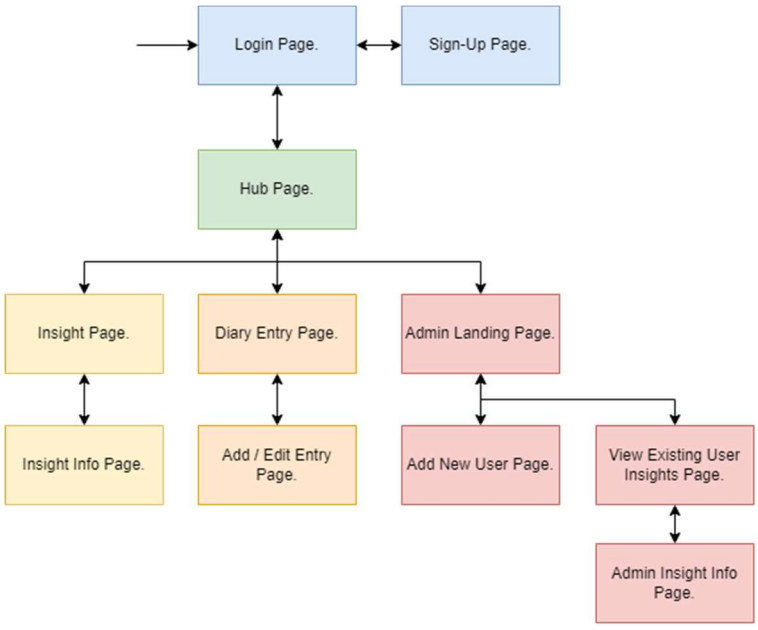

Figure 11.7: Navigational Structure Diagram.

11.1.2 Python Back-End

The PyB element of this software is responsible for producing insights using NLP, and is designed to operate silently in the background. The benefits of this are twofold: Firstly, there is no need to show the users the background processing done by the application. PwD are easily stressed and overloading them with this information would be counterproductive. Secondly, Android as a platform is not designed for intensive tasks such as NLP. Mobile architecture struggles with straining tasks, so offloading this to a typical computer removes this issue, preventing application hanging for more streamlined behaviour.

Users interface with this server by saving entries in the mobile application or reading them out to MiRo. To monitor its behaviour, the PyB has various text-based prompts with messages indicating when new requests are sent and received, as well as monitoring Firestore events. This is especially useful when looking for bugs in the code, as the user can see exactly what is happening as the pipeline progresses.

11.1.3 MiRo Robot

Like the PyB, most of MiRo's processing is done externally on a network linked computer. MiRo is controlled using Python scripts and a ROS Bridge. When activated via the GUI, MiRo will record the users voice as they read out their entry to it. The GUI (Figure 11.8) is made up of a simple labelled button to start the interaction with the robot, as well as a text field that gives updates on the state of the interaction. This was done so that the user can see when MiRo is active and doing something, improving the user experience. Developing the GUI was optional, as MiRo can be controlled using the terminal, however we decided it was necessary to implement, as terminal commands are alien to PwD. The GUI simplifies this process down to just a click.

MiRo is designed to respond while the user is talking by nodding, blinking, moving its ears and head, to indicate to the user that it is listening, showing human like responses that make interacting with it more engaging, which has been shown to improve cognition and slow the effects of dementia over time (Faisal et al., 2024). This addresses the loneliness often experienced by PwD. By using MiRo as a connection point for the application, we can give users the feeling that they are having meaningful interactions with the robot, while simultaneously monitoring their feelings and other insights, which can then be saved and shown to carers for peace of mind. Once the user has stopped talking, MiRo detects this and sends the voice recording to the PyB for processing, producing insights that appear in the mobile application, as well as responding with a finishing animation to indicate the process was successful.

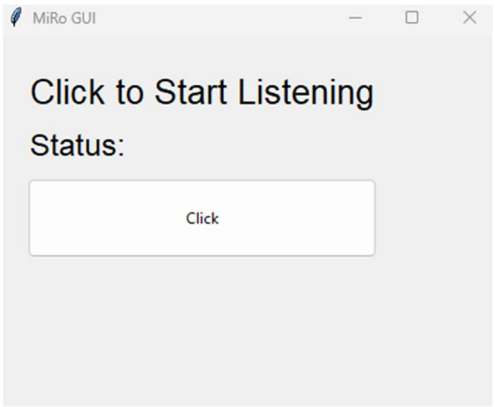

Figure 11.8: Screenshot of Simple MiRo GUI.

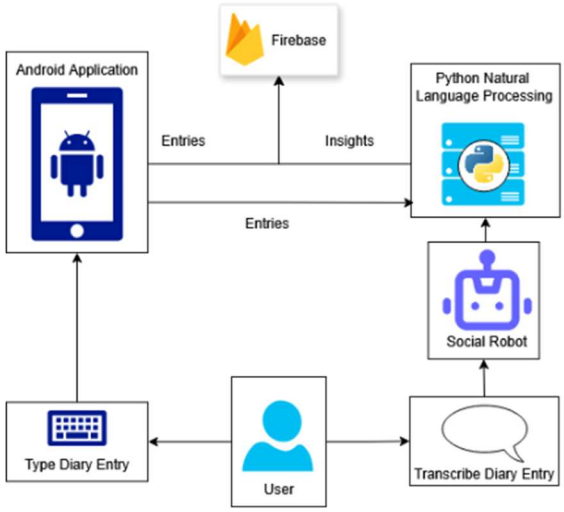

Figure 11.9: Diagram showing system flow.

12 Implementation

This section discusses the implementation of each component, including the designs of relevant algorithms and structures.

12.1.1 Mobile Application

12.1.1.1 Account Management System

Logically, the first interaction that a user has with the application is the Login and Sign-up page. This is the entry point into the application and is implemented using an Android activity component. Activities are used frequently throughout the application as they represent a single, focused thing that the user can do, which is useful in providing a clearer navigational structure. We implement buttons and text fields as Android widgets, ensuring we declare any text as a variable in XML rather than hardcoding, as this improves maintainability and flexibility over time. We use a Fragment, hosted within the Login activity, for the Sign-up page, promoting the reusability of components and a well organised activity lifecycle.

To implement the login functionality, we used Firebase Authentication (Auth) to manage and store email-password combinations with API calls to send user details to be verified. The class responsible for this, "FirebaseAuthHelper" is also responsible for creating and managing the current user by holding the current user-ID to verify other Firestore operations, such as creating new entries. Additionally, it is at this stage where a unique code for each user is generated on a successful login. This was done using the SecureRandom library, which can create strong random numbers by unpredictably seeding each value. Doing this ensures that codes are secure and cannot be maliciously brute forced.

```
public static void loginUser(String email, String password, FirebaseAuth
mAuth, Context context) {
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    // Login successful
                    Toast.makeText(context, "Login Successful",
                      Toast.LENGTH_SHORT).show();

                    // Generate a code for the current user.
                    String permanentToken = generateRandomCode(8);
                    FirebaseUtil util = new FirebaseUtil();
                    util.addSecurityCode(permanentToken,
                        mAuth.getCurrentUser().getUid());
                    // Redirect to HubActivity
                    Intent moveToHubIntent = new Intent(context,
                        HubActivity.class);
                    context.startActivity(moveToHubIntent);
                } else {
                    // If login fails, handle the error...
```

Figure 12.1: Extract of Login using Firebase Authentication.

12.1.1.2 Data Storage

Early on in development, the application was designed to use a Room database to locally store user data on their device, and Auth to handle login data. The rationale behind this was to alleviate security risks and provide an internet independent application. However, it became clear that a more sophisticated system was needed as it would be difficult to enable data sharing between accounts using local data storage. To address this, Firestore is now used to store all the structured data needed for the application using a Collection, Document system. The benefit of this is easy access to data across devices using database queries, as well as its natural integration with Android and Auth, allowing Firestore to implement security privileges using Auth and Firestore rules (Figure 12.2).

```
service cloud.firestore {
  match /databases/{database}/documents {

    // Allow users to read and write their own profile data
    match /users/{userId} {
      allow read, write: if request.auth.uid == userId;

      // Allow users to write entries under their profile
      match /entries/{entryId} {
        allow read, write: if request.auth.uid == userId;
```

Figure 12.2: Firestore Rules using Firebase Auth.

Implemented is the structure shown in Figure 12.3. Fields in quotation marks represent the documents that belong in that collection, with the final collection "permissions" being used for the admin system. You will notice the abundant use of sub-collections, this makes it easier to sort insights by type and unlocks the option for more specificity in database queries for more efficient API calls, which is useful when dealing with on-the-fly asynchronous database calls from the mobile application.

```
Collection: users (Top-Level)
        Subcollection: access_codes
        Subcollection: users
            Document: "user_UID_here"
                Subcollection: entries
                    Document/s: "Diary Entries"
                Subcollection: event_insight
                    Document/s: "Event Insights"
                Subcollection: med_insight
                    Document/s: "Medication Insights"
                Subcollection: senti_insight
                    Document/s: "Sentiment Insights"
                Subcollection: permissions
                    Document/s: "Admin Links"
```

Figure 12.3: Representation of Firestore Structure

To handle Firestore access, the class "FirebaseUtil" was implemented. As Firestore uses an API, it requires asynchronous operations. For this, FirebaseUtil was designed to be callback driven to avoid UI delays when waiting for API responses. This has benefits for modularisation, as it means components can handle results of database queries differently depending on their needs. However, one downside to this approach is avoiding recursive callbacks, leading to a sluggish and unstable application. To combat this, each component that accesses FirebaseUtil implements a callback interface that is no more than two callbacks deep, preventing "callback hell" for a more responsive application.

```
public void getAllEntries(FirestoreCallback callback){
    // Get the user UID.
    String uid = FirebaseAuthHelper.get_user_id();
    if(uid != null && !uid.isEmpty()){
        // Get the instance of Firestore.
        db = FirebaseFirestore.getInstance();
        // Reference to the entries collection.
        CollectionReference entriesRef = db.collection("users")
                .document(uid)
                .collection("entries");

        // Wait for the task to complete asynchronously
        entriesRef.get().addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
```

```
                List<DiaryEntry> diaryEntries = new ArrayList<>();
                for (QueryDocumentSnapshot document : task.getResult()) {
                    // Convert Firestore document to DiaryEntry object
                    DiaryEntry entry = document.toObject(DiaryEntry.class);
                    diaryEntries.add(entry);
                }
                // Use a callback to handle returned entries.
                callback.onReturnEntries(diaryEntries);

            } else {
                // Handle task failure...
```

Figure 12.4: Example FirebaseUtil Method using Callbacks.

12.1.1.3 Hub Page

Implemented as an activity, the Hub provides access to all other parts of the application using a series of buttons, created using Android widgets, and descriptive images, created using Android Image Assets. These buttons redirect users to different activities using Intents, which tell the operating system which activity to open when a button is pressed. Intents were used as they tie in well with lifecycle management, resulting in a more efficient application.

Another role of the Hub is obtaining and checking runtime permissions. Runtime permissions are needed in Android for more dangerous tasks, like showing foreground notifications. These permissions must be explicitly granted by the user before the application can use the associated feature. The permission strategy we use for this is 'Ask Up Front', where the user is prompted to accept permissions as they load the application. This was used due to the range of different permissions needed, as well as wanting to reduce disruption while the application is in use, which becomes especially important if the user has dementia, due to the stress and confusion it may cause.

12.1.1.4 Diary Entry Page

The Diary Entry activity presents a list of existing entries using a RecyclerView component. RecyclerView is an adaptive list that can dynamically create list items using a given format for provided data, implemented in XML. From this, we define unique behaviours for each list item, for example telling it which data to use to populate sections of the diary edit fragment when the edit button is pressed. We chose to implement the edit screen as a Fragment as it shares data and functionality with the Diary Entry activity.

12.1.1.5 AI Insights Page

Like the Diary Entry page, the AI Insights page is implemented using a RecyclerView and adaptive sub-views defined in XML. We define a RecyclerViewAdapter to differentiate between each type of insight and reformat the XML according to which data is available. Doing this means that we don't have unused fields taking up screen space, improving the simplicity of the UI. We implement a Fragment showing a more detailed view of each insight, and data is pulled from each insight object, used to populate the fields of the Fragment.

12.1.1.6 Admin Page

The admin activity is responsible for allowing carers to view their patients' insights remotely. This is implemented in two sections to improve usability. Firstly, the registration page, which uses a Fragment, takes in an input code and nickname and checks this code against the Firestore collection of unique codes to find a match using "AuthHelper" (Figure 12.5). If a match is found, a link is formed between the admin user and the corresponding user who provided the code. We do this by adding the corresponding UID to a subcollection 'permissions', which is checked each time a user starts the admin page in order to get connected user insights, the flow of this can be seen in Figure 12.6. Codes are automatically generated when a user logs in and are visible from the Hub page. We implemented an extra layer of security by changing the codes each time the user logs in, meaning that the same code cannot be used indefinitely. This is beneficial as it helps protect the privacy of users.

Secondly is a page where the admin user can view a list of all the insights for their registered users. This follows a similar format to the AI Insight page, implemented using a RecyclerView and Adapter, however, a unique element is a new button to each insight, allowing the admin to manually set a reminder for that insight on their own device. We opted for a manual system as one of the concerns raised during Patient and Public Involvement feedback involved carers being overwhelmed by many notifications from many registered users. To address this, admins must now manually pick which insights they want to appear as reminders, simplifying organisation and reducing stress.

```
if (uid != null && !uid.isEmpty()) {
   db = FirebaseFirestore.getInstance();
   // Find the user with the access code
   CollectionReference collectionReference = db.collection("access_codes");
   // Find the document with the correct access code.
   collectionReference.whereEqualTo("access_code", code)
   .get().addOnCompleteListener(task -> {
       if (task.isSuccessful()) {
           if (!task.getResult().isEmpty()) {
               DocumentSnapshot document = task.getResult()
               .getDocuments().get(0);
               // Get the UID of the target.
               String target_uid = document.get("uid").toString();
               callback.onSuccess(target_uid, nickname);
       }
```

Figure 12.5: Extract of Admin Code Checking

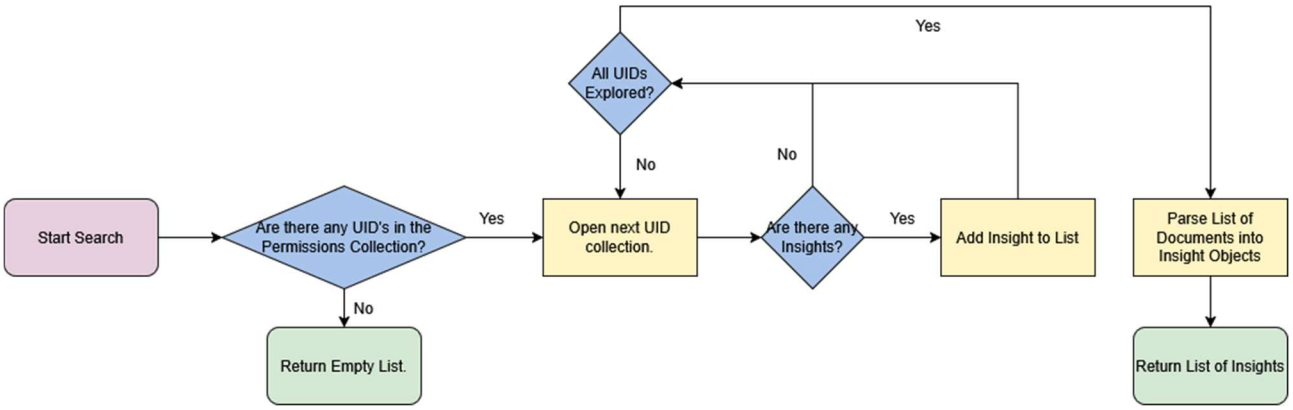

Figure 12.6: Admin Insight System Flow Diagram.

12.1.1.7 Reminder Service

A crucial part of the system is the use of notifications for automatic reminders. To implement this, we used a foreground service and associated notification. We used a foreground service as they have a higher priority in the Android Runtime, making it less likely to be destroyed if the system reclaims resources. This is good, as it means the notification will run for a long time to aid with daily organisation.

The service runs indefinitely in the background, periodically checking for new insights using FirestoreUtil. It does this by maintaining a queue of Reminder objects using a linked list. Reminder objects are made up of an Insight object, due date and time, as well as details to be displayed. When a Reminder is at the head of the list, it is used as the currently displayed notification. Dismissing a Reminder will remove it from the linked list, pushing the next reminder forward to take its place. Changes in the linked list are acted upon by checking the active notification against the head of the list and updating the head when a new Reminder is added with a closer due date. Checks are automatically carried out every minute by a background executor, which periodically calls the “checkReminder” method (Figure 12.7) to carry out these actions. We used an executor as a way of asynchronous checking, without blocking normal functionality, allowing us to pick the interval at which checks are carried out. A persistent notification is used as a constant reminder to the user

that there is a task due. Once a task has been completed, the user can press a button on the notification to dismiss it, also checking for the next reminder in sequence. This sequential method of handling reminders is a good way to keep day-to-day organization simple, enabling users to focus on one activity at a time.

```
private void startReminderCheck() {
    // Start the thread checking reminders
    scheduler.scheduleWithFixedDelay(this::checkReminders, 0, 60,
    TimeUnit.SECONDS);
}

private void notifyNewReminder(){
    // Check if there are any reminders
    if(current_reminder != null && !reminder_queue.isEmpty()){
        // Check the head of the list has not changed.
        if(current_reminder != reminder_queue.getFirst()){
            // If it has changed, update the reminder and current
            current_reminder = reminder_queue.getFirst();
            setNotification();
        }
    }
    // Do nothing if it is the same.
    else{ checkReminders(); };
}

private void addReminder(Reminder reminder) {
    // Null checks
    if (reminder == null || reminder.getDue_date() == null) { return; }
    // Find the correct place to insert the new reminder
    for (int i = 0; i < reminder_queue.size(); i++) {
      Reminder currentReminder = reminder_queue.get(i);
      // Compare dates of current and new
      if(reminder.getDue_date().compareTo(currentReminder.getDue_date())<0) {
            reminder_queue.add(i, reminder);
            return;
            }
      }
      reminder_queue.add(reminder);
    }
```

Figure 12.7: Extract of Reminder Service.

12.1.2 Python Back-End

12.1.2.1 Flask

The entry point to the PyB uses Flask. This library enables the PyB to open a communication tunnel for incoming HTTP requests from the mobile application. To ensure the integrity of this connection, we enable payload checking to confirm that incoming HTTP POST request have the correct payload, matching the specification of data needed for NLP. This improves the robustness of the connection, preventing incorrect data from passing through.

12.1.2.2 Pre-Processing Pipeline

Pre-processing is the first step in NLP tasks. It boils down to turning large raw text inputs into machine understandable, formatted data, so that we can feed the results into various processing algorithms (Hardeniya *et al.*, 2016). As such, after the raw text data is obtained, either from the application using Flask, or directly from MiRo, we run the pre-processing pipeline. This pipeline consists of multiple steps to obtain machine understandable data. Firstly, we tokenise the raw text into sentences using the NLTK library. This enables us to handle the text sentence by sentence, an effective way to look at each context individually for optimal

understanding. Following this, we then pass each sentence to “token_pipeline”, which was implemented to transform each sentence token into word tokens, applying other text manipulations along the way. These steps are as follows:

1. Transform to lowercase.
2. Remove any punctuation using the String punctuation corpus.
3. Filter out any non UTF-8 Characters, replacing them with their closest UTF-8 counterparts.
4. Tokenise the sentence into word tokens.
5. Remove tokens that are stop words*.

We are then left with a list of individual formatted tokens.

* Stop words are filler words like ‘the’, ‘if’, ‘at’ which do not add any useful meaning to the sentence for NLP. We remove these to reduce noise and improve classifier accuracy.

“James sat on the chair.”	{“james”, “sat”, “chair”}

Figure 12.8: Mock Result of Token Pipeline.

Next, we pass each list of tokens to “lemmatise_tokens”, which is responsible for Part-of-Speech (POS) tagging and lemmatisation. POS tagging is the process of converting a list of tokens into a list of tuples (Hardeniya *et al.*, 2016), where each tuple consists of the token and a tag representing the words grammatical function in the sentence. This allows us to apply each words grammatical function to our classifier to make accurate predictions using context, which is important sentiment analysis as words can have different meanings depending on their position in a sentence. Following POS tagging, we use NLKT WordNetLemmatizer to lemmatise each sentence, reducing each word to its dictionary root. This allows us to create a bag-of-words model, using the frequency of each word root in each sentence to make more accurate predictions. We implemented lemmatisation, rather than a stemming as lemmatisation produces more accurate word roots, taking more time to do so, which is appropriate when working with small to medium bodies of text such as diary entries. This means we can have the benefit of lemmatisation with less of a performance hit.

{“james”, “sat”, “chair”}	*{(“james”, NOUN) (“sat”, VER), (“chair”, NOUN)}*

Figure 12.9: Mock Result of “lemmatise_tokens” method.

The final step in pre-processing is Named Entity Recognition (NER). NER can use POS tags to determine contextual meaning about words, extracting things like names, businesses, dates, locations etc. This is exceptionally useful for automatic event detection, as we can use the tagged meanings of words to work out features of events. In this implementation, we use Spacey NER, as it works well with NLTK and has pre-trained and tested models, making its output more likely to be accurate. NER overwrites existing POS tags in a tuple array with recognised word meanings, so as a result, we store the output of NER in a separate array, so we can still access the POS tagged tokens. This is important as each detection system requires different elements of analysed text to be able to function correctly, so it is useful to keep both POS tagging and NER.

One problem we encountered using this design were missing NER tags for street names and addresses, a major issue as we need this information to automatically plan events. To resolve this, we implemented regular expressions and pattern matching to improve street detection. We use Python RegEx to look for common words associated with addresses, i.e., ‘road’, ‘lane’, ‘boulevard’, and use these to extract street names from the text, which may be missed by Spacey NER. These are then added to the array under the tag ‘STREET’.

{(“james”, NOUN) (“sat”, VER), (“chair”, NOUN)}	“lemmatised_text” = *{(“james”, NOUN) (“sat”, VER), (“chair”, NOUN)}* *“named_entity_recogition” = {(“james”, PERSON)}*

Figure 12.10: Mock Final Output of Pre-Processing.

```
def pre_process_pipeline(text_body):
    # Check if String is empty
    if len(text_body) == 0:
        # Return an empty dictionary
        return  {"lemmatised_text" : {},"named_entity" : []}
    # Tokenise into sentences
    tokenised_sents = tokenize_to_sentences(text_body)
    # Define a dictionary to store result. Each index represents a new
      sentence of tokens.
    token_dictionary = {}
    for index in range(0, len(tokenised_sents)):
        # Get sentence to process
        sentence = tokenised_sents[index]
        # Process it and get result
        token_array = token_pipeline(sentence)
        # Lemmatise and POS tag the value
        lemmatised_array = lemmatise_tokens(token_array)
        # Store result in dictionary if it exists
        if len(lemmatised_array) > 0:
            token_dictionary[index] = lemmatised_array

    # Named Entity Recognition
    named_array = named_entity_recognition(text_body)
    # Return processed text
    processed_text = {
        "lemmatised_text" : token_dictionary,
        "named_entity" : named_array,
        "sentences" : tokenised_sents
    }
    return processed_text
```

Figure 12.11: Pre-Processing Method Code Extract.

12.1.2.3 Corpus Preparation

Now that we have our processed user text, we need something to compare it to. This is done using a large dataset (corpus) consisting of relevant text data that is used to train a text classifier. As this application looks for two different types of data, sentiment and medication (events are handled differently), two corpuses were needed. The first is used for sentiment analysis. This corpus was obtained using Kaggle and consists of ~43,000 labelled tweets of varying length and context. Tweets were chosen as they often represent people's inner dialog, which is closely comparable to sentences written in a diary or journal. The initial corpus is not suitable for use in training a classifier, so a script was implemented to filter and format the corpus automatically. It does this by reading each tweet and associated label, processing it using the same pipeline as used for user entries, then writing it to a new csv, resulting in a usable labelled dataset. We use the same pipeline as that of user entries as a consistent format is needed to produce more accurate predictions when using a classifier, reducing potential bias. The second corpus for medication analysis is more complex in structure. It is composed of a list of medication names, their recommended dosages, known side effects and how common those effects are. To make this, a web scraping script was implemented and used to extract data from the British National Formulary, which includes all the data we need. This script systematically crawls through the data using the BeautifulSoup library, constructing new URL's to explore as it discovers new medications. It then formats this data using RegEx and similar steps to the pre-processing pipeline before storing it in a csv file. The resulting corpus is made up of ~1500 medication entries.

12.1.2.4 Sentiment Analysis

The first stage of analysis is looking at Sentiment, deciding if a sentence is overall positive, negative or neutral. We did this by training a classifier on the corpus of labelled tweets. We implement a method, "generate_classifer" to generate a classifier, which uses a 75-25 stratified train-test split using the Scikit

library. Scikit provides implementations of different classifiers and allows different methods of testing classifier performance for optimisation and parameter tuning. We first create training and testing datasets, 75% of the data is chosen using stratified sampling to reduce sampling error, which would negatively affect the performance of the model. These datasets are then transformed into a TF-IDF matrix, which label each datapoint with a level of importance based on its frequency of appearance in the text, balancing based on rarer words, which may hold more meaning. Using TF-IDF means we can compare the similarity of two sentences by looking at each matrix and their frequency values, allowing us to make predictions about each. We then train a Logistic Regression classifier using this matrix, which is used to make predictions. We discuss the results of this testing in more detail in Evaluation section 13.1.4.

Finally, the trained corpus is saved to a Pickle file so that it can be reused, as retraining the classifier is resource intensive. This way, we can simply load a pre-trained classifier when needed.

To use the trained classifier, "predict_multi_sentiment" (Figure 12.12) was implemented. This method takes in a text input, and a tuple containing the trained classifier, a vectorizer and a transformer. The text input is vectorized into a matrix and transformed into a TF-IDF matrix, and the classifier is used to predict the similarities between the user data and the trained data. We then take the prediction with the highest probability and return it.

```
def predict_multi_sentiment(text_input, classifier_tuple):
    classifier, vect, transformer = classifier_tuple
    # Join tokens into single sentences if more than one element
    if isinstance(text_input, list):
        text_input = [' '.join(text_input)]
    # Transform to matrix
    text_matrix = vect.transform(text_input)
    # Weight the matrix
    text_weighted_matrix = transformer.transform(text_matrix)
    # Predict the sentiment
    sentiment = classifier.predict(text_weighted_matrix)
    return sentiment
```

Figure 12.12: Prediction Method Implementation.

12.1.2.5 Event Detection

Event detection takes a different approach to text analysis by using the results of NER to determine if there are any events in the text. To do this, "event_analysis" (Figure 12.13) checks the NER tuple for any words labelled as 'DATE'. Having a date in the text inherently suggests an event, and this is used to start the event detection process. We then cross reference this date with the list of raw sentences from the user entry to find the original sentence that the date is from. We compare every word in this sentence against every word in the NER tuple and keep a dictionary of all words that appear in both, as this means they form part of the same event sentence.

Using this dictionary, we look at every event detail and parse it according to its type, disregarding any types that are not useful for planning events. This then forms a complete Event object, which consists of date, time, location and attendees. One important step in this process is date parsing. We do this using the "dateparser" library, which can take a worded date, i.e., '1st of January 2025', and converting it into a numerical date, '01/01/2025'. We do this to keep all dates consistent across events for better maintainability, and to filter out details detected as dates that do not fulfil the requirements to plan an event, like incomplete dates.

The final step in event detection is filtering out bad events. We check each event object and remove any that don't meet the standard of a completed event. Completed events are defined as events with a detected date that is greater than 'today', and either a location or at least one attendee. This step is taken to remove meaningless events that clutter up the system, for example if the user were to mention a date in their entry, but not mean it as an event, this would initially get picked up as an event, before being removed by the filter system.

```
def event_analysis(raw_sentences, ner_dict):
    # Dictionary storing each event object.
    event_dict = {}
    event_counter = 0
    # Check each element of ner to check if it is a date.
    for index in ner_dict:
        if ner_dict[index] == 'DATE':
            # Find the sentence that contains the key.
            for sentence_index in range(len(raw_sentences)):
                sentence = raw_sentences[sentence_index].lower()
                # Look for the string in the dictionary keys.
                if sentence.find(index) != -1:
                    # Set the sentence as the source of the date.
                    source = raw_sentences[sentence_index].lower()
                    # Look for any other relevant details that are in ner
                    detail_dictionary = {}
                    for detail in ner_dict:
                        if detail in source and detail != index:
                            # Add extras to separate dictionary storing the
                              detail and its type.
                            detail_dictionary[detail] = ner_dict[detail]
                    # Add details to an object and store in the dictionary
                      using the event number.
                    event = Event(index, source, detail_dictionary)
                    event_dict[event_counter] = event
                    event_counter += 1
```

Figure 12.13: Extract of Event Detection.

12.1.2.6 Medication Detection

Medication Analysis (Figure 12.14) uses simple comparison-based extraction, as medications are easy to detect using only their names. The first step is to tokenise the list of raw sentences into individual tokens, using the pre-processing pipeline. A unique detail about medication pre-processing is avoiding the use of lemmatisation, as it can interfere with complicated medication names and result in incorrect, non-matching word roots. Once this is done, we load the medication corpus and convert it to a dictionary. This means we can use dictionary lookups which are O(1) efficiency, allowing us to do multiple searches quickly if there is more than one medication in the user text. We then search the medication dictionary for any tokens that match the tokens in our user entry. If any are found, we add the medication name and details to a list. We then check for keywords using a set of related keywords, used to add context to the medication. Finally, we add the detected medication and context to a list and return the list once all the sentences have been explored.

```
# Take the input sentence or text.
def medication_analysis(raw_sents):
    # First check against medication database for any medication that match.
    # Get the database dict from pickle
    med_dict = restore_from_pickle(get_filepath("medication_pickle"))
    # Process each sentence checking for medications
    detected_medications = {}
    for index, sentence in enumerate(raw_sents):
        # Tokenise the sentence
        tokens = token_pipeline(sentence)
        keys = med_dict.keys()
        for token in tokens:
            if token in keys:
                medication = token
                # Array of details
```

```
                details = med_dict[token]
                # Check for keywords
                detected_keywords = check_for_keywords(tokens)
                # Save the sentence as the context for the drug if keywords
                  are detected.
                if len(detected_keywords) > 0:
                    # Add context to the detected medication if relevant.
                    details.append(sentence)

                detected_medications[medication] = details
    return detected_medications
```

Figure 12.14: Extract of Medication Analysis.

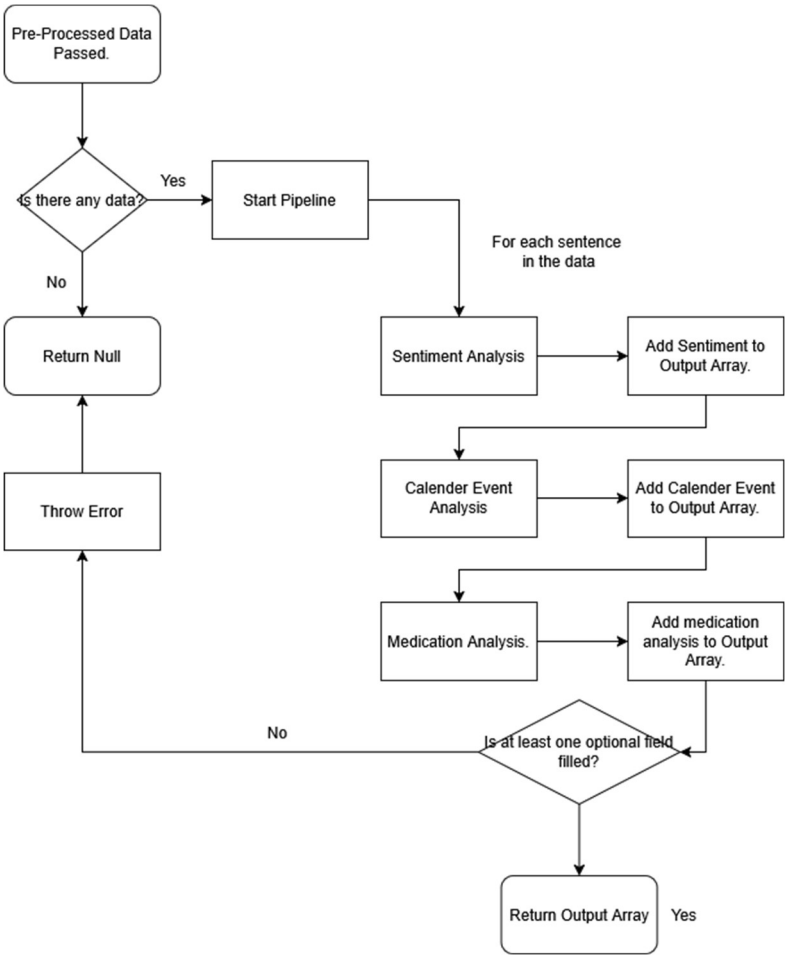

Figure 12.15: Simplified View of Insight Pipeline

12.1.3 MiRo Robot

MiRo was implemented in Python and ROS. We use the “miro2” and “rospy” library to enable ROS commands and MiRo integration in Python scripts. To start the interaction with MiRo, we implemented a simple GUI using Tkinter, removing the need to use terminal commands for ease of use.

To establish a connection with MiRo’s on-board computer, we use RobotInterface, a class which is part of the MiRo MDK, which simplifies MiRo interactions. By initialising RobotInterface, we subscribe to all MiRo’s sensors, and create publishers for pushing actions to the robot, such as movement commands.

Next, we perform a startup animation to indicate a listening state by rotating its ears three times. We do this to improve the interaction with MiRo, making it seem like it is engaged in the conversation by providing visual feedback. We then implemented a method “callback_mics” (Figure 12.16) to handle any messages posted to the microphone audio topic, recording the data to a buffer. We calculate the average volume for the data, sending it to the method “is_silent” which determines if the audio is quiet enough to be determined silent using a constant threshold. If the audio is silent for a given period, the recording is stopped and written to a .wav file.

```
def callback_mics(self, msg):
      # If recording
      if not self.micbuf is None:
          audio_data = np.array(msg.data, dtype=np.int16).reshape(-1, 4)
          # Append mic data to store
          self.micbuf = np.concatenate((self.micbuf, msg.data))

          # Decide if to finish recording
          if self.is_silent(audio_data):
              # Check if silence is long enough
              if time.time()-self.last_non_silent_time>SILENCE_DURATION:
                  print(f"Silence detected for {SILENCE_DURATION}
                  seconds, stopping recording.")
                  self.outbuf = self.micbuf
                  self.micbuf = None
                  print (" OK!")
          else:
              self.last_non_silent_time = time.time()
```

Figure 12.16: Extract of “callback_mics” method.

The second stage is processing this audio recording. Initially, this was implemented as a direct connection to the NLP system, running on the same machine. However, this had to be changed due to incompatibility between Linux libraries running MiRo and ROS, and the Windows based implementation of NLP, resulting in the libraries not working together.

The solution we implemented was to use HTTP to communicate with the Flask API running on the PyB to send recording data. Once MiRo finishes recording, we convert this audio from stereo to mono using the ffmpeg audio processing library. One of the limitations of GCTS is that it is only capable of processing mono audio, so we are required to take this extra processing step. We then send this new file to GCTS where a transcription is returned, and we can extract a text string from the result. We then automatically delete both the stereo and mono audio files to preserve user privacy and save device storage.

From here, we use a Flask connection via HTTP POST request to forward this text to the NLP system, this text is now treated as a regular diary entry, following the same path as described in the NLP insight pipeline and resulting in a created insights stored in the Firestore. We then implemented a method “create_entry” that takes the transcription and formats it into a standard diary entry, storing it in the Firestore with an automatically generated title specifying the entry was created using MiRo. This helps distinguish how the entry was created in case the transcription produced any mistakes.

To improve the perceived friendliness and communicative nature of MiRo, multiple animations were implemented using RobotInterface to make it appear as if MiRo is engaged in the conversation. We defined a set of motions MiRo can make while a recording is taking place, such as blinking, lifting or lowering its head, moving its head side to side, nodding, moving its ears, and combinations of these movements. We implemented a method to decide which of these movements to perform, using random choice selection. This method runs on a separate thread, using the standard threading library, as to not interfere with the audio collection process. This is important for fluid animation as the audio collection loop has a high polling rate and animations take time to perform, resulting in overlapping animations which are jagged and incorrect. Once the thread is running, MiRo can periodically complete one of the defined actions, as well as blinking.

Blinking is implemented to always occur as it makes MiRo seem more natural, as blinking is an inherent part of human facial movements. Another action, “none”, is where MiRo will stay in the same position and blink. This action makes MiRo seem less fidgety and calm, and prevents it from constantly moving around, which may make the user uncomfortable after time. Actions take place after random time periods, so multiple moves can occur one after another, or not at all for up to 6 seconds. This intentional randomness mimics that of humans. Finally, when the user input has finished processing, a shutdown animation is played, letting the user know that the process is complete.

13 Evaluation

This section discusses the various methods of testing carried out to verify the effectiveness and performance of the system.

13.1.1 Unit & Integration Testing

Unit testing was iteratively carried out to test each feature in isolation. This method of testing means that we could confirm each individual method works correctly, giving an opportunity to catch bugs and improve code quality. This is especially important when considering the Android lifecycle, as this can lead to unexpected application behaviour if handled incorrectly, possibly leading to stress and confusion for PwD. Unit tests were implemented to validate the application’s responses to lifecycle events, ensuring robust error handling and contribution to a smoother user experience. We also implemented tests for the PyB (Appendix C) to ensure each method produced a correct result for accurate and useful insights.

Integration testing was conducted to assess the interactions between multiple components and verify their behaviour in combination. For the mobile application, this was done by checking responses to user interactions to verify each component works together without error, and helped detect any critical issues with the design, such as unexpected errors and crashes. The method of interaction chosen was a standard simulated user testing alongside a monkey test, where rapid, random inputs are simulated using an emulator to try and trigger unexpected behaviour over time. These tests revealed any prominent issues, allowing us to fix them before user testing.

For the PyB component, integration testing focused on verifying the correctness produced insights. This was done using Postman to send simulated diary entries (Figure 13.1) designed to produce insights of every type. The results were then compared to expected outputs to validate the system’s accuracy. This method worked well as we could pinpoint which systems were not working correctly by looking at each type of insight in turn, narrowing down where any errors lied.

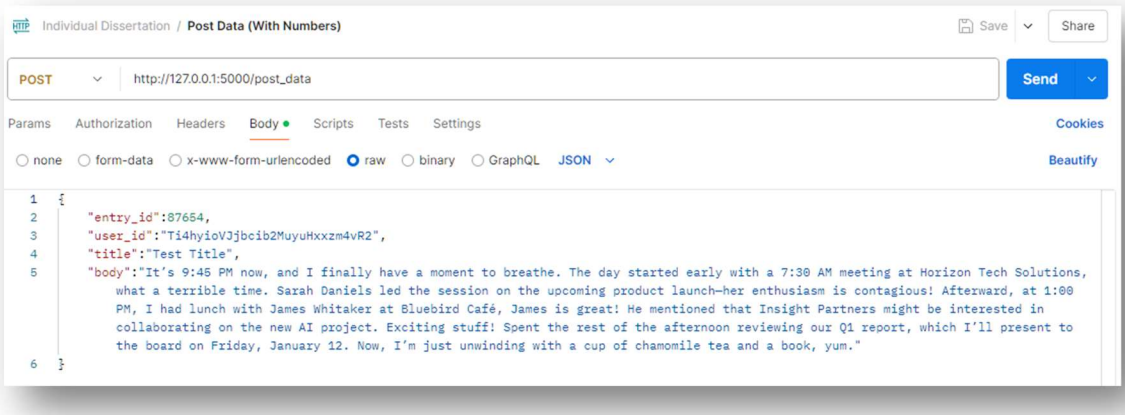

Figure 13.1: Sample test entry for PyB using Postman.

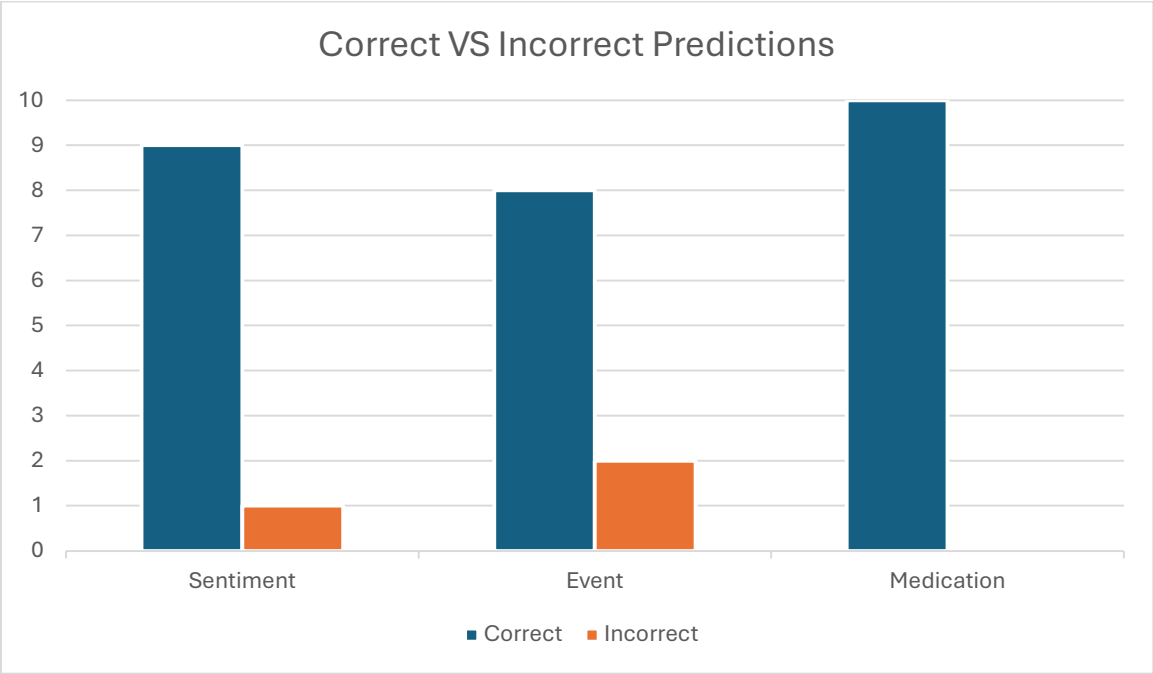

Figure 13.2: Chart showing results for Postman testing.

13.1.2 Patient and Public Involvement

It was ultimately decided that it would be unethical to use PwD for user testing due to their vulnerability, and we did not want to cause any undue harm by subjecting them to a testing procedure using unfamiliar technology. As such, another approach was taken; We attended a Patient and Public Involvement (PPI) event hosted by the University of Nottingham Medical School (February 2025). During the event, the project was presented to a group of ~20 people in the field of dementia, ranging from carers to medication professionals, demonstrating the project and how it could be beneficial. This session concluded with a Q&A, where attendees shared their thoughts, providing valuable feedback.

The intention for the session was to gauge for interest in the project, and to test for perceived usefulness in the real word. Our findings indicated a general concern for the usability with PwD, in particular with progressed symptoms such as loss of fine motor skills, preventing the use of small touchscreens. They suggested that MiRo would play an important role in user interactions, and in some cases will be the only possible interaction point for the system. This reinforces the need for a social robot aspect as an important accessibility tool, providing care further into the decline of cognitive function.

After conversation with industry professionals N.Chadborn and A.Moemeni of the University of Nottingham, it was decided that the system is more appropriate for people with early onset dementia, especially when introduced shortly after diagnosis, promoting it as a learned experience. Early introduction of this technology is important as individuals with dementia often experience rapid decline in their ability to learn and retain unfamiliar tasks. Studies have shown (De Wit *et al.*, 2021) previously learned behaviours have a reduced chance of being forgotten as the condition progresses.

Guests also mentioned usefulness of the reminder features, highlighting that PwD can suffer from anxiety when trying to remember important events, and a feature to keep them updated would be highly beneficial. This was considered when designing the reminders, opting for a sequential reminder system, where only the next reminder in the sequence is shown. This static reminder will stay in the user's notification bar as a constant reminder of the next task they have due. Showing only one reminder at a time was intentional as to reduce the anxiety with having to manage multiple events at once, giving sole focus to the next event.

13.1.3 Usability Testing

Usability testing for the mobile application was done in two different ways.

Firstly, we asked a group of 5 volunteers to independently use the application for a period of up to 30 minutes. We then asked these individuals to fill out an anonymous survey (Appendix B) to gather their feedback from their time spent with the application. Survey questions mainly used a scale-based system of 1-5 (Figure 13.3), also including text-based responses for qualitative data. Using the results from this survey, we can gather the strongest opinions about the application as follows:

1. All 5 respondents found the application easy to use and navigate, all giving a score of above 4. This means it is highly likely that the application will be appropriate for older users and PwD in early stages of their condition, as the application is designed specifically for this.
2. Most respondents found the generated insights useful, 4/5 giving a score of 5. This shows us that the types of insights that are generated are relevant and useful in daily organisation. One respondent gave a score of 2, and when asked to elaborate said that they found the medication insights overwhelming as they contained a lot of information about each medication, without being specifically targeted to that individual user. From this we can gather that users may desire a more tailored approach to insights, perhaps by building a profile on each user and using that to determine which details to show, especially for medication insights.
3. One user commented that they would find the application useful day-to-day, without having dementia themselves, showing that perhaps there is a wider audience to the application, or alternatively that the application is not specialised enough for PwD. Further user testing is needed to confirm this.

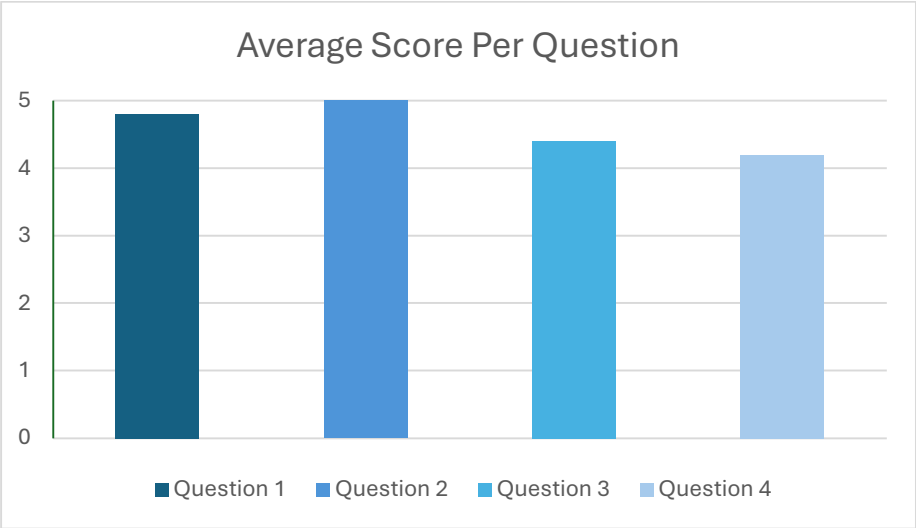

Figure 13.3: Graph showing average user score per question.

Secondly, we conducted a one-to-one interview with a user, asking them to narrate their use of the application while they explored its features. We then noted down the responses and carried out a follow up interview after the test was over. The following summarises the responses:

- The user found it easy to navigate the application using the standard Android back button, but commented that an “integrated back button” in each screen may make it easier to control.
- The user mentioned that they liked the loading icons while fetching data, as it showed them that “something is happening in the background”.
- The user commented that they liked the pop-up notifications while using the application but said that sometimes they “stayed on the screen for too long”, blocking other controls.
- The user liked how quickly insights appeared in the insight page, however noticed that they misspelt the name of a medication, and it was then not registered by the system. This suggests a form of spell checking is needed to improve medication detection.

- The user was able to quickly connect to another account using the admin system and a provided code, and liked the ability to view their insights, commenting that it may be nice to add other features such as location tracking, or integrating heart rate monitoring using "some sort of smart device".

The overall takeaway from the second study is that the application is generally easy to use, and feature rich, however there is still room for improvement and additional features that may expand its usefulness in the future.

13.1.4 Classifier Testing

When using NLP classifiers, it is essential to evaluate performance and effectiveness. The following details the ways in which this was measured.

1. *Execution Time* – Used to test for responsiveness of classifier predictions, improving corpus tuning by optimising the number of data points, as well as tracking performance of different classifier models.
2. *Accuracy* – The simplest measure of performance, comparing the number of correct predictions against the total number of predictions. This metric gives a general idea of how a model performs overall.

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

3. *Precision* – Precision is a measure of how accurate positive predictions are. It may be useful to think of it as the following: "*Of all the instances I predicted as positive, how many were actually positive*". Precision is useful for minimising false positive values and gives us an idea of how likely a model is to be correct.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

4. *Recall* – Recall measures how easily a model can find all the actual positive instances. It differs slightly from precision as follows: "*Of all the actual positives, how many did I correctly predict*". This allows us to measure false negatives and can be used to improve the chance of capturing all positive instances.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

5. *F1 Score* – The mean of precision and recall, a more robust measurement of performance than accuracy as it handles dataset imbalance. This is used as another general performance metric but is also useful to monitor how balanced a dataset is. If the accuracy and F1 score are similar values, it strongly suggests a balanced dataset.

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

6. *Confusion Matrix* – A simple table showing the number of correct and incorrect predictions for each classification of data, and what they were incorrectly predicted as. This is a fundamental tool in NLP testing and helps identify where a model is weakest.

Following this testing, the model was improved over time to create a more robust and accurate model. This can clearly be seen in the following figures, with a Naïve Bayes Multiple Word classifier eventually being chosen for this application.

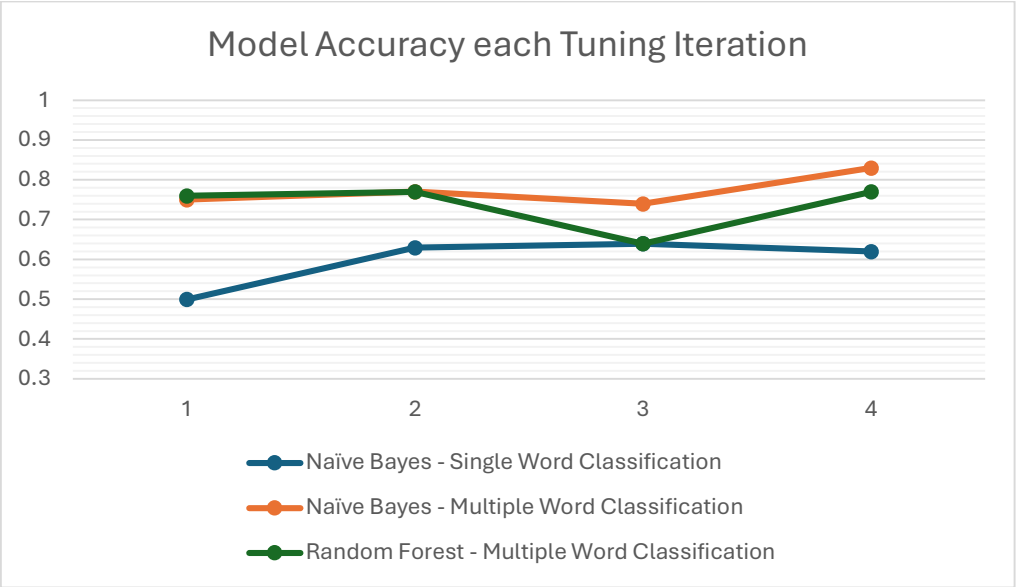

Figure 13.4: Graph Showing Sentiment Model Accuracy Over Time.

We can look at Figure 13.4 which shows the three models that were tested for this project, and their accuracy over four iterations of training. The figure tells us that the best performing model is Naïve Bayes (NB) Multi-Word classification. This is expected, as context is important for analysis and we can see a direct comparison of the performance of single word classification, which is significantly outperformed. The figure also shows a comparison of accuracy for a Random Forest (RF) Multi-Word classifier. In the first instance, RF outperformed NB and continues to be similar over each iteration. However, if we look at Figure 13.5, it becomes clear why RF was not chosen as the final classifier, RF taking an immense amount of time to make a prediction as compared to NB. This invalidates RF as an option, as we need quick predictions for a more responsive user experience.

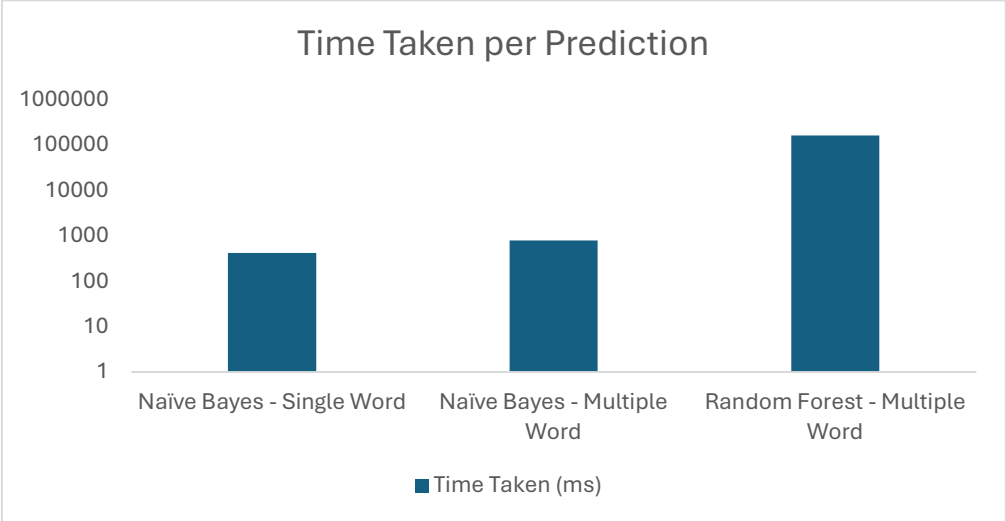

Figure 13.5: Graph Showing Time Taken per Prediction.

We tested performance using a 75/25 train-test split. We used the testing data and compared the results of predictions made with this data, to those made on the original training data. Using this method, we got an initial accuracy of ~76%. This tells us that the initial corpus is effective at predicting sentiment from a text, however there is still margin for improvement. Looking at the classification report (Figure 13.6) we can see that the weakest area of performance is in neutral, ‘neu’, predictions. If we look at the middle row of the

confusion matrix, we can see an that most of the issues lie with mistaking positive and negative statements as neutral statements, suggesting that the corpus is weaker in this specific area.

Train Test Split: 75/25				
Total Corpus Size: 43083				
Overall Accuracy: 0.76492				
F1 Score: 0.75917				
Confusion Matrix:				
2847	617	149		
566	2127	443		
219	538	3265		
Classification Report:				
	Precision	*Recall*	*F1-Score*	*Support*
Negative	0.78	0.79	0.79	3613
Neutral	0.65	0.68	0.66	3136
Positive	0.85	0.81	0.83	4022
Accuracy	0.76		10771	
Macro Average	0.76		10771	
Weighted Average	0.77		10771	

Figure 13.6: Initial Testing Report.

Using this new information, we adjusted the corpus and repeated the process over ~10 iterations. Most notably by removing stop words and using more strict formatting. By doing this, we reduced the amount of noise in the dataset by ruling out many of the datapoints that may be less relevant. The final resulting dataset now has ~27,000 datapoints, and running the tests again we see an improvement to ~83% accuracy (~92% for positive sentiment) and improvements in performance in all three sentiment types, but most notably a large improvement for neutral predictions. We can also see that the F1 score and Accuracy are similar values, indicating a balanced dataset.

To see the details of the implementation of this testing, see Implementation section 12.1.24.

Train Test Split: 75/25				
Total Corpus Size: 27481				
Overall Accuracy: 0.83393				
F1 Score: 0.83267				
Confusion Matrix:				
1530	376	39		
180	2476	118		
150	272	1724		
Classification Report:				
	Precision	*Recall*	*F1-Score*	*Support*
Negative	0.82	0.79	0.80	1945
Neutral	0.79	0.89	0.84	2780
Positive	0.92	0.80	0.86	2146
Accuracy	0.83		6871	
Macro Average	0.84		6871	
Weighted Average	0.84		6871	

Figure 13.7: Adjusted Corpus Testing Report.

14 Summary and Reflection

14.1.1 Project Management

This dissertation was undertaken using a SCRUM Agile approach, splitting tasks into individual sprints of two weeks in length. This was managed using Trello to track tasks depending on level of completion. The planned structure was to build a secure base of research and methodology before developing and implementing the final product in the latter half of academic year. The sprint system was beneficial as it allowed for adaptivity over time. This was especially useful due to the complexity of the system, involving many parts.

Complications with the mobile application resulted in a complete rework of the storage structure, delaying work on the MiRo robot for multiple sprint cycles, meaning that steps had to be taken to reduce the workload required from MiRo, reducing scope and axing plans for additional functionality.

Another setback was a failure of the university network during the exam period, resulting in a major exam being pushed back three weeks. This meant time that had previously been allocated to this project was then needed to prepare for an exam resit. This impacted MiRo development and app finalisation, so work was initially slow in this area. Finally, a passing in the family in early March further delayed work on the project, which was later accounted for by reallocating resources from other modules.

The initial Gantt Chart can be seen in Figure 14.1, followed by a secondary Gantt Chart in Figure 14.2 more closely representing the work we carried out. From this we can see more time was needed for application development, it becoming more time consuming than initially planned, while elements like user testing needed to be brought forward to allow for more stringent testing. Despite this, the project and its management were a success, and all the targets were met.

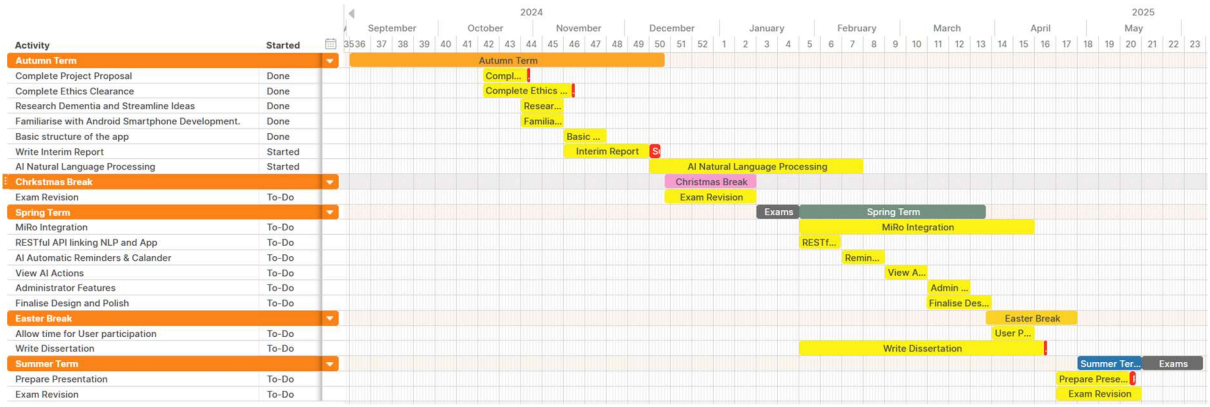

Figure 14.1: Initial Gantt Chart.

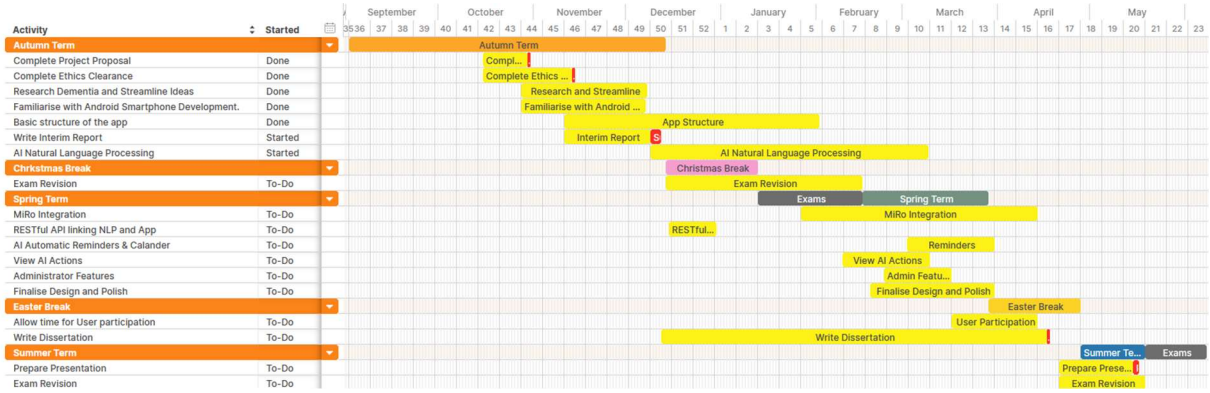

Figure 14.2: Final Gantt Chart.

14.1.2 Conclusion

Bringing this project to its natural conclusion, dementia brings with it challenges for not only those diagnosed with the condition, but those who care for them. While treatments for dementia exist, most are outdated and inaccessible to regular people. This project solves this by innovating and building on existing ideas, introducing a natural language powered system combined with a social robot to improve the quality of life for PwD while easing the burden of care on loved ones with a secure, unintrusive monitoring system.

We began by investigating existing solutions to dementia care, pinpointing the strengths and weaknesses apparent in each and using these to construct an idea for a more effective method of care. By focusing our research on proven solutions, we were able to take iterative steps of improvement to build a more cohesive and useful final product on a solid ground of existing works.

Using this new knowledge, we built the mobile application following strict design principles to ensure accessibility for those suffering with disabilities parallel to dementia, successfully creating an effective user interface. This app includes features to enter and edit diary entries, which are stored online using an implementation Firestore for multi-device compatibility. A bespoke, queue-based notification system was also introduced, guided by feedback from PPI, to minimise cognitive strain when managing multiple reminders at once.

At the same time, we developed an effective AI using various NLP technologies, most prominently Naïve Bayes Classification and NER, to detect insight from a user entry, demonstrating the effectiveness of these systems when working together with diary entries. Once an initial classifier was implemented, we then began the lengthy process of iteratively improving its performance using various methods of tuning to optimise for accuracy across each type of prediction, eventually landing on a respectable final accuracy of 83%.

Following from this, work began to produce a system for the MiRo robot, prioritising user experience with human-like responses such as blinking and nodding, while listening to a user's voice input. Using GCST, we successfully implemented a system allowing the user to transcribe a diary entry to MiRo, which is saved automatically to Firestore. This is an important step for accessibility, improving interaction with the system by providing companionship, whilst removing the need to use a small touch screen for most interactions.

Finally, to verify the usefulness and working operation of the system, we conducted comprehensive testing of the system in its entirety, using a variety of methods to ensure optimal performance and usability. Testing revealed success across the board, with all 5 test participants enjoying the ease of use provided by the app, and unit and integration testing showing the system working fully as expected.

Overall, this dissertation presents a promising step forward in digital dementia care, integrating NLP and robotic interaction into a cohesive, user-focused mobile application to improve quality-of-life. The sections that follow explore areas for future development, reflect on the personal and academic growth achieved throughout this project, and outline the specific contributions made to the field.

14.1.3 Future Directions

This project has the potential for a variety of future improvements, and acts as a foundational basis for further innovation.

One idea, reinforced PPI feedback, is to improve the interactivity with MiRo, adding features to dynamically communicate back to the user for improved social interaction and companionship, perhaps by using a large language model and an AI text-to-voice model, as well as using MiRo's touch sensors for more dynamic responses.

Similarly, pairing the application to other smart devices (wearables, cameras, etc) to collect data and automatically set reminders could provide a central monitoring hub for carers and PwD. An interesting idea would be to use date detection in a smart fridge, sending notifications for when food is about to go out of date to prevent PwD accidentally consuming gone-off produce.

Further development for classification tasks could also be carried out, such as feature engineering and deep learning. These would dramatically improve the accuracy of the model and open new avenues into the types of insight we can produce.

Finally, during PPI, one attendee, a consultant in neurorehabilitation, mentioned the potential use of the project in rehabilitation for people with brain trauma, particularly with memory loss and regaining daily structure. Areas like this could be explored to expand the usefulness of the application to help more people in daily organisation tasks and potentially with rehabilitation.

14.1.4 Personal Reflection

This section provides a short personal reflection from the perspective of the author.

On reflection, I am proud to have successfully developed a comprehensive system to improve the quality-of-life for PwD. I have thoroughly enjoyed working on this project and have seen growth in both technical and soft skills, while my ability to manage time and work independently has matured significantly. I have also demonstrated my ability to produce maintainable code, seen in the quality of this work.

I was also able to effectively convert many setbacks into opportunities, demonstrating flexibility and resilience. Most notably, adapting to difficult ethical conditions for user testing, leading me to a change in testing methodology, instead leaning into application usability testing for a general audience as well as attending a PPI session in February, where I independently held a presentation to ~20 attendees. This experience not only provided valuable, real-world feedback on the system but also greatly boosted my confidence in public engagement and communication.

I am also exceptionally proud to have been accepted for an abstract presentation at the Alzheimer's Association International Conference (AAIC) in July, where I will be able to share this work with others in the field to gain new ideas for where to take the project next.

14.1.5 Contributions

This project offers several improvements and adaptations to existing systems, while introducing the new concept of using NLP in combination with social robotics as a method of improving the quality-of-life for PwD.

We have shown that NLP can be used in practical scenarios to help organise daily activities by plotting events automatically and extracting insight from journal entries, effectively adapting this existing form of dementia care into an AI driven tool. This is combined and improved with the use of a social robot, aiding in accessibility, and providing companionship in the form of a physical robot with human like responses.

By working alongside people with real world experience, we have provided research into the usefulness of mobile applications for PwD, concluding that they may be impractical for older PwD and those with progressed symptoms, instead offering the idea of introducing them earlier as a learned experience, such as in cases of early onset dementia.

Additionally, we have submitted an abstract detailing the project to the AAIC, which has since been accepted and approved for a poster presentation in July 2025. By presenting this idea at the largest international conference on dementia research, we hope to gain further insight into how it may be adapted and improved, as well as opening a dialog with people in related fields as to how our idea could potentially be transferred to other projects, with the goal of producing a more effective solution.

15 Bibliography

Awada, I.A. *et al.* (2018) 'Adaptive User Interface for Healthcare Application for People with Dementia', in *2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5. Available at: https://doi.org/10.1109/ROEDUNET.2018.8514150.

Belcic, I. and Stryker, C. (2024) *What Is Supervised Learning? | IBM*. Available at: https://www.ibm.com/think/topics/supervised-learning (Accessed: 11 February 2025).

Bird, S., Loper, E. and Klien, E. (2009) *Natural Language Processing with Python*. O'Reilly Media Inc. Available at: https://www.nltk.org/ (Accessed: 9 February 2025).

Carros, F. *et al.* (2020) 'Exploring Human-Robot Interaction with the Elderly: Results from a Ten-Week Case Study in a Care Home', in. New York, NY, USA: ACM, pp. 275, 1–275, 12. Available at: https://doi.org/10.1145/3313831.3376402.

De Wit, L. *et al.* (2021) 'Procedural Learning in Individuals with Amnestic Mild Cognitive Impairment and Alzheimer's Dementia: a Systematic Review and Meta-analysis', *Neuropsychology Review*, 31(1), pp. 103–114. Available at: https://doi.org/10.1007/s11065-020-09449-1.

Faisal, M. *et al.* (2024) 'Robot-based solution for helping Alzheimer patients', *SLAS Technology*, 29(3), p. 100140. Available at: https://doi.org/10.1016/j.slast.2024.100140.

Google LLC (2024) *Speech-to-Text AI: speech recognition and transcription*, *Google Cloud*. Available at: https://cloud.google.com/speech-to-text (Accessed: 10 March 2025).

Hagstrom, C., Heppner, A.G. and Kunde, K.N. (2021) 'The Effects of Socialization on the Progression of Dementia'. Available at: https://spark.bethel.edu/etd/696/.

Hardeniya, N. *et al.* (2016) *Natural Language Processing: Python and NLTK*. Packt Publishing Ltd.

Lotsa Helping Hands LLC (2025) *Care Calendar Website*, *Lotsa Helping Hands*. Available at: https://lotsahelpinghands.com/ (Accessed: 28 January 2025).

Medisafe® Medication Adherence Platform (2025) *Digital Drug Companion*, *Medisafe*. Available at: https://www.medisafe.com/digital-drug-companion/ (Accessed: 28 January 2025).

Murel, J. and Kavlakoglu, E. (2024) *What is bag of words? | IBM*, *What is Bag of Words?* Available at: https://www.ibm.com/think/topics/bag-of-words (Accessed: 11 February 2025).

Müller, C., Paluch, R. and Hasanat, A.A. (2022) 'Care: A chatbot for dementia care'. Available at: https://doi.org/10.18420/MUC2022-MCI-SRC-442.

National Institute for Health and Care Excellence (2025) *BNF content published by NICE*, *British National Formulary (BNF)*. Available at: https://bnf.nice.org.uk/ (Accessed: 4 March 2025).

Pedregosa, F. *et al.* (2011) 'Scikit-learn: Machine learning in Python', *Journal of machine learning research*, 12, pp. 2825–2830. Available at: https://doi.org/10.5555/1953048.2078195.

Petr Balog, K., Faletar, S. and Jakopec, T. (2023) 'Older Adults' Attitudes toward Digital Technology and Perceptions of Its Usefulness: Example of the City of Osijek, Croatia', *Proceedings of the Association for Information Science and Technology*, 60(1), pp. 686–690. Available at: https://doi.org/10.1002/pra2.840.

Prescott, T.J. *et al.* (2018) 'MiRo: Social Interaction and Cognition in an Animal-like Companion Robot', in. New York, NY, USA: ACM, pp. 41–41. Available at: https://doi.org/10.1145/3173386.3177844.

Quinlan, C.K. and and Taylor, T.L. (2013) 'Enhancing the production effect in memory', *Memory*, 21(8), pp. 904–915. Available at: https://doi.org/10.1080/09658211.2013.766754.

ROS: Home (no date) *ROS.org*. Available at: https://www.ros.org/ (Accessed: 5 December 2024).

Salton, G. and Buckley, C. (1988) ‘Term-weighting approaches in automatic text retrieval’, *Information Processing & Management*, 24(5), pp. 513–523. Available at: https://doi.org/10.1016/0306-4573(88)90021-0.

16 Appendices

The following section lists the relevant extra content as part of this work. Appendices can be found in the final submission under the corresponding file names.

- Appendix A: Copy of User Stories – “user_stories.pdf”
- Appendix B: Copy of a User Evaluation Survey – “user_testing.pdf”
- Appendix C: Unit Testing on Python Backend – “unit_testing.pdf”
- Appendix D: User Manual and Installation Guide – “user_manual.pdf”