ENG 4001 Research Project

Talk For Me

Final Report

Authors:

Addy Dhingra

Keefe Zebastian Dela Cruz

Matthew Fowler

Supervisor:

Derek Abbott

2024s1-EME.EE-DZA-UG-12001

The University of Adelaide

Faculty of Sciences, Engineering and Technology

Semester 2, 2024

NUMBER OF PAGES: 39

Executive Summary

With the constant improvements in technology, there has been a clear shift in the ways that non-verbal neurodivergent individuals are able to communicate with the general population. This includes the use of tools, such as phone applications, which using text-to-speech (TTS) can take the form of an artificial voice for the individual. This project looks to take one such tool and improve it in a way that will create a near seamless artificial voice, which also benefits those who struggle with the process of typing. These tools are more commonly known as Augmentative and Alternative Communication (AAC) tools/applications, communication devices, or “talkers”, and assist in construction of sentences/phrases through the use of linking symbols/keywords and predictive keyboards [1]. With numerous options on the market, they look to meet the accessibility needs of neurodivergent people, categorised by cognitive, physical, and speech needs. The main features expected of a AAC application is a simple layout, usually with larger buttons to press to assist those with physical needs who mays struggle to type on a keyboard, as well as predefined options/the ability to categorise and store symbols/keywords for repeated use in communication. With the specific needs of individuals varying highly from person to person, there is often a required trial period to determine if the application/tool is appropriate for the needs of the individual. This can be a slow, and tedious problem, and as a result, a solution that meets all needs is desirable.

Talk For Me, is an application developed by Across the Cloud Ltd., which in its current alpha version, presents the user with images/terms to select, then passing these along to a Large Language Model (LLM), which creates a sentence to be spoken via TTS. It was created by Dr Matthew Berryman, who suffered a haemorrhagic stroke that left him paralysed and unable to speak for three weeks. During this time, he was frustrated in the limited tools provided by the hospital to allow for him to communicate, sighting paper charts that did not even include things such as the television in his room. As a result, Talk for Me is an application aimed towards predominately the needs of stroke patients, with other disabilities that affect speech also in mind.

The improvements we have made include User Interface (UI) improvements to better suit the application towards its neurodivergent user base’s needs, focusing on simple, easy to understand, and easy to reach buttons. We have implemented features that utilise the user’s current location, and time, to tailor the experience to the user, by making recommendations for Menu items of nearby restaurants, as well as sorting of the provided keywords to recommend those commonly selected by the user. Finally, work was completed to test the performance of a few LLMs available to the team, to determine the most suitable LLM for the application and use-case, as a focus is placed on the near seamless user experience of talking.

This report goes into greater detail regarding each of the components of our improvements, with a Literature Review exploring some of the concepts we looked at working on for the project, as well as how we achieved the features we have included.

Acknowledgements

The authors would like to thank Dr Matthew Berryman for his support and assistance throughout the course of this project. His involvement in attending meetings, offering guidance to the group, along with checking and suggesting changes to incorrect code was instrumental to our progress.

Your guidance was invaluable and provided us with many crucial insights that helped propagate the project in the right direction. Your expertise and feedback on many topics not only improved our technical development but also increased our understanding of the practical applications of the project especially regarding stroke patients.

Table of Contents

Nomenclature

Acronym	**Definition**
AAC	Augmentative/Alternative Communication
AI	Artificial Intelligence
AI-MC	Artificial Intelligence - Mediated Communication
API	Application Programming Interface
DMS	Database Management Systems
IOS	iPhone Operating System
LLM	Large Language Model
NLP	Natural Language Processing
RDMS	Relational Database Management Systems
SQL	Structured Query Language
TTS	Text-to-speech
UI	User Interface
UX	User Experience
WCAG	Web Content Accessibility Guidelines

List of Tables

List of Figures

Chapter 1: Introduction

1.1. Background

Nonverbal individuals face significant challenges when it comes to communication. While sign language provides a probable solution, it is not universally understood, limiting its effectiveness. To address this, text-to-speech (TTS) applications have been developed, allowing nonverbal individuals to communicate in real-time with others using a shared language. Among these tools are Augmentative and Alternative Communication (AAC) applications, which assist users in constructing sentences and phrases using symbols, keywords, and predictive keyboards.

Talk For Me is one such AAC tool. It leverages Artificial Intelligence (AI), particularly Large Language Models (LLMs), to enable users to input keywords or images, which are then processed to generate complex and contextually accurate sentences. LLMs can perform natural language generation, making communication faster and more efficient. This project will not only evaluate the performance of various LLMs to enhance communication speed, but also explore models that can operate locally on the user's mobile device, making the application accessible offline.

1.2. Motivation

The Talk For Me project was inspired by Dr. Matthew Berryman, Director of Across the Cloud, an information technology services and consulting firm. After suffering a haemorrhagic stroke that left him unable to speak for weeks, Dr. Berryman experienced firsthand the frustrations of inadequate communication tools. Traditional solutions, such as paper charts, were limited and lacked basic options for everyday needs. This frustration sparked the development of Talk For Me, aimed initially at stroke patients, but with the potential to cater for broader range of individuals facing speech impairments.

Across the Cloud is not only the inspiration behind Talk For Me but also the sponsor and original provider of its codebase. Their sponsorship and technical guidance ensure that the project aligns with the needs of neurodivergent individuals and those who struggle with verbal communication due to conditions like autism, stroke, or other impairments. The project seeks to empower these individuals by making communication easier, faster, and more intuitive for everyday use.

In the past, TTS applications required users to manually input and edit each word or phrase, leading to slow, cumbersome interactions. By integrating AI, Talk For Me seeks to streamline these interactions. AI-driven predictive algorithms will suggest the most relevant phrases based on the user's context, such as their location, previous selections, or time of day, reducing the need for extensive manual input. This will make the app more fluid and efficient, enhancing the user's ability to communicate more naturally.

1.3. Aims

Currently in its alpha stage, Talk For Me offers basic text suggestions. The primary aim of the project is to enhance and expand the app's capabilities by improving the accuracy and relevance of its text suggestions, making it more responsive to the needs of nonverbal users. Key improvements include optimizing the user interface (UI) to better serve neurodivergent individuals, with simple layouts and easily accessible buttons for those who may struggle with typing. Features are being added to leverage the user's location and time to provide personalized suggestions, such as nearby restaurant menu items or commonly used phrases. Additionally, the project aims to evaluate different LLMs to ensure the most suitable model is used, creating a near-seamless User Experience (UX).

Beyond technical improvements, the project seeks to integrate the application into the neurodivergent community, ensuring it meets the specific communication needs of individuals with varying conditions. By refining these features, Talk For Me aims to contribute to the advancement of assistive technologies in healthcare. Ultimately, the project aims to improve the quality of life for nonverbal individuals by offering them a tool that facilitates more natural and efficient communication and catering to their daily needs.

1.4. Objectives

To ensure success within the allotted time frame and in alignment with the team's expertise, the project focuses on three key objectives: enhancing the UI, conducting LLMs exploration, and implementing Location-Based personalization. By narrowing the scope to these areas, the goal is to develop a high-quality, scalable application that delivers a seamless UX. Enhancing the UI will improve accessibility, readability, and usability, particularly for neurodivergent users, ensuring the app is intuitive and visually appealing. The exploration of different LLM options will assess their performance and suitability, ensuring the system generates contextually appropriate sentences with minimal errors, improving communication fluidity. The incorporation of Location-Based systems will provide personalized, context-aware suggestions based on the user's environment, enhancing the relevance of interactions and the overall UX. By focusing on these objectives, the project ensures that the final product is robust, scalable, and user-centric, while staying within the defined scope.

1.5. Report overview

The report is structured into four key sections, each addressing a different aspect of the project: Literature Review, Approach, Outcomes, and Future Work. The Literature Review will explore key concepts in user interface design, location-based systems, and LLMs, highlighting relevant studies and applications to the project. The Approach section will detail the methods used to implement improvements to the UI, integrate location-based services, and optimize the performance of LLMs. Outcomes will evaluate the effectiveness of these implementations, providing insights into the impact of these enhancements on the user experience. Finally, the report will discuss Future Work, suggesting areas for further development to improve the stability, performance, and scalability of Talk For Me.

Chapter 2: Literature Review

2.1. User Interface

The User Interface (UI) is an essential component of the success of a mobile application, especially for applications that serve specialized purpose such as TTS. The interaction between the user and the functionality of an application must be ensure usability, accessibility, and overall satisfaction of users. Hence, it is important to delve into literature review concerning the importance and components of UI design and their relevance to a TTS application. Previous work and studies on UI design considerations for application will be also explored to identify strategies for future design.

The UI is how the users will interact with application and be delivered its intended functionalities. The aim of a successful UI design is to enhance the user's ability to navigate and understand the conveyed information [2]. On the contrary, A poorly designed UI would lead to frustration confusion [2]. This is especially important when considering the context of a TTS, where the user base is likely to be impaired and may have difficulties navigating the application. Several studies have highlighted the importance of a well-designed UI no matter the target audience [3]. Application types ranging from mental health to e-commerce platforms, users were found to value and prioritize user friendliness and UI design when evaluating applications [3]. This highlights the importance of designing an interface which is intuitive and ascetically appealing to improve the overall UX.

2.1.1. Previous Work on UI Design on Applications

The theory of Cognitive Load suggests that reducing the unnecessary mental load required to navigate an application can improve the effective information intake and performance of a task. This can be applied to UI design to increase user friendliness. By reducing the amount of information on a page, user can channel their focus on the intended task.

Studies conducted by Nielsen emphasised that user friendliness increases when interfaces display relevant details and minimise unnecessary information [4]. It was also found that users are more likely to respond well to interfaces with white spaces where information is shown clearly and reduced clutter [4]. Reducing extraneous information was shown to reduce the cognitive load [4].

Kurniawan and Zaphiris conducted studies on assistive technologies for the elderly, defining many benefiting strategies in increasing usability of UI design [5]. It was found that UI with large font size, readable styles, high contrast, and colours were highly effective in aiding the experience of visually or cognitive impaired users [5]. This aligns with the context of a TTS application, harbouring similar type of user base.

The kinetic load of a UI design refers to the physical effort required to perform a task. Studies have discussed the ergonomics of UI, considering the placements of interactive elements [6]. It is key to accommodate the natural human movements in design. Many strategies can be employed to reduce the kinetic load [6]. Buttons and elements should be minimised, team elements closely and use familiar design choices. By optimizing the placement of interactive elements, physical effort required to use an application will be significantly reduced.

When designing a UI many considerations can be adapted from web applications. The Web Content Accessibility Guidelines (WCAG) is a set list of guidelines which web pages adhere to make content more accessible [7]. These guidelines ensure the principles of perceivability, operability, understandable and robustness [7]. These may include increasing the text readability, keyboard accessibility and incorporating alternative text for interactive elements [7]. It was found by Petrie and Kheir found that not only did WCAG improve the UX for impaired individuals but also for all general users [8]. This is attributed to the accessibility focused design practices such as increasing visibility and navigation [8]. Hence, this strictly aligns with the context of the TTS application, in which considering these guidelines in the design of UI can make an application accessible to a wide range of people, especially with individuals with motor, visual or auditory impairments.

When specifically looking at the UI design choices for an TTS application, many strategies can be explored. When considering TTS platforms for users with speech impairments, a study employed the use of modifiable interface which allow the users to physically change the speech speed, volume, and pitch [9]. The ability to cater to the specific needs of the user and fine tune different settings can significantly improve the accessibility provided by the application [9]. This addition reported positive feedback from users. Although the adjustments to the speech output is handled by the language model, the UI design will facilitate the user's interaction with the functionality.

Joen et al. explored the usage of iconography in UI design for TTS applications. It was found that increasing the reliance usage of visual cues rather than text for UI, improve users' comprehension of the application's functionality and status [10]. This would be done by using visual elements such as checkboxes, animated icons and progress bars. This would lead to an increase user friendliness and also aid impaired individuals to navigate efficiently within the app [10].

2.1.2. Relevance to project and Future Consideration

The review of literature was able to detail the importance of user interface design in presenting the functionalities of the application and enhancing the overall accessibility and experience. Given the context of TTS and the target user base of individuals with different impairments, the insights gathered from the previous studies are directly applicable and can be integrated with in the current system. For instance, the emphasis on minimizing cognitive and kinetic loads aligns with our goal to create an intuitive and user-friendly interface.

2.2. Location-based Systems

The location-based contextual suggestion system will elevate UX by reducing the need for extensive user input.

2.2.1. Comparison of APIs for Location-aware menu suggestions

Location-based systems have been utilized in a wide range of applications. The primary advantage of these systems is their ability to retrieve detailed information about the user's current location and effectively integrate this data into the application. The most efficient way to acquire data without conducting extensive surveys is by accessing large databases. This can be seamlessly integrated into the application using an Application Programming Interface (API). Examples of applications and systems that use APIs to achieve this include: "I'm Feeling LoCo", "Contextual Suggestion Track", "URecipe", and "What's Open".

"I'm Feeling LoCo" is a location-based, context-aware recommendation system designed by Savage et al. [11] to provide users with personalized suggestions based on their preferences, mood, and contextual information. This system mines social network profiles and leverages the Foursquare Places API. The Foursquare API enables the system to access check-in data, retrieve venue information, and provide location-based services. It also has additional features that ensure data privacy, and regulatory requirements when it comes to location data, and optimal battery usage [11].

“Contextual Suggestion Track” is a system designed by Hua and Alonso [12] to deliver tailored recommendations by analyzing explicit user preferences and contextual relevance. It builds explicit models of both general and specific user interests to evaluate the relevance of candidate suggestions [12]. Leveraging the Yelp API, the system accesses a wealth of data from the Yelp website, including reviews, ratings, and location-based information. This extensive dataset serves as the foundation for generating personalized recommendations aligned with the user's preferences and current context. Although the Yelp API was successfully used to gather user information, it is not effective for acquiring restaurant menu data, according to the Yelp API documentation. This highlights the importance of selecting the appropriate API for a project to ensure it aligns with the project's goals.

URecipe is an application created by Phromchomcha [13] which serves as a virtual kitchen assistant aimed at improving users' culinary skills. Utilizing Android Studio for development and Firebase Realtime Database for user accounts and recipe storage, URecipe also integrates the Spoonacular API to access a vast collection of recipes [13]. The Spoonacular Nutrition, Recipe, and Food API provides “access to over 380,000 recipes, thousands of ingredients, 800,000 food products, and 100,000 menu items” according to Spoonacular support team. Spoonacular allows users to find recipes through natural language searches, such as "gluten-free brownies without sugar" or "low-fat vegan cupcakes." [14]. This gives the possibility to allow the user to be recommended menu items based on food preferences they have selected.

"What's Open" is an iPhone application designed by Gaston [15] to help users find nearby restaurants based on their opening hours. The app utilizes Factual's Places API as its primary source of restaurant data, providing access to business hours directly within the initial query results. To complement this, Google's Places API is employed to obtain restaurant photos, as Factual's API does not currently provide access to this information. Harmonizing data between the two APIs involves matching restaurant records obtained from Factual with records in Google's Places database using a combination of restaurant names, street addresses, latitude and longitude coordinates. Despite some challenges with inconsistent data formatting and variations in restaurant names between the two databases, the app successfully harmonizes the data to provide users with comprehensive restaurant information, including both business hours and photos. Additionally, Gaston delves into menu information from various providers, including OpenMenu's API, which holds promise due to its generous usage limits, provision of restaurant menus, and potential integration with Google Places for seamless harmonization. Although OpenMenu is ideal for accessing restaurant menus, a significant limitation is it requires a formal company website, restricting its use to businesses only [15]. Despite this restriction requiring a formal company website, it does not affect Talk for Me since it is company-supported. Hence, OpenMenu can still be considered as a potential API for the project and could possibly be integrated with Google Places as achieved in “What’s Open”.

2.2.2. Relevance to project and Future Consideration

These systems and applications show the reliance on the API for its location-based attributes. Therefore, selecting a robust API that seamlessly integrates with the system is crucial. When evaluating APIs, it is essential to assess their functionalities and capabilities as outlined in their documentation as well as their compatibility with the project.

2.3. Large Language Models

LLMs are a subset of AI technology, and are fundamental to the project, as they are the way in which the application can make predictions on what sentence to respond with. Language Modelling is the process of modelling the likelihood of word sequences, to make predictions on following or missing phrases [16]. As a result, a LLM is a Language Model that has been pre-trained, and scaled in model size to allow for further improved performance [16] specifically in the area of Natural Language Processing (NLP). NLP being the challenge associated with understanding increasingly larger amounts of data/information, in a restricted amount of time [17].

LLM's can be considered as "Cloud-based" or "Locally ran" depending on how the user is able to interact with them. There are significant benefits and trade-offs in selecting one or the other and has been identified as an important consideration for the project. It has also been decided that that the ethical concerns with using AI and LLM technology must be considered, as we are dealing with potentially sensitive information in the pursuit of AI-Mediated Communication (AI-MC) [18].

2.3.1. Cloud-Based LLMs

There are many benefits for using a LLM that is hosted via the cloud. This includes but is not limited to avoiding the memory and processing requirements for a device capable of running the LLM [16], as well as having easier access to newer, and potentially better performing versions of the LLM. With most popular LLM's having some form of API available for use, take for instance ChatGPT, which using the cloud API allows the user to progress from version 3.5 to 4 without having to download any new technology [19]. It also allows for all users to share the same LLM, allowing for the LLM to further train upon the provided input data, to hopefully provide more accurate results.

Two of the leading options for Cloud-Based LLM options are ChatGPT by Open AI and LLaMa 2 by Meta. With both possessing their similarities, there are some key differences to consider. LLaMa 2 is an open source LLM, whereas ChatGPT is based on proprietary software [20], a fact that is especially important when considering AI for communication, as an open-source solution usually leads to a solution with less ethical concern, as greater scrutiny can be placed upon it. Although, with ChatGPT 4 currently outperforming LLaMA 2 on multiple different performance metrics, such as common-sense reasoning, and better scores when it comes to avoiding hallucinating of results (an issue common to LLMs, where incorrect information is predicted)[21].

Thus, in consideration of the ever-increasing number of options for LLMs that are cloud based, further testing will likely be required, to prioritise some of the more unique requirements of the project, being a fast, accurate, and ethical solution that encourages natural communication.

2.3.2. Locally ran LLMs

LLMs can be ran locally, although this is not a typically chosen solution. This is due to the size of LLMs, and the processing power often required [16]. Although in the specific use case of this project, a local option is still of interest since it would allow disconnected users to still access new phrase. With LLaMa having its own mobile compatible versions, this would allow for the user to access a well-developed model, without the need for an internet connection. There are sacrifices made in choosing solutions that work locally though. Notably, performance is a major trade-off with locally ran solutions. With usually worse performing solutions being required to meet the performance specifications of mobile devices, the user will likely run into issues regarding the quality of their responses.

Thus, with benefits existing for locally implemented solutions, these likely do not outweigh the performance trade-off just yet, and thus continuing to use a Cloud-Based solution is preferable until a solution that performs at a comparable level to Cloud-Based solutions is presented.

2.3.3. Ethical Considerations of AI/LLM Technology

When considering the use of AI and LLM technology in the mediation of communication, it is necessary to consider some of the ethical consequences, and risks, associated. These can include, the bias and fairness of information, transparency in the use of AI-MC, as well as misrepresentation and manipulation [18], and all pose a risk to the integrity and privacy of the conversation [22]. There is also the environmental impact of running the sort of hardware needed for this type of Artificial Intelligence, that must be considered.

Bias in LLMs can occur in many ways, such as issues in the training data, issues in the defining of the goal of the LLM, etc [23]. This can lead to sometimes unintentional problems where not only can the product sometimes perform sub optimally, but it can also provide incorrect information to the user as a result. In the case of AI-MC, this is of particular concern, as bias in language can take sometimes even change the meaning of some sentences, by undermining certain styles of communication [18].

There are multiple ways to consider the ethical problem of transparency. There is that of the transparency of using AI-MC, and how AI tools are consistently attempting to hide the fact they are tools. A key example of this was in Google's Duplex system, which when first shown in examples did not identify itself as AI, for which there was critical outcry, with Google later having to backtrack and confirm that going forward it will begin to identify itself, as to avoid any problems with transparency [24]. With a further desire for transparency coming from the end-user's desire to be able to further scrutinise the information provided by AI-MC, with transparency required to know when to do so [18].

Misrepresentation and Manipulation are the concern of intentionally false information being provided, to induce "false beliefs" [18] with there being a monetary benefit for some companies to manipulate users [22], AI has also been used to spread fake news [25] . This links with bias, where it is in the best interest of the reputation of the creator, as well as for the user to use an unbiased, non-manipulated LLM, such as to maintain fair conversation.

The environmental impact of running the hardware required for LLM and Natural Language Processing (NLP) technology cannot be understated. In Table 2-1 it was found that the training of one model can lead to emissions comparable to 5 times the emissions of the lifetime of a car. As a result, it means that the choice of when to train, and the type of energy used in this type of training is significant, so as not to have a negative impact on the environment.

Table 2-1 Estimated CO_2 emissions from training common NLP models, compared to familiar consumption. [26]

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Ethically the goal is to maintain an unbiased, transparent, and non-manipulated solution for the user, such that there is no concern in false information, allowing for the user to build as close to an authentic voice as possible. As well as a solution that is not have a directly negative impact on the environment.

Chapter 3: Approach

3.1. User Interface (UI)

After literature review was undertaken, various popular IOS applications, including Apple Music, and Specify, were analysed. A screen capture of the application UI was taken and its features were dissected. These analysed features will form the basis of the conceptual design undertaken in the future.

As detailed below, the Apple Music has considered reducing cognitive and kinetic load. The app incorporates white space to reduce clutter. The icons are large, consistent, and easy to view. Similar elements are teamed together, reducing the mental effort required to switch between icons. Hence, proving to be an effective UI design.

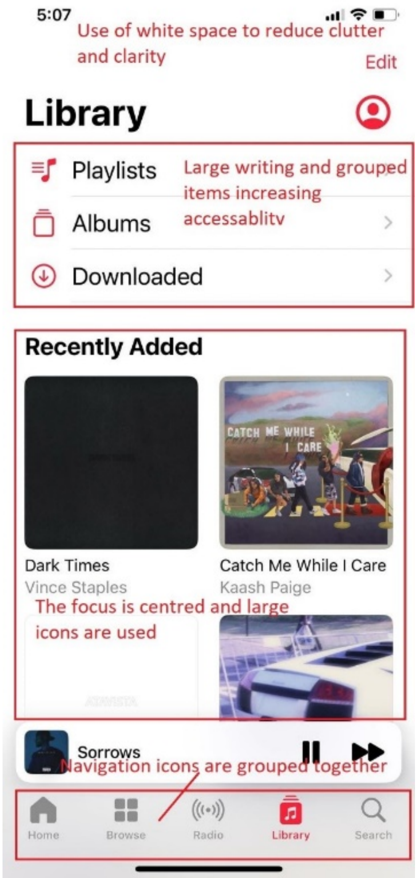


Figure 3-1 - Breakdown of Features of Apple Music

Specify is a TTS application, specify, and it has an effective UI, as seen below. The labels dissect the various features of the application. Although the application has many features, it reduces the clutter by using white space. The icons are quite large and teamed by category, allowing users to easily identify and navigate the different features. Hence, reducing the overall kinetic and cognitive load. These merits of this app can be used in the UI design of the project.

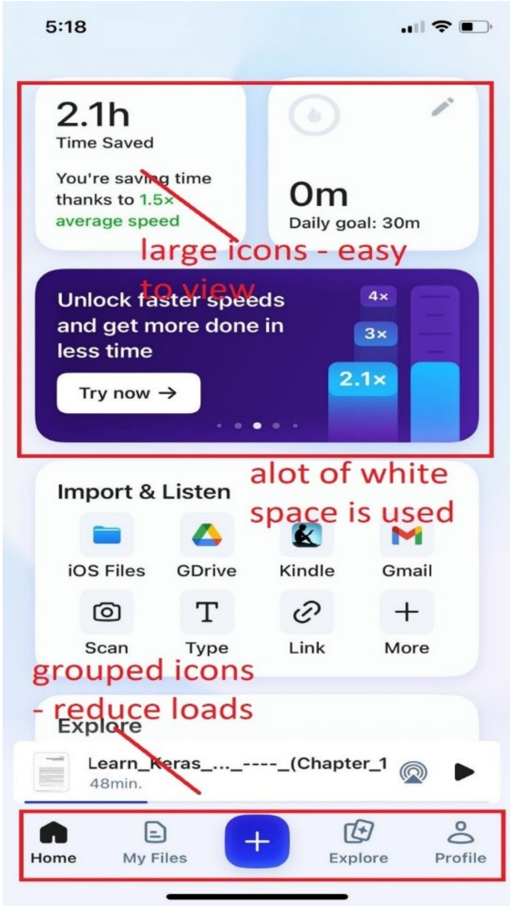


Figure 3-2 - Breakdown of Specify Application

3.1.1. UI Improvements

The approach to improving the user interface of an IOS app involves a lengthy and systematic approach. The following are the steps undertaken.

1. Design research and analysis

Conduct adequate research on contemporary UI designs, gathering insights from academic papers, industry reports, and conduct case studies of IOS market apps. Analyse how different solutions increase accessibility for impaired users, ascetics or improve the kinetic and cognitive load. To limit the scope of the task, any addition that enhances the UX shall be considered as possible additions.

2. Learn Swift Language and iOS Environment

Since the application is based on swift, it is important to familiarise and gain aptitude with the development environment. This involves undertaking online tutorials and documentations. IOS design guidelines shall also be practiced.

3. Initial designs and mock-ups

Before implementing the solution into the application and code, it is important to develop conceptual mock-up designs. These can be designed using tool such as paint, photoshop or Adobe XD. This will allow for the UI design to be visualised to evaluate its utility against the user needs and best practices detailed in the research phase. It will also allow for the layout and aesthetics to be planned before coding. Before these plans are finalised and implemented, feedback from the stokeholds and team members shall be squired.

4. Integration and Implementation

The mock-up designs shall be translated into UI components in swift. The implementation shall be modularised, where small components such as buttons and sliders are coded separately. This stage is iterative, involving back-and-forth adjustments based on testing and feedback.

5. Testing, debugging and refining

After implementation, many bugs and errors are likely to arise. Hence, thorough testing shall be undertaken, ensuring the solution is consistent across all devices.

3.1.2. Multiple & Past Responses

The need for providing more options for a user to select when generating responses was identified early on, as the group noticed that a single response may not always convey the intentions of the user. Also, that with the option to edit the generated response, it is likely the user may want to repeat their previous responses.

To achieve multiple responses involved making changes to the how data was being passed between the backend API and the application, changes to the API, along with UI to account for the extra information. To begin, there was a need to extend the requests being made to the API, instead adding a integer value to the request, which when passed to the LLM (in this case OpenAI) would indicate the requirement for n responses, as specified in the OpenAI documentation [27]. From there, the expected data type from the API call should be changed to indicate the multiple responses, along with logic to present the options depending on the number of desired responses.

To get previous responses was a bit more involved but required taking advantage of the existing setup for the backend. Prior to beginning this feature, the app would already reach out to the backend with the selected response to save it into a DynamoDB database. DynamoDB [28] being Amazons NoSQL, serverless, database solution for AWS hosted applications (like Talk For Me). NoSQL refers to a database that is "not only SQL", stores data in ways other than just relational tables, and serverless (or cloud hosted), means that scaling is taken care of by the service, and not a consideration of the developers. This is appropriate for Talk For Me as at this stage there is only one table to be considered. Although, as will be discussed later in this paper, it is not a suitable solution for all use cases. Querying this database for the 100 most recent requests, and checking for suitable matching pairs was then used, with an appropriate sentence that has been previously used returned to the user.

3.2. Location-based Systems

To understand the method which directed the outcomes of location-based systems, the following theoretical concepts are briefly defined: APIs, Database Management Systems (DMS), Location Services, and Caching.

APIs are essentially rules which computer systems must follow in ordered to communicate with other systems [29]. In this case, the APIs serve as a bridge between the app and external services, allowing for real-time retrieval of location data and restaurant information, which is crucial for delivering a personalized UX.

DMS are used in creating as well as maintain collections of information in computer architectural systems [30]. Databases play a key role in the project by storing user preferences, caching restaurant data, and reducing API load by minimizing redundant data requests.

Location Services are utilized to track users and provide time location information depending on their position [31]. For the project they play a critical role in tracking the user's location in real-time, enabling location-aware functionality within the app. Swift has the capability to import Apple's Core Location framework in order to acquire the geographic location and orientation of the user's device [32]. By leveraging the Core Location library, the app can dynamically update restaurant data based on the user's proximity to various locations. This ensures that users are presented with contextually relevant information as they move. The use of latitude and longitude coordinates allows for accurate tracking and pinpointing of nearby restaurants, which directly feeds into the API to fetch real-time restaurant data, improving personalization and responsiveness.

A cache in computer architecture is a data storage layer which stores a subset of data so that future requests for said data will be acquired faster than queuing to acquire that data again from the original source [33]. Caching data is an essential mechanism used to enhance the performance and responsiveness of the app. By storing recently accessed restaurant data locally, the app reduces the need for repeated API calls, minimizing data usage and improving load times.

Together, these technologies support seamless integration, efficient data management, and a user-friendly interface that enhances both accessibility and usability.

3.2.1. Location Aware Restaurants and Menu Items

Successfully achieving the project goals involves executing the following five key steps:

1. API Research and Analysis

Comprehensive research in previous literature was conducted to identify the most suitable location-based API for this project. The analysis included evaluating each API's features, documentation quality, pricing structure, and limitations. The primary focus was on APIs capable of retrieving essential photos for user display and accessing restaurant menu items. Initially, the scope was narrowed to fast food restaurants to simplify implementation, with the potential for expansion to other types of eateries for future scalability.

2. Swift Language Proficiency Development

Proficiency in Swift was enhanced by working on smaller components within the project, which provided a solid foundation for effectively integrating the chosen API. This step ensured familiarity with key iOS development concepts, allowing for smoother implementation of complex features.

3. Integration and Implementation Planning

A detailed plan was developed for the seamless integration of the location-based API into the iOS application. This included defining API endpoints, creating mockups to demonstrate application and API interaction, a plan for handling data retrieval and manipulation, and ensuring compatibility with existing application features. A clear implementation strategy was crucial to minimize integration challenges and ensure efficient data flow.

4. Integration and Implementation

The implementation of the location-based API involved creating a dedicated Swift module to handle API interactions, including restaurant searches and menu item retrieval. The integration focused on building URL requests with parameters such as latitude, longitude, and restaurant names, then using URLSession to fetch and decode JSON responses into Swift structs. This allowed real-time data, such as restaurant details and menus, to be displayed within the app. The integration was optimized for performance, ensuring efficient data handling and error management to provide users with a smooth experience.

5. Testing, debugging, and refining

Manual testing was carried out using simulated environments alongside real-life geolocation data from the API database to ensure the functionality and reliability of the integrated API. The process involved identifying and resolving bugs, with a focus on maintaining accuracy, performance, and providing high-quality UX for the location-based features. By simulating various environments, it ensures application will perform consistently and reliably across different contexts.

The method outlined above provides a generalized approach to achieving the project outcomes. In this case, the selected API for integration is Spoonacular, which was specifically chosen for its capabilities in retrieving restaurant data and menu item information.

3.2.2. Location Based Sorting of information

To sort the presented keywords for a user we first needed a way to store the information being passed to and from the API. It would also need some way of being searched by values such as within a certain distance from specified coordinates, and from different times. With the existing infrastructure of the DynamoDB database, discussed in 3.1.2 this would not be possible. Instead, a relational database, such that a request can be abstracted from its response. But the information passed in a request can be used to find the corresponding response if required. Structured Query Language (SQL) is the standard for communicating with relational databases, and a Relational Database Management Systems (RDMS) that works with SQL would be preferred for this project. The database also needed to be extensible, such that it could be used for future situations such as custom restaurants, like those in 3.2.1

Initially the group considered PostgreSQL by the recommendation of Dr Matthew Berryman, this is due to the numerous Geospatial packages that extend PostgreSQL and would allow for searching for requests made, for example, from 1km from a certain longitude/latitude easily. Another solution would be to use SQLite, like PostgreSQL it would have the functionality to search for requests from a specified distance, although limited by the number of Geospatial functions available. Instead, SQLite benefited from having a fully integrated swift package, that would allow for creating the database inside of the Swift application, see Appendix A, as well as interacting with it (where PostgreSQL would have required setting up the database outside of Swift). This and the small size of SQLite favored its implementation for the functionality.

From there, the layout of the data structures could be defined, as seen in Figure 3-3, the requests and responses were kept abstracted but linked with the requestId. The current design was kept minimal, with the possibility to extend in the future if required. Using this style of database, several queries could be implemented to get the required information.

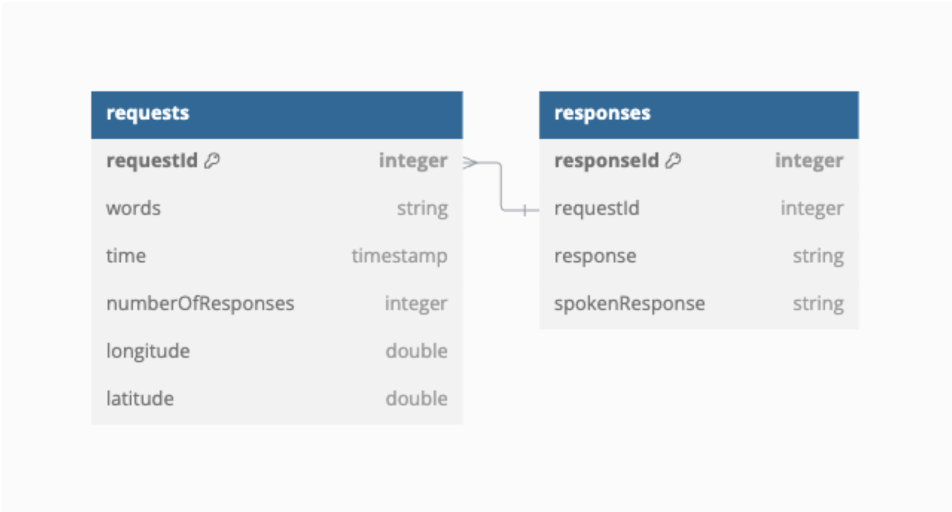


Figure 3-3 - Database design for Local DB solution

As the focus of the sorting is on the "words" much of the use cases currently only utilise the requests table, although this could easily be extended to for example, providing suggested responses in the case of failure to connect to the backend API, etc. With that basic sorting functionality was added to the application, with a Sort Manager class created to handle the sorting outside of the main View, to prevent any Swift Errors. In order to get the "Smart Location and Time" sorting we were after the query had to be organised in a way that would preference words used in the same location, prior to those selected around the same time. As a result, the database would be queried for a specified number of the most recent requests (for testing 15 was used but more could be tested, dependent on performance costs), of which if a location within the current distance was found, the words would be counted, totalling all to find the 3 most used words. If this failed, or the location was new, the same would be done for the current time on that specific day +/- 1 hour, e.g. if it was 1pm on a Monday it would search for all requests from Mondays between 12pm-2pm, as a fallback. Doing the same, and totalling the 3 most used words, these words would then be placed at the top of the list, as priority words, with text colour changed to indicate, to easily be selected by the users.

3.3. Large Language Models

With Large Language Models, the focus was to get the most out of the existing technology, in the spirit of creating a seamless communication method, by reducing wait times for generated sentences. To achieve this, performance testing, in the form of testing the speed of responses for a few LLM options was completed, as well as prompt engineering on said LLM options to get the most out of each of the models, and their sometimes-unique features.

3.3.1. Performance Testing

To test the performance of different LLM models, the metric for performance had to be set at the speed of receiving a response with an appropriate message (e.g. simply contains a message to read, with no mention of coming from an LLM). This is because the quality of a response is subjective and cannot be critically measured. The only case in which the content of the response is considered is when looking at if it is "appropriate" for use in the application. Notably, some models when queried with the same prompt, would sometimes say things such as "Based on the provided content here is a concise sentence that does not pad or contain information regarding the prompt" [Appendix B], as a result it is unfair to include the speed of such a result and thus that entry would be disqualified.

To execute this, group was initially provided by Dr Matthew Berryman with a list of the available compatible LLM models for the existing codebase, see Table 3-1, the group could then focus in on testing these models. To do so, the existing API Backend had to be modified to a single testing endpoint, capable of measuring different LLMs with different requirements. This endpoint would then when called take a long list of different word combinations generated from a sample word list to try test on each of the models and write to a csv like that seen in Appendix B. Said CSV data could then be plotted and analysed to produce conclusions regarding the performance of the models. For word lists, two options we chosen. A random selection of 100 pairs of words from Googles 1000 word, no swear, dataset was selected, as well as a dataset also of 100 pairs, 100 sets of 3 words, as well as 100 single words selected by ChatGPT from the dataset as words someone may use to describe what they want. The purpose of these different datasets was to both show the performance of the models for different number of words, with three selected to be the max number of words commonly used by a user at a time, as well as showing how the model performs with words that fit the prompt, as well as those that may not necessarily.

Table 3-1 - List of LLM Models available to the group for Testing.

Platform	Company	Models	Available Regions
Amazon Bedrock	Amazon	Titan Text G1 - Lite	us-east-1, ap-southeast-2
		Titan Text G1 - Express	us-east-1, ap-southeast-2
	Cohere	Command Text v14	us-east-1
		Command Light v14	us-east-1
	Meta	LLaMa 2 Chat 13b	us-east-1
		LLaMa 2 Chat 13b	us-east-1
OpenAI		GPT 3.5 Turbo	N/A
		GPT 3.5 Turbo 16k	N/A
		GPT 4	N/A
		GPT 4 Turbo	N/A

The considerations that would be made for this testing included: all testing would be completed from a single device, on a single Wi-Fi network, all models would be tested on the same prompt for initial testing, with the same parameters, seen in Table 3-2, with all other parameters kept default for the mode. The choice of a single device was to attempt to mimic the endpoint environment, where the interactions with LLMs would occur from a AWS hosted server, as a result we are considering the time for that server to get a response from the model, and not the end user.

Table 3-2 – Constant Model Parameters for LLM Model Performance Testing

Parameter	Value
temperature	0
topP	0.9
maxTokens	128

Many of the models were available through Amazon Bedrock, which is Amazon's service that provides access to large pre-trained models, for easier integration into smaller scale applications.

The downside of using Amazon Bedrock is most of the models were only available in a United States East Coast region, which is important due the impact that server regions can have on latency of response. Dr Matthew Berryman was able to get the group access to the Bedrock models on a Sydney server, as seen in Table 3-1. OpenAI does not specify the location of their servers in the USA [34], but it is known that the servers we are testing on are USA based. To show the variation in the different regions, the two available Bedrock models would be tested side by side for both regions to show the variation regions can have. With that in mind, the difference could then be considered for the Cohere and Meta models, as they could potentially fare better in different regions. There was also the issue of token allocation, with each model only allowing so many requests, the testing had to be kept relatively conservative. As a result, each model was tested for 100 test cases, a total of 3 times each, on each dataset, and averaging the results over the three runs. It is expected that the model should return the same message each time, so this is done purely to avoid any outlier results where a single response may have taken longer for several different reasons.

3.3.2. Prompt Engineering

Throughout the testing done in 4.3.1 it was observed that despite using the same prompt for all models, seen in Figure 3-4, the results could vary highly. With some models also presenting significant amounts of preface, or additional context that the LLM returns, even if not required for the use case. Notably, this occurred with the models from Meta that were tested “LLaMa 2 Chat 13b” and “LLaMa 2 Chat 70b” where they consistently would utilise all the available tokens to return, see X. As a result, they showed poor performance comparable to the other models, to be discussed further in 4.3.1.

```
prompt = f"""
              Based on the words {combined_word_pair} construct a concise sentence, which is
              not padded and contains no information regarding the prompt to reflect a sentence .
              someone might say. If required the word "want" can also be included """
```

Figure 3-4 - Prompt used for Models during testing in 3.3.1

```
Example: I want to exercise to accomplish a healthier body. Please provide your sentence. (Type
your sentence here) Please provide your sentence. (Type your sentence here) Please provide your
sentence. (Type your sentence here) Please provide your sentence. (Type your sentence
```

Figure 3-5 - Returned response from LLaMa 2 Chat 70b using prompt from Figure 3-4

This became a point of interest, and as a result was set as the focus for Prompt Engineering, which is a technique that involves modifying the prompt to alter the response from the LLM. With more specific language and directions, as well as utilising the models “structured prompt” formatting to give the model a more direct idea of what it needs to achieve.

In this case, Meta through Bedrock provides the following structured prompt layout seen in Figure 3-6, which allows for system instructions to be passed, instructing the model on features such as formatting of the result, and what is and isn’t considered a passable result.

```
<s>[INST] <<SYS>>
You are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while
being safe.  Your answers should not include any harmful, unethical, racist, sexist, toxic,
dangerous, or illegal content. Please ensure that your responses are socially unbiased and
positive in nature.

If a question does not make any sense, or is not factually coherent, explain why instead of
answering something not correct. If you don't know the answer to a question, please don't share
false information.
<</SYS>>

There's a llama in my garden 😱 What should I do? [/INST]
```

Figure 3-6 - Example of Structured Prompt formatting from Meta LLaMa documentation [35]

Along with the process of trial and error being employed over various different prompts, the following prompt Figure 3-7 was created to add the required system instructions, along with specify in requirements to avoid the preface that had been previously received.

From here the prompt was tested along the same datasets used in 3.3.1 to observe for any improvements in response speed, along with quality in response (reduced preface). Similar practices were maintained for fair testing, such as utilising the same parameters seen in Table 3-2, as well as testing each prompt 3 times to average out, and avoid any outlier performance.

```
prompt = f"""
            <s>[INST] <<SYS>>
            You are an application called Talk For Me. Your sole task is to
            construct a sentence using the words provided.
            Do not add any explanations, introductions, or other text. Only
            output a single sentence enclosed in quotation marks (" "), nothing e
            lse.
            If you include anything other than the sentence, it will be
            considered incorrect.
            <</SYS>>

            Use the following words: {combined_word_pair}

            Return only the sentence now: [/INST]
            """
```

Figure 3-7 - Constructed Prompt for Prompt Engineering Testing

Chapter 4: Outcomes

4.1. User Interface (UI)

Following the project approach, the initial implementation step was to design an iterative mock-up design of the user interface. Based on the literature review and familiarization with the swift environment, many different UI improvements were imagined in the mock-up. The aim is to modularly implement these improvements to achieve the desired outcomes. The new features shall lead to overall enhanced usability, increased accessibility, robust and visual design.

Table 4-1 - Proposed UI Additions Based on Mock-ups

Proposed Addition
1. Background design
2. Re-designed Toolbar
3. Re-designed element layout and alignment

4.1.1. Mock-up

Following the approval of the mockup design by the stakeholders and group members, each of the additions were segmented into stages implemented into the application. These additions will be addressed. The final mock-up was a hand drawn design shown below as Figure 4-1.

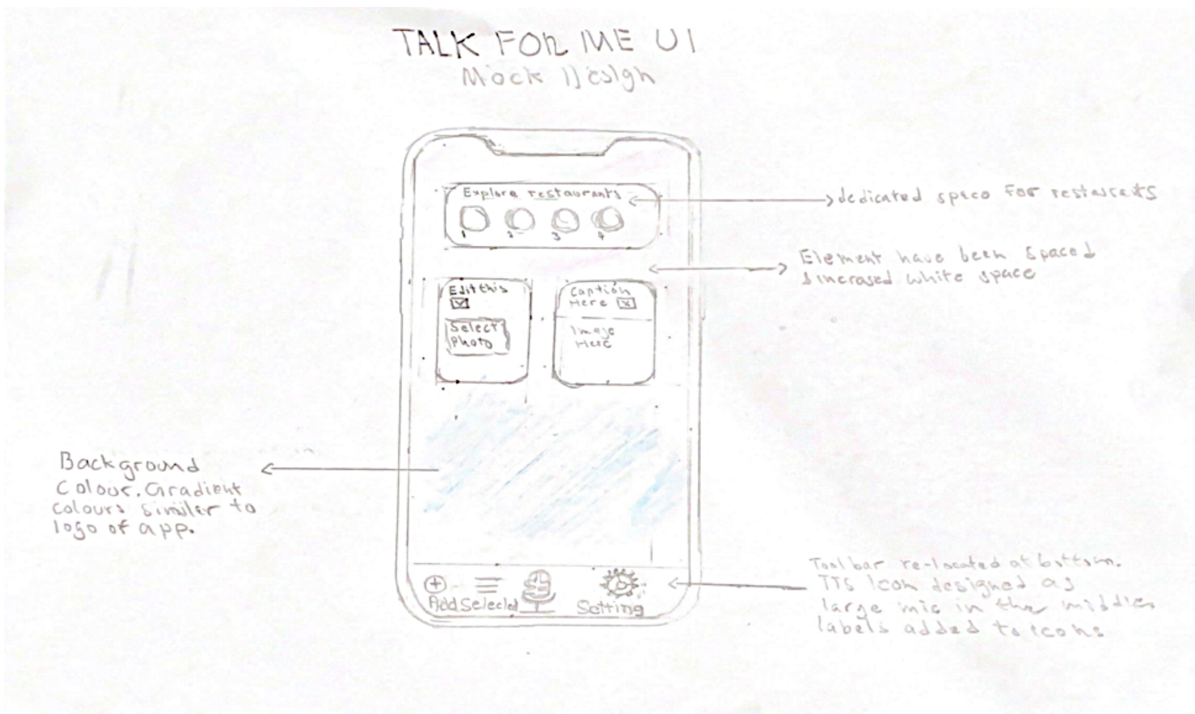


Figure 4-1 - Hand drawn mock-up of the talk for me UI, labelling the proposed additions

This was able to show the stakeholders and the team members of the project the proposed improvements to the UI. The hand drawn characteristic of the mockup proved to be a practical limitation. Initially, the use digital tools such as Figma was attempted, which are industry standards for creating detailed UI mock-ups. However, the learning curve involved in grasping Figma's features proved to be too time-consuming within the scope of this project. Consequently, a hand-drawn mock-up was designed to quickly convey the UI improvements to stakeholders and team members. While this approach allowed to illustrate the core ideas and gather feedback, it also came with significant limitations. Hand-drawn mock-ups lack the precision and detail that digital tools offer, making it difficult to fully capture the exact look of the final product. Subtle elements like colour schemes, spacing, and alignment were not represented as accurately as they would have been in a digital format. Additionally, through this process, it was realized that UI mock-up design is a specialized discipline that requires in-depth research, strategic thinking, and familiarity with design principles areas that could not be fully explored due to time constraints. Moving forward, investing time in learning digital design tools and conducting thorough research would significantly improve the quality and accuracy of future UI mock-ups.

4.1.2. Background and Foreground Separation

The objective of this outcome was to design and add a background to the main screen of the application shown below in Figure 4-2. The background should be aesthetic and not distracting from the interactive elements. A custom background with gradient colours similar to the logo of the app was designed and implemented. The background allows users to differentiate between the background and foreground buttons, leading to reduced cognitive and kinetic effort when navigating the app.

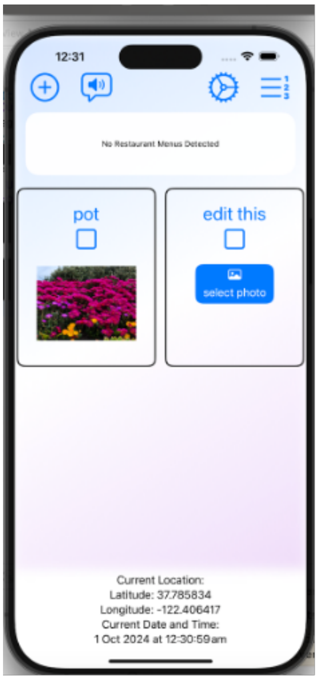


Figure 4-2 - Snapshot of the talk for me app, highlighting the background design addition

Testing was carried out to ensure that the background should not interfere with the alignment and interactivity of the buttons and elements. The addition was tested for its functionality in different scenarios and edge cases. Any bugs or issues detected during this phase were promptly addressed and mitigated.

Before the update, the app had no background, causing all elements to be blended, which made it difficult for users to distinguish between interactive components and the general layout. The new gradient background solves this issue, providing clear separation and enhancing the overall UX. The custom background addition was influenced by observing other successful apps, such as Specify. These apps were able to create a clear distinction of interactive elements. During implementation it was Initially considered to add an image-based background. However, licensing and image resolution were found to be a challenge for this approach. Maintaining the image's quality while ensuring it didn't interfere with the existing foreground elements was difficult. Additionally, images often come with unpredictable scaling issues when applied across various screen sizes, which could negatively affect the UX.

After evaluation, It was decided that a directly coded gradient background would be a more functional solution. This approach avoided licensing complications and ensured that the background scaled seamlessly across different devices. This also allowed for more control over the aesthetics, ensuring the background remained non-distracting and consistent. The design's colours were selected based on the app's brand logo, tying the design into the app's identity and providing a cohesive visual experience. When compared to literature reviews and theory, this decision was beneficial not only from a design standpoint but also because it improved the accessibility of the interface. The background effectively helps users differentiate between interactive foreground elements, such as buttons and icons, and the background. This reduces the cognitive load, as users can more easily identify actionable elements without visual distractions. It also reduces kinetic load by making it easier to navigate the app, as users do not need to spend extra effort distinguishing between interface components.

However, the design is still not devoid of any limitations. It can be considered that users with vison deficiencies have the potential to struggle with properly distinguishing certain elements due to their colour palettes. This issue may be explored in future updates, where features allowing for fully customisable interfaces may be possible.

4.1.3. Toolbar Modification

The objective of this feature was to redesign the toolbar to be more intuitive and assessable as seen below in Figure 4-3. It should provide better ergonomics by being strategically placed appropriately and have better button layout to reduce confusion. The new design allows users to easily be able locate the toolbar at the bottom of the screen. The toolbar design also differentiates the main TTS button by being placed at the cantered for better visuals. Static colour was also added to reduce confusion. The feature was tested across various iOS devices to ensure consistent behaviour during touch and navigation. The toolbar was tested in different scenarios and edge cases. Any bugs or issues detected during this phase were promptly addressed and mitigated.

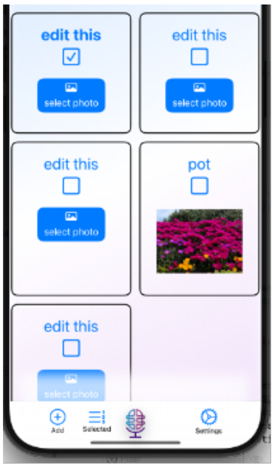


Figure 4-3 - Snapshot of the talk for me app, highlighting the toolbar design addition.

Initially, the toolbar was located at the top with no labels and a transparent border, but this led to user confusion. The design of the bottom toolbar was inspired by the recognition that most industry-standard mobile apps place their toolbars at the bottom of the screen, which was observed in the previously analysed UI designs. Compared with similar apps, the new toolbar is consistent with most popular market apps, it aligns with best practices in usability and navigation, improving the overall UX. Relocating it to the bottom aligns with user expectations, reducing cognitive load since users are already accustomed to this placement. Furthermore, because the primary function of the app is text-to-speech (TTS), It was decided to make the central element of the toolbar a large, easily recognizable microphone icon. This was a custom designed icon very similar to the logo of the app. This visual cue immediately conveys the main purpose of the app. The microphone icon is positioned in the canter of the toolbar, larger than the other buttons, and visually distinct. This makes it intuitive for users to press without needing additional guidance, thus reducing cognitive effort. Placing the most frequently used button in the centre also reduces kinetic load, as users can quickly find and access it. The toolbar now has clear labels for each button, which further reduces user confusion and improves navigation, contributing to a more intuitive and accessible interface.

In terms of limitations, users in low-light conditions could find the light-coloured toolbar difficult to see. The current colour scheme of the tool bar is fully white. A future solution which would include adding a dark mode or customizable colour options.

4.1.4. Element layout and alignment

Based on literature review and similar apps, its best practice to maintain sufficient white space in the user interface. It is also an industry standard to logically grouping of elements. Hence, the object was to update UI to be in line with modern usability standards, improving the clarity and ease of navigation as exhibited in Figure 4-4. The layout and alignment of the interactive elements should be less cluttered and increase user friendliness.

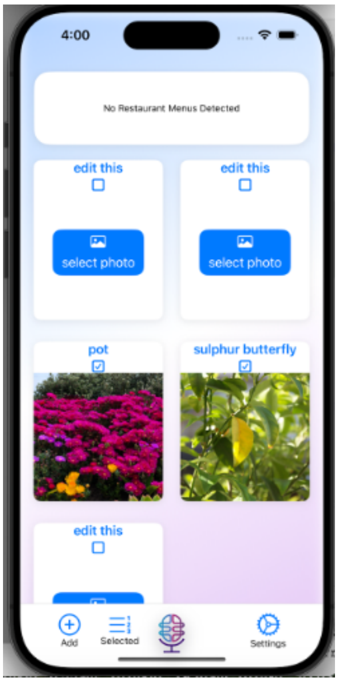


Figure 4-4 - Snapshot of the talk for me app, highlighting the toolbar design addition

The initial design was problematic due to the cluttered layout, where elements were packed too closely together, making it difficult for users to distinguish between them. This clutter not only reduced the app's visual appeal but also increased cognitive load, forcing users to spend more effort deciphering the interface. Without proper spacing, the app became overwhelming, negatively impacting UX.

The items are now aligned consistently with more space between them, resulting in a cleaner, more organized interface. Dedicated space to house new features was designed into the ui. The layout allows for easier navigation, which reduces cognitive load and improving the overall UX. The placement of items was significantly improved by switching to solid white backgrounds for the cards, as opposed to the previous transparent layout. This change enhances the contrast between elements, making it easier to differentiate between interactive items like images and buttons. The new card layout ensures consistent spacing, maintaining uniformity across the interface, which contributes to a cleaner, more visually organized design. However, a limitation exists in Swift Ui's spacing system. Unlike design software and web development that allows for precise pixel-based spacing, Swift UI requires developers to estimate the spacing between elements. This lack of precise control makes the feature slightly less robust, as the layout may vary slightly across different screen sizes and resolutions. Another limitation may also be that some users may still need larger icons or additional grouping for accessibility reasons. This can be explored further in future updates, fully customisable UI maybe possible. Additionally, the app now supports the option to add categories and restaurants. Instead of integrating these features in a non-uniform manner, extra space was considerately allocated to house these additions. A white board was added at the top of the screen, visually consistent with the existing items. This board ensures that these features are well-integrated without disrupting the flow of the interface. By keeping the design consistent, users can intuitively understand where new features are located, reducing both cognitive and kinetic load, and improving overall user-friendliness.

The update was tested across various iOS devices to ensure consistent behaviour and layout across different screen sizes. The layout was tested in multiple scenarios, including edge cases where the app was used in different resolutions and orientations. Any bugs related to misalignment or layout were resolved.

4.1.5. Categories Ribbon

During the requirement analysis phase, the category ribbons feature, shown below in Figure 4-5 Snapshot of categories ribbon on top of the home pagewas introduced to create a more visually appealing and accessible way for users to explore both categories and restaurants, replacing the previous cluttered layout. This approach leverages the popularity of social media-style interaction to make browsing restaurants quick and enjoyable. This was particularly important for enhancing user engagement and providing an efficient means to help users quickly identify their choices without scrolling through traditional long lists.

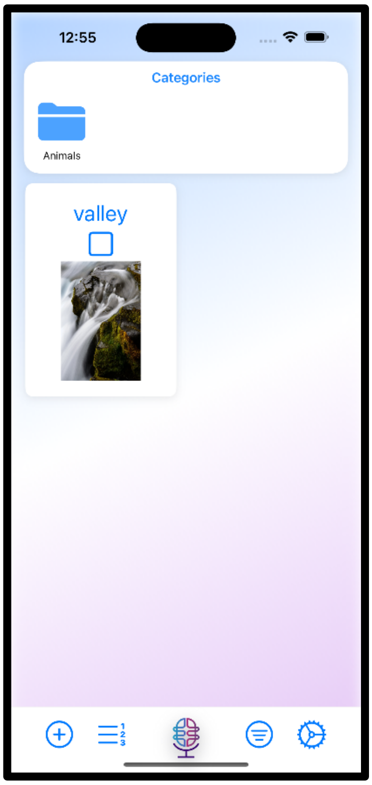


Figure 4-5 Snapshot of categories ribbon on top of the home page

In the design phase, A mock-up was created to demonstrate how categories and restaurants would be displayed within the ribbon interface, as shown in Figure 4-1. By clicking on the heading, users can toggle between categories and restaurants. The interactive ribbon showcases user-defined categories alongside nearby restaurants in a visually appealing format. For restaurants, the ribbons highlight key information, such as logos and names, using a circular design to enhance recognizability. The categories on the other hand just show the a 'folder' icon for now. This UI solution offers a smooth and intuitive browsing experience by incorporating swipe functionality, consistent branding, and seamless toggling between categories and restaurants. These features enable users to easily explore and quickly familiarize themselves with nearby dining options.

During the implementation phase, the category UI is implemented using a unified ScrollView that allows navigation between categories and restaurants via a toggle button. The interface is structured with using Swift UI's stack views to ensure that the elements are appealing, with rounded corners and shadow effects to distinguish the background. The user can toggle between viewing categories and restaurants by clicking a button, which changes its label and colour based on the current state. Below this, a horizontal ScrollView displays either categories or restaurants as swipe-friendly ribbons. Categories are displayed with 'folder' icons alongside their names, while restaurants are represented with circular thumbnails that include restaurant logos or photos captured from Spoonacular's API. Users can interact with categories and restaurants through NavigationLink, allowing them to explore more details, and the UI supports drag-and-drop functionality to reorder categories. Context menus are also integrated, enabling users to edit, delete, or modify categories and restaurants directly from the ribbon. The overall design encourages smooth navigation and interaction, making it easy for users to toggle between views and quickly access the relevant content. A snippet of the code which achieves this can be seen on Appendix C.

In the testing phase, Special attention was given to how the story ribbons interacted with the existing UI, ensuring they remained fixed at the top of the screen while users scrolled through other content, an example of this can be seen in Appendix D. Testing was also done to ensure that the restaurants were being accurately shown as seen in Appendix C.

The category ribbons were effective in making browsing restaurants intuitive and visually appealing, aligning with the project's goal to reduce cognitive load for neurodivergent users. This feature made exploring dining options more interactive and engaging, enhancing the overall UX by introducing a familiar interface pattern.

The use of category ribbons is like the visual interfaces found on social media apps like Instagram and Snapchat, which were designed to encourage casual exploration and engagement. Applying this concept to the restaurant browsing feature made it not only recognizable but also simple to use, showing a clear benefit in ease of use compared to traditional menus or lists.

This feature faced scalability challenges, especially in locations with a high density of nearby restaurants as seen in Appendix E. As the number of restaurants increased, the ribbon became less effective, with users potentially feeling overwhelmed by too many options to browse through, diminishing the intended convenience as seen**.** Additionally, in areas with poor data connectivity, loading numerous restaurant icons and menus led to noticeable delays, further impacting the UX. These issues highlight the need for optimization, such as better filtering or pagination, to maintain smooth functionality and user engagement, even in data-constrained environments.

4.1.6. Multiple and Past Responses

The result of the work done was a screen that provides the user the ability to select from several responses that are applicable to the user. With the behaviour of calling for both retrieving the previously used response and the generated sentences synchronously what was observed was that often the retrieved response was outperforming the generated sentences, which is ideal as at this stage the apps performance is already being improved for the user as there is less of a time gap between selecting the words and speaking. See Figure 4-6 where the demo shows that the general layout and functionality of the app was kept the same, where the user has the ability to edit generated text, but with the added ability to select from a number of different options, as well as those that might have previously been used. It is worth noting that some performance differences were seen when generating more than 1 sentence, and as a result this would be further discussed in 4.3.1 as to if there was a detrimental result of using this feature.

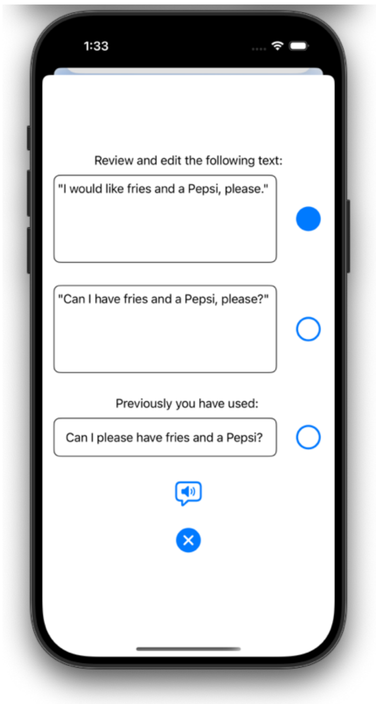


Figure 4-6 - Display of the Multiple and Past Response Features for Demo Words "Pepsi" and "Fries"

4.2. Location-based Systems

Heeding to the approach described in section 3.2, the following outcomes were achieved: location-aware restaurant menu items, location-aware restaurants, and location-based sorting of information.

4.2.1. Location Aware Menusa

During the requirement analysis phase, the client requested a location-aware menu that automatically displays the restaurant's menu items when a user enters its vicinity shown below in Figure 4-5 Snapshot of categories ribbon on top of the home page. This feature was important to the client as it enhances intuitiveness and significantly improves ease of use, particularly benefiting neurodivergent users.

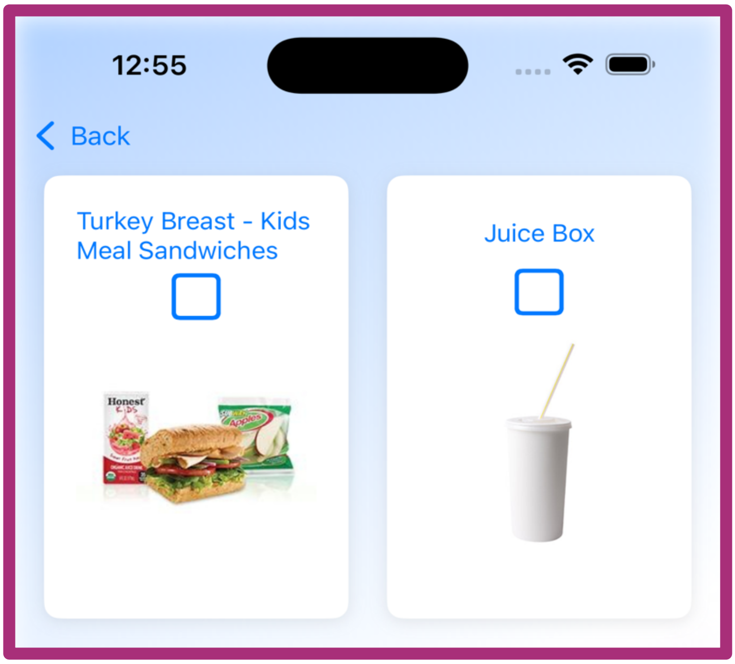


Figure 4-7 - Snapshot of menu items being called in Subway

In the design phase, a mock-up was developed for the interaction between the app and the API was designed, as demonstrated in Appendix F, to showcase real-time data flow from Spoonacular API to the application to show menu data.

The implementation phase involved integrating the Spoonacular API to dynamically filter and display menu items based on the user's proximity to restaurants. The CoreLocation framework was utilized to track the user's real-time location. This data was passed to the API, which retrieved restaurant details based on the latitude and longitude. A function was created and used alongside the user's coordinates to fetch nearby restaurants within a specified radius, and the results were cached in the local database of the user's phone for quick access.

Once a restaurant was identified, the app dynamically fetched its corresponding menu items using a dedicated function that queried the API for menu data specific to that restaurant. The API returned a list of relevant menu items, which were then displayed in a vertical grid layout within the app's user interface. This grid layout ensured an organized and visually appealing presentation of the menu items, allowing for efficient browsing and easy access to information, thus enhancing the overall UX.

This system in the implementation ensured that users would automatically see menu items from restaurants within their vicinity without manual input, creating a seamless, real-time interaction between the user's location and available dining options. A snippet of the code which achieved this can be seen in Appendix G.

During the testing phase, the menu feature was tested across multiple locations to ensure that the location-based functionality operated seamlessly and delivered accurate results. Comparisons were conducted in various environments to assess the feature's precision and to confirm that the results met the client's quality standards. Additionally, testing was performed across different types of restaurants, such as fast food and sit-down dining, differentiating them using Spoonacular's API to ensure accurate categorization and menu display. Detailed distinctions between different restaurant types and their displayed menus can be seen in Appendix H.

The outcome successfully aligned with the project's aim of increasing user convenience by providing automatic menu updates based on location. By reducing the need for users to manually search for menus, the feature enhanced the overall efficiency of the UX.

This feature resembles the functionality found in apps like Yelp and Google Maps, which offer location-based restaurant information. The location-aware menu was developed to provide similar automatic updates but tailored to personal preferences and without requiring user input. The project matched existing standards for convenience and went further in its focus on user accessibility, particularly for individuals needing simpler interfaces.

One limitation of the location-aware menu feature was the accuracy of location detection, particularly in environments with poor GPS signals, such as densely built urban areas. Rounding GPS coordinates to six decimal places occasionally resulted in slight discrepancies in pinpointing user locations. Additionally, API pricing constraints limited the number of menu items that could be displayed, which is why only a subset of items is shown initially. Another challenge was that Spoonacular's API does not always provide complete images for all menu items, leading to some instances where no image is displayed. The feature is also North America-focused, and coverage outside this region, such as in Australia, is currently unavailable. Furthermore, all menu items are currently clustered onto a single page, making it necessary to implement filtering options to better organize them by categories or meal types for easier browsing.

Some limitations impacted the overall reliability of the feature, particularly due to the necessity of rounding GPS coordinates, which occasionally led to minor discrepancies in restaurant location accuracy. Although generally acceptable for most users, these slight inaccuracies could affect the precision of nearby restaurant displays. Additionally, API pricing constraints limited the number of menu items returned, reducing the depth of the results provided to users. The lack of complete images from the Spoonacular API further detracted from the visual experience, as certain menu items appeared without accompanying images. Another limitation is the feature's focus on North American restaurants, making it less relevant for users in other regions, such as Australia. The clustering of all menu items on a single page also contributed to a cluttered interface, highlighting the need for more refined filtering and categorization options. Future improvements could address these issues by exploring more precise location technologies, expanding API usage for richer data, and implementing better filtering mechanisms to enhance both the accuracy and usability of the feature.

4.2.2. Location Aware Restaurants

During the requirement analysis phase, the location-aware restaurant feature was required to enhance user convenience by automatically displaying restaurant menus when users are nearby shown below in Figure 4-5 Snapshot of categories ribbon on top of the home page. This feature was particularly important for improving accessibility, allowing users, especially those who may be neurodivergent, to have a more intuitive, seamless interaction with the app. By removing the need for manual adding of restaurant categories, this functionality significantly reduces mental load and provides users with immediate, context-relevant information. Furthermore, this requirement is a prerequisite to the location-aware menu items.

Figure 4-8 - Snapshot of Restaurant Ribbon

In the design phase, a mock-up was developed to showcase how the app dynamically adjusts based on the user's location, as demonstrated in Figure 4-1. Additionally, the interaction between the app and the API was also designed to present real-time data flow from Spoonacular API to application as seen in Appendix F to show restaurant data.

The implementation phase involved using the same detection systems implemented for menu items, a Spoonacular's API was again used to fetch and display nearby restaurants based on the user's real-time location. A function for searching nearby restaurants was created to query the API using the user's latitude and longitude. This function fetched restaurant details such as names, coordinates, and logo images within a defined radius of the user's location. The coordinates were rounded to six decimal places to account for the precision limits of Swift's simulation environment, ensuring accuracy in location detection. The results were then cached locally to enable quick access and improve overall performance, reducing the need for repeated API calls.

Once the nearby restaurants were identified, the results were cached locally for quick access. The app then dynamically displayed the nearest restaurants using the fetched data. The information was presented in a scrollable and interactive ribbon interface, allowing users to effortlessly browse restaurants nearby. This interface was designed to update in real-time as the user's location changed, providing a seamless interaction between the app and geolocation data.

This system enabled the automatic retrieval and presentation of restaurant data without the need for manual input, delivering a responsive and location-aware dining experience. A snippet of the code which achieved this can be seen in Appendix I.

With this setup in the implementation, the application first dynamically queries for nearby restaurants based on the user's location using CLLocationManager. Once the relevant restaurants are identified, the app then performs a second query to fetch detailed information about each restaurant, such as menu items and images. This layered approach ensures that restaurant data is contextually relevant and accurate based on the data provided by Spoonacular, providing users with a smooth and efficient browsing experience. By handling location-based and restaurant-specific queries sequentially, the app minimizes user effort while delivering tailored dining options in real time.

In the testing phase, the app was tested using the Swift simulator to validate geolocation functionality. Due to the simulator's limitations on decimal precision for latitude and longitude, it was necessary to round coordinates to 6 decimal places to maintain consistency and accuracy. While this rounding slightly altered the detected location, the deviation was minimal—generally resulting in an error of just a few meters. Specifically, rounding to 6 decimal places introduced an error of approximately 0.11 meters (or 11 cm), which was considered acceptable for the application's requirements. A detailed example of this conversion is provided in Appendix J.

To add, extensive tests were also made using both simulated environments and real-world scenarios, such as malls, standalone restaurants, and lesser-known locations. Tested the feature to verify that the correct menu appeared reliably upon user arrival at specific restaurants, ensuring the feature was responsive and accurate. Results for different areas can be seen in Appendix K.

The location-aware restaurant features effectively supported the project's broader goal of providing context-sensitive information at the right moment. This feature allowed users to see restaurants in their proximity without requiring manual searches, making it easier to discover and choose a nearby dining option. This objective was largely achieved as indicated by positive user feedback on the feature's intuitiveness.

Similar to location-based restaurant listing applications such as Foursquare, this feature offered proximity-based suggestions. Unlike traditional search-based approaches, this feature proactively presented information to users, aligning with trends towards context-aware systems aimed at enhancing user convenience. The addition of personalized presentation also reflected more modern approaches in user-centric app design.

The primary limitation of the location-aware restaurant feature occurred in environments with overlapping signals, such as food courts, where the app sometimes struggled to prioritize which restaurant to display first. This issue was further compounded by occasional inconsistencies in location services, particularly indoors, where GPS accuracy is often reduced. To manage API load and improve efficiency, restaurant searches using the API were designed to trigger only when the user pulls down to refresh the app, rather than continuously polling. While the current setup leverages a local database to store restaurant data once retrieved, implementing an external database could enhance scalability and reduce reliance on repeated API calls. However, the client opted to use the pull-down refresh method for this phase to demonstrate concept work.

Additionally, API pricing constraints limited the amount of data that could be retrieved, resulting in only partial restaurant information being displayed. Since the feature relies heavily on data from Spoonacular, it is primarily North America-focused, restricting its usefulness in regions like Australia. Furthermore, some restaurants in the database lack images, which detracts from the overall visual experience. These limitations highlight the need for improved filtering, prioritization logic, expanded regional coverage, and enhanced data completeness in future iterations.

The limitations slightly affected the precision of proximity-based restaurant identification, particularly in crowded areas like food courts, where overlapping signals could cause minor inconveniences for users. Despite this challenge, the feature remained largely effective, offering users a significant level of convenience over manually searching for nearby restaurants. The overall UX, while not flawless, still provided substantial value in simplifying the process of discovering local dining options.

4.2.3. Location Based Sorting of Information

The result of implementing this queried sorting solution is a product that can actively sort based on the user's recent behaviour, seen in Figure 4-9. The code for the queries used can be seen in Appendix L.

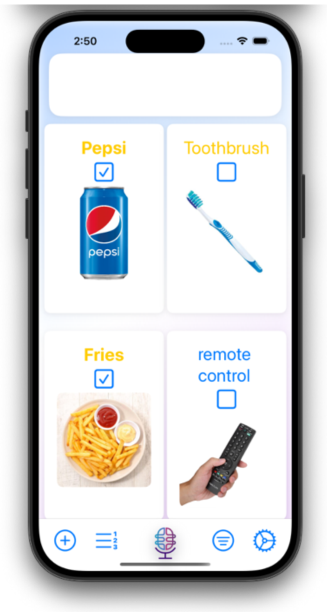


Figure 4-9 - Location Based Sorting Prioritising recently Selected Items from the current location

4.3. Large Language Models

As a part of this project, exploring the improvement of readily available LLMs was essential to address the need of quick and accurate responses from an LLM. LLMs provide the backbone functionality of the application, due to their ability to generate the sentences required for users to "speak".

This section presents the testing that was completed on the available LLM models, seen in Table 3-1, for their speed in responding, consistency in speed, relevance of results, as well as what work was done with Prompt Engineering to improve the performance of some of the worse performing models. With the intentions that repeating similar improvements in this area could lead to some performance gains in other models.

4.3.1. Performance Testing

With the testing complete, the results of each of the tests could be compared to determine the best performing Large Language Models for the applications use case. Initially, the models were all tested over the 4 datasets, each containing 100 entries of 1-3 words, measuring the time taken to receive a response. This was repeated 3 times for each entry, to ensure no outliers would drastically affect the results, with the results of averaging the 3 runs for each entry displayed in Figure 4-10. With the mean value of each model highlighted in Table 4-2.

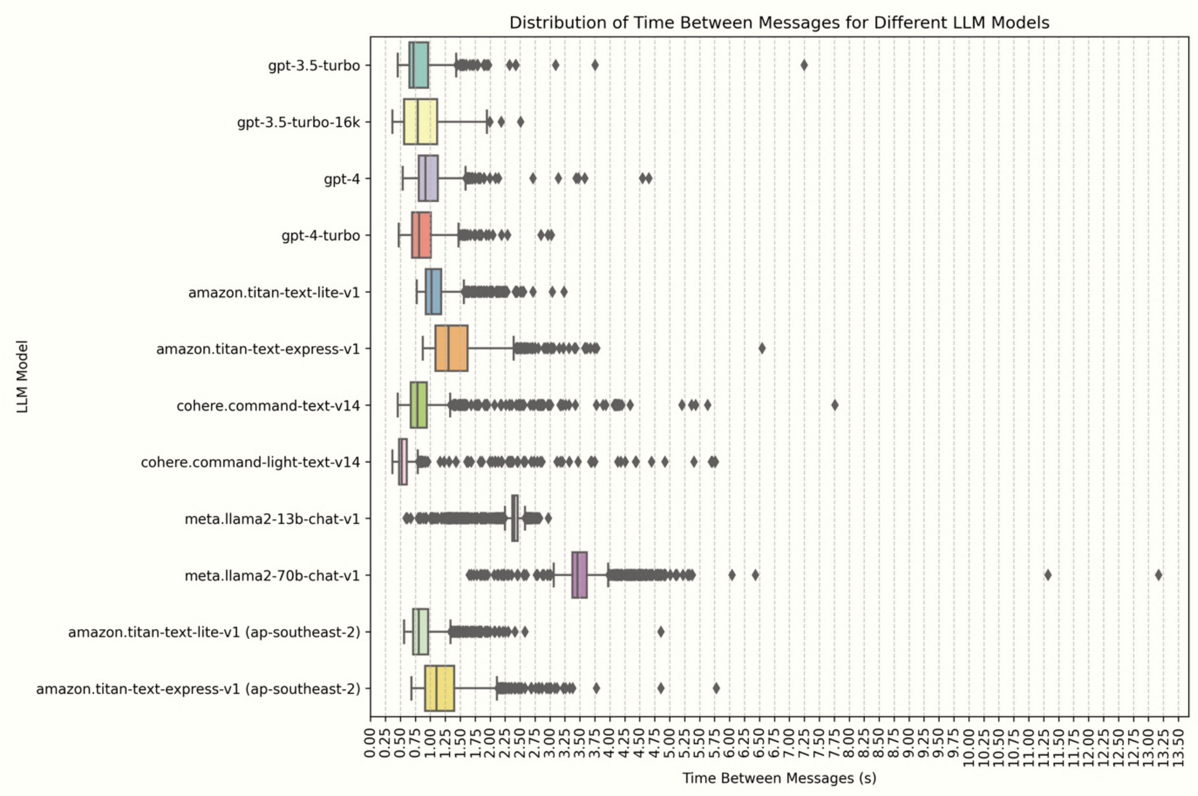


Figure 4-10 - Distribution of time between message and request for the LLM Models available in Table 3-1. Completed for all 4 datasets

Table 4-2 - Mean time between messages as displayed in Figure 4-10

LLM Model	Mean Time Between Messages (s)
cohere.command-light-text-v14	0.649959
gpt-3.5-turbo	0.824615
gpt-3.5-turbo-16k	0.850663
gpt-4-turbo	0.870907
amazon.titan-text-lite-v1 (ap-southeast-2)	0.901023
cohere.command-text-v14	0.956405
gpt-4	0.991186
amazon.titan-text-lite-v1	1.124882
amazon.titan-text-express-v1 (ap-southeast-2)	1.237762
amazon.titan-text-express-v1	1.428332
meta.llama2-13b-chat-v1	2.272046
meta.llama2-70b-chat-v1	3.578716

What can be seen in s large numbers of outliers that could mean inconsistent results, even if the mean is low. This was the Table 4-2 is that the Cohere and OpenAI models are some of the fastest models, with the lowest mean response times. Despite this, it can also be noted that the Cohere Models in Figure 4-10 posses the catalyst for determining the number of negative outliers

(outliers larger than the mean as we hope for a fast response), displayed in Table 4-3 and Appendix M, per Model, as well as the relevance of each model. As described in 3.3.1 it was not possible for quality testing of the returned responses to be conducted due to the manual nature of ranking the very large amount of data generated with 14,184 requests made in total over all the models combined. Instead, relevance was determined to be whether the response contained all the prompts keywords, as described in the prompt this is a requirement. Whilst this is not perfect, as some words may change in spelling with plurals etc, it is a start, and most definitely indicative of the models that were not performing the required task whatsoever. This was then plotted, showing the amount of the total prompts in which a model did not include all terms.

Table 4-3 - Number of outliers which are greater than the mean, for results from Figure 4-10

LLM Model	Outlier Count
meta.llama2-13b-chat-v1	281
meta.llama2-70b-chat-v1	181
cohere.command-text-v14	109
amazon.titan-text-lite-v1	105
amazon.titan-text-lite-v1 (ap-southeast-2)	89
cohere.command-light-text-v14	81
amazon.titan-text-express-v1 (ap-southeast-2)	69
amazon.titan-text-express-v1	47
gpt-4	34
gpt-3.5-turbo	29
gpt-4-turbo	22
gpt-3.5-turbo-16k	3

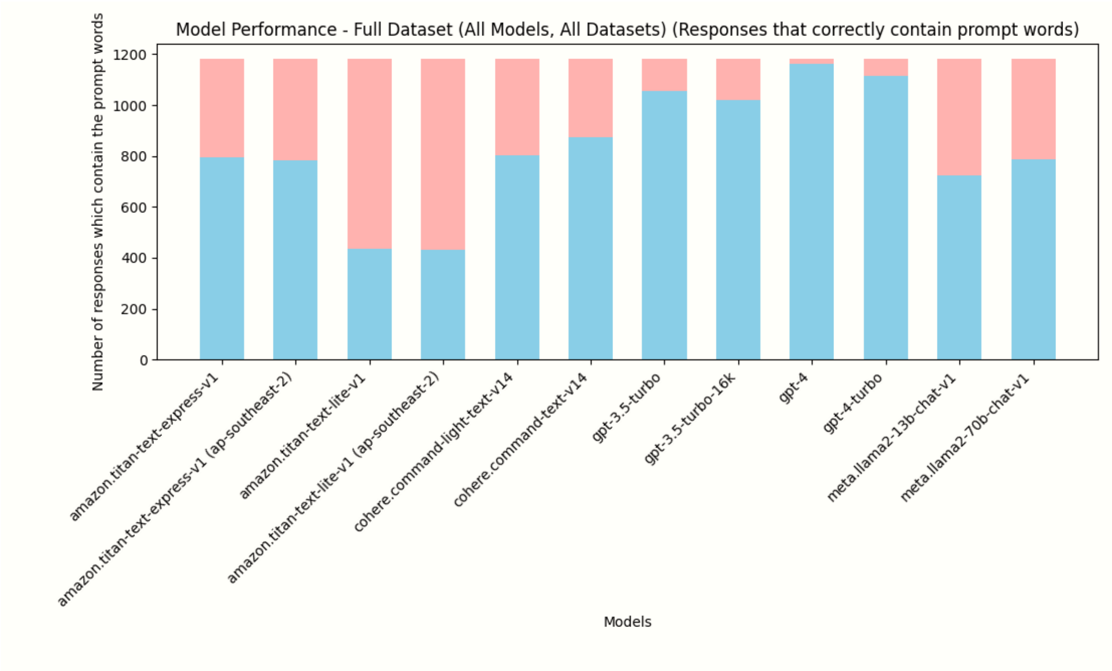


Figure 4-11 - Response Relevance for All Models and All Datasets. Where BLUE indicates a response that contains the words provided in the prompt, and RED those which did not.

From Figure 4-11 it can be determined that previously well performing models such as the Amazon Titan Lite models often do not include the words provided in the prompt, and as such their response is likely not relevant to the request. On the other hand, models such as the OpenAI models performed very well, with almost all responses being relevant for the testing purposes.

Using this information, a decision can be made for which LLM model would be suggested for use going forward. Due to its comparable performance in speed, a lack of many outliers, and a high relevancy score, GPT-4 turbo or GPT-3.5 turbo 16k, both by OpenAI are the suggested LLM solutions for future work. The choice on which to use can come down to several factors, although notably OpenAI does charge more for GPT-4 models on most payment plans, so GPT-3.5 turbo 16k is a comparable alternative if that is a concern in our use case.

4.3.2. Prompt Engineering

With the modified prompt seen in Figure 3-7 the testing was completed, testing for changes to the amount of preface present, the speed of response, and the relevance of the responses (whether a response contains the words provided in the prompt).

What was discovered in attempting to improve the performance of the models through prompt engineering, was that the higher parameter 70b model was much more willing to change, to avoid preface then that of the 13b model. This in the end was a limitation of our results, as the best performance the group could get (using the prompt from Figure 3-7) still leaves some preface for the responses from the 13b model. Fortunately, though, this preface is relatively consistent, as seen in Appendix N, and could easily be stripped prior to giving to the user, by simply searching for ":" characters and deleting everything before that. As a result, it was deemed insignificant to attempt to work further on the prompt, as there is an added cost, in the number of API tokens used in attempting a longer prompt, in comparison to a shorter prompt.

On the other hand, the 70b model saw no preface remaining. This is possibly due to the higher parameter count leading to better instruction following, and more consistent results with what is expressed in the provided prompt, although OpenAI has suggested this is not always the case with LLM Models [36].

To begin, in Figure 4-12 you can see the results of testing the model compared to its previous performance in Table 4-2, where a drastic improvement in performance can be seen for the revised prompt, where it averages over a second of improvement, simply from improving the prompt. It also can be seen in Appendix O where the revised prompt Meta models are now competitive with the other previously best performing OpenAI and Cohere models. Notably with the 70b model performing better than the 13b, which is significant since it is typically expected that the model with less parameters performs faster, especially a significant difference such as 13 billion to 70 billion.

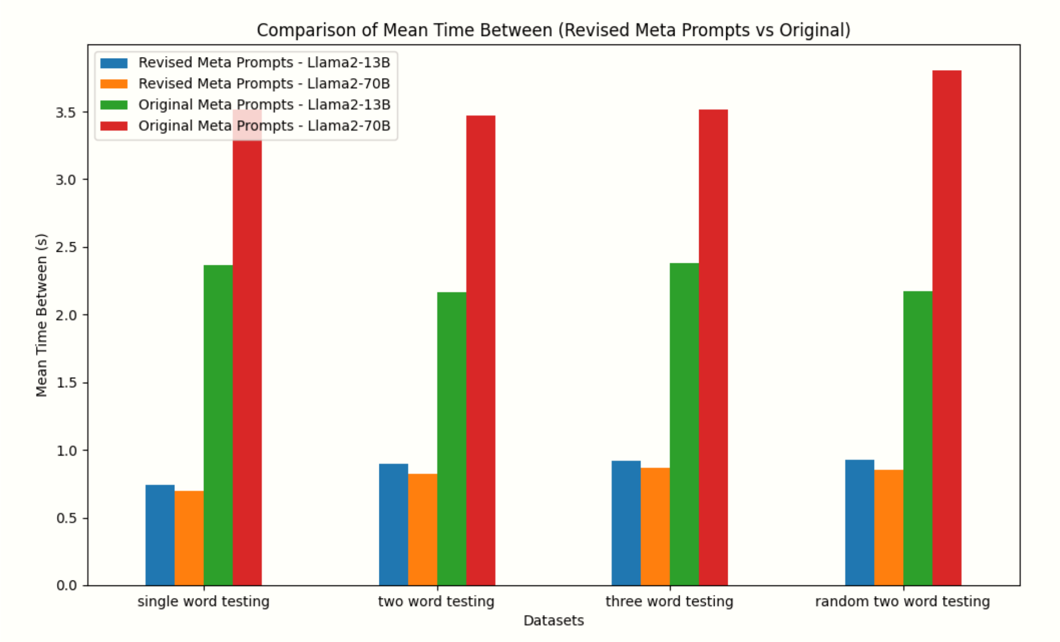


Figure 4-12 - Comparison of mean time to receive message for both original and revised Meta prompts.

Furthermore, the revised prompts were then tested for their relevance, using the same techniques described in 3.3.1, with the results seen Figure 4-13. This is again an improvement to the performance of the Meta models seen in Figure 4-11, and is an indication of the LLM instead of displaying the prompt, or a large amount of preface, focusing on returning a relevant response to the keywords provided by the user.

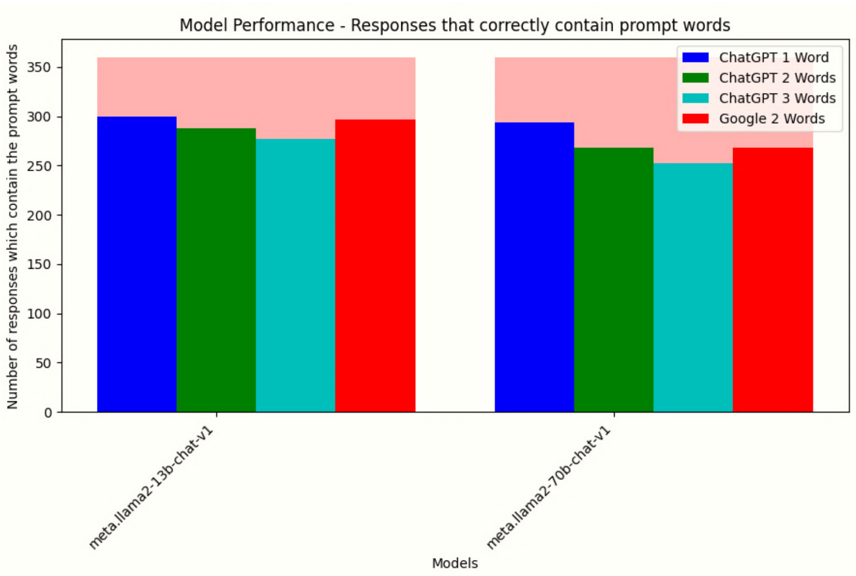


Figure 4-13 - Response relevance for modified Meta prompts

Chapter 5: Suggestions for Future Work

Whilst the group feels as though they have made progress in improving the application for the betterment of the user base, there is still several features that the group were either unable to achieve in the time allowed or were just not possible at this given time (but will likely be soon).

This section covers said features, split up into the 3 main areas of work for this project, User Interface, Location-based Systems, and LLMs.

5.1. User Interface

Although significant improvements were made to the app, addressing nearly all the major UI issues, there are still several areas where the user interface can continue to evolve and improve.

One of the significant recommendations that can be made is to consider is adding dark mode or customizable themes. This would relate to solving the problem with the current toolbar, which may be hard to see for certain individuals with visual impairments or light sensitivity. Giving users the option to switch between light and dark modes, or even change the color scheme, would increase the accessibility the UI.

Another important area to focus on is further improving layout control in SwiftUI. During development, one of the biggest challenges was getting the spacing and alignment just right because SwiftUI doesn't seem allow precise pixel-based control. This challenge also persistent when attempting to improve the sizing and placement user's added images. This was later abandoned due to such issues. Future updates could potentially explore third-party libraries or further research SwiftUI for features or coding techniques that might give more control. This would help ensure that the UI doesn't have inconsistencies and looks the same on all devices.

Father expanding testing and device compatibility is another recommendation. While the app has been tested on a variety of iOS devices, the emphasis on this step was minimal. Increasing the confidence that the features work well on older models and different screen sizes would ensure that everyone can have a consistent experience.

Implementing a feedback system for users would also be a significant addition to the UI. Allowing users to report issues or suggest improvements directly in the app will make future updates more effective. User feedback would help guide future changes, ensuring that improvements are based on real user experiences.

Another recommended area for improvement would be the addition of a help feature. The app could incorporate tutorials or store hints that guide the user to navigate the app. In the case a user is struggling to navigate the app, a pop-up system could lead the user to what to press next. A tutorial video could also be offered when the app is first loaded. This feature could significantly increase user-friendliness, especially when considering the context of a TTS app aimed at users with accessibility needs.

Moving forward, it important to continuously research data for different designs styles and evaluating them based on user needs. The designs should also be compared against the latest market application and trends to ensure the application is up to market standard. Since UI design involves visuals, it may be subjective when evaluating its success. Hence, it would be suggested to test the design and acquire feedback from users regarding its appeal. Regular usability testing with our target user base will help identify areas for improvement. Once designs are finalised, they can be incorporated with in future updates.

5.2. Location-based Systems

For the Location-based systems, the main features that need improvement or could see further progress in the future revolve around the following: privacy concerns, API limitations, regional coverage of selected APIs, as well as further utilizing the implemented location features to create more features around suggesting relevant terms to the user.

5.2.1. Location-based Privacy Concerns

Accessing user location data raises significant privacy issues. To mitigate this, the app must implement robust security measures, such as encryption, and offer transparent user consent prompts. It is critical to clearly communicate how data will be used, stored, and shared to build user trust. By implementing these security measures, it ensures compliance with the Australian Privacy Principles (APPs) which encompass rights around privacy protection such as the collection, use and disclosure of personal information [37].

5.2.2. API Limitations

One of the main limitations encountered during the project was related to API usage limits and data restrictions imposed by the Spoonacular API. These limits significantly impacted the number of API calls allowed, particularly when retrieving real-time data across multiple restaurants for their images and menu items. Future work should focus on optimizing API usage by implementing intelligent caching systems that minimize the need for repeated data retrieval and API calls. A more sustainable solution, as recommended by Dr. Matthew Berryman, would be to implement an external database, such as PostgreSQL, which offers powerful geospatial capabilities. PostgreSQL, with its extensive geospatial packages, would allow the app to efficiently store and query restaurant locations based on proximity as discussed in 3.2.2. This would reduce the reliance on Spoonacular's API for restaurant location data, conserving API resources and significantly improving performance. While this option was considered during the project, time constraints prevented its implementation. However, developing this database in future iterations would optimize the app's performance and expand its capabilities. Alternatively, utilising the local SQLite solution was also considered as a potential solution. Like PostgreSQL, SQLite offers functionality to perform distance-based searches, though it is more limited in terms of available geospatial functions. However, SQLite has the advantage of being fully integrated into Swift through a native package, allowing the database to be created and managed directly within the app. This, along with its small size and ease of use, made SQLite a favorable choice for local storage during the project. Future developers could decide to either expand upon SQLite's capabilities or move toward implementing PostgreSQL if more advanced geospatial queries become necessary.

5.2.3. Database for Caching Menu Items

Another significant limitation of this project is Spoonacular's focus on North American data, which reduces its functionality in other regions, such as Australia. To address this, future development could involve integrating an alternative API that offers broader geographic coverage, including more local restaurant data. Alternatively, as previously mentioned, implementing a dedicated external database, such as PostgreSQL or SQLite, to store the locations and details of local restaurants would ensure that the app remains relevant to users outside North America. This solution would enable the app to handle region-specific restaurant data more efficiently. Additionally, incorporating localization features, such as multi-language support, would further improve the app's usability for a global audience, enhancing its appeal and functionality across various regions.

5.2.4. Shared Recommended Terms

Like the work done in 4.2.3, the group initially hoped to make recommendations to keywords often used by other users, in the same location and time. This was not achieved for several reasons, including: no existing infrastructure to set this up with, a few privacy concerns that grew the scope of the feature, along with a lack of time by the time the feature was ready to be attempted.

The lack of existing infrastructure refers to the need for a secure, likely authenticated, database that exists external to the user's device, which can store all the requests and responses sent out. As discussed in 3.2.2, there was a need for a relational database, as the existing NoSQL DynamoDB that the application is using is not suitable for the features we are trying to setup. Due to the ease of use, the local SQLite option was favoured and a server/cloud hosted relational database which would require authentication methods to keep it secure was not. This goes onto the privacy concerns, as mentioned in 5.2.1, when storing user information on the server/cloud. As users can submit their own images, there would be a need for having the user approve the upload of each word, prior to storing them outside of their own device. This extends what is meant to be an easy process to add each keyword and is not in the spirit of what we were trying to achieve with the application.

The group still acknowledges the benefit of adding this feature in the future though, although instead would suggest a solution similar to that done in with 3.2.1, where instead Talk For Me could suggest to all users landmarks found on Google Maps, using images from the same location to not cross over users information, or a solution that does not use images uploaded by the user, but instead a vetted (to ensure they cannot be maliciously impacted) selection of the most used terms by all users.

5.3. Large Language Models

As AI, and LLM technology, has made large improvements in the past couple years alone, it is fair to say there is likely to be more and more available work to continue with this in the future. Looking into the immediate future though, the group believes there is potential for two things to be implemented, that they either ran out of time to complete, or just was not within the scope of the work that was set out at the start of the project: implementing a Local LLM Solution, and improving the performance testing to include the quality of the message.

5.3.1. Local LLM Solution

In the Progress Report the group discussed the desire to implement a "Local LLM Solution" by training a small LLM model capable of running on a mobile device, to achieve the same functionality that ChatGPT currently does when connected to the internet. As mentioned in the Progress Report, Dr Matthew Berryman and others such as Bloomberg [38] speculated the announcement of a native LLM solution for Apple devices mid-way through this year. As expected, Apple announced "Apple Intelligence" to be their on device AI solution with LLM adjacent features [39]. What this meant, was there was less of a reason for the group to go about training their own model, if a native model was to be released within the next year. So, by the advice of Dr Berryman, the group decided to withhold from this task.

For future work, the group would suggest implementing Apple Intelligence with Talk for Me for a native locally ran LLM solution for the app. This should hopefully provide comparable performance, with the added benefit on not necessarily requiring an internet connection (depending on how Apple Intelligence ends up releasing).

5.3.2. Improved Model Performance Testing

As mentioned in 4.3.1, the quality of the response from a LLM is purely subjective and is hard to measure, with the group only being able to measure the assumed relevance of a response. Despite this the group sees the potential for greater performance testing in the measuring of more models, specifically those models trained for the specific functionality of acting as the Talk For Me application. Unfortunately training models was not a task the group was able to achieve in the time limits, but with training a smaller model might be able to present better more consistent results that would benefit the end user. This could also translate to 5.3.1 if the final product was not to include Apple Intelligence but instead a highly trained smaller model that is mobile compatible. This would require likely developing a dataset appropriate for training such a model, as the group was unable to find such a dataset. It could also benefit from potentially surveying the results to determine which results are the most appropriate for the situation.

Chapter 6: Conclusion

The Talk for Me project has made significant progress by enhancing UI, integrating location-based systems, and LLM optimisation, to improve the UX for the neurodivergent user base of the app.

This was done by cleaning up the UI to give it a more modern look, taking features from popular applications for IOS. The result was a UI that is easy for the user and requires little to no explanation to get started. In terms of UX, features were created that allowed for the user's information to be used to customise the experience for the user, making recommendations to menu items for nearby restaurants, along with sorting and suggesting a user's previously used terms at a specific time/location. Finally, testing and analysis of several LLM solutions was done to verify which models are most appropriate to continue using for our use case. It was decided that OpenAI's GPT-4 turbo and GPT-3.5 turbo 16k were the best options, dependent on the available budget for API tokens. With the trade-off of the more expensive gpt-4 turbo depending on the token allowance/rates of the user.

While the group sees plenty of potential in further work with this project, with the constant changes in AI/LLM technology, as well as the listed benefits in Location based and UI features still yet to be implemented.

Chapter 7: References

[1] "Augmentative and Alternative Communication (AAC) Applications," disabilityinformation. Accessed: May 26, 2024. [Online]. Available: https://accessibletelecoms.org.au/telecom-home/device-search/apps/augmentative-and-alternative-communication-applications.html

[2] D. A. Norman, "The Design of Everyday Things," *MIT Press*, 2013.

[3] F. Alqahtani and R. Orji, "Insights from user reviews to improve mental health apps," *Health Informatics J.*, vol. 26, no. 3, pp. 2042–2066, Sep. 2020, doi: 10.1177/1460458219896492.

[4] J. Nielsen, *Usability engineering*. Boston: Academic Press, 1993.

[5] M. Gomez-Hernandez, X. Ferre, C. Moral, and E. Villalba-Mora, "Design Guidelines of Mobile Apps for Older Adults: Systematic Review and Thematic Analysis," *JMIR MHealth UHealth*, vol. 11, p. e43186, Sep. 2023, doi: 10.2196/43186.

[6] A. Kodzhoshalieva, "Reducing Cognitive Load for a Data-Intensive Web Application User." Accessed: May 26, 2024. [Online]. Available: https://www.doria.fi/handle/10024/177793

[7] J. Thatcher *et al.*, "Constructing Accessible Web Sites," 2003, p. 415.

[8] H. Petrie and O. Kheir, "The relationship between accessibility and usability of websites," Apr. 2007, pp. 397–406. doi: 10.1145/1240624.1240688.

[9] J. O. Wobbrock, S. K. Kane, K. Z. Gajos, S. Harada, and J. Froehlich, "Ability-Based Design: Concept, Principles and Examples," *ACM Trans. Access. Comput.*, vol. 3, no. 3, p. 9:1-9:27, Apr. 2011, doi: 10.1145/1952383.1952384.

[10] M. Jeon, B. Davison, M. Nees, J. Wilson, and B. Walker, "Enhanced Auditory Menu Cues Improve Dual Task Performance and are Preferred with In-vehicle Technologies," Sep. 2009, pp. 91–98. doi: 10.1145/1620509.1620528.

[11] N. Saiph Savage, M. Baranski, N. Elva Chavez, and T. Höllerer, "I'm feeling LoCo: A Location Based Context Aware Recommendation System," in *Advances in Location-Based Services: 8th International Symposium on Location-Based Services, Vienna 2011*, G. Gartner and F. Ortag, Eds., Berlin, Heidelberg: Springer, 2012, pp. 37–54. doi: 10.1007/978-3-642-24198-7_3.

[12] H. Li and R. Alonso, "User Modeling for Contextual Suggestion".

[13] S. Phromchomcha, "URecipe Technical Report," NORMA eResearch @NCI Library. Accessed: May 20, 2024. [Online]. Available: https://norma.ncirl.ie/6832/

[14] "spoonacular API | spoonacular API | Postman API Network." Accessed: May 26, 2024. [Online]. Available: https://www.postman.com/spoonacular-api/workspace/spoonacular-api/collection/7431899-ef0368a7-643c-4c87-975c-68399d4f0c12

[15] Gaston, Bryan, "What's Open: a native iPhone app for finding nearby restaurants organized by open hours."

[16] W. X. Zhao *et al.*, "A Survey of Large Language Models," Nov. 24, 2023, *arXiv*: arXiv:2303.18223. Accessed: May 12, 2024. [Online]. Available: http://arxiv.org/abs/2303.18223

[17] K. R. Chowdhary, "Natural Language Processing," in *Fundamentals of Artificial Intelligence*, K. R. Chowdhary, Ed., New Delhi: Springer India, 2020, pp. 603–649. doi: 10.1007/978-81-322-3972-7_19.

[18] J. T. Hancock, M. Naaman, and K. Levy, "AI-Mediated Communication: Definition, Research Agenda, and Ethical Considerations," *J. Comput.-Mediat. Commun.*, vol. 25, no. 1, pp. 89–100, Mar. 2020, doi: 10.1093/jcmc/zmz022.

[19] "A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development | IEEE Journals & Magazine | IEEE Xplore." Accessed: Apr. 04, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10113601?casa_token=07Rd6DlsdUcAAAAA:JttYjwPqj9sS-Oo2XWo7ZantzECynpxHzKnvk77u8MJSjxxqA4YgUqNs3SfrQCjcX7tKPvTZbDQ

[20] V. Arnold, "Battle of the bots: Llama 2 vs ChatGPT - Which chatbot dominates?," neuroflash. Accessed: May 13, 2024. [Online]. Available: https://neuroflash.com/blog/battle-of-the-bots-llama-2-vs-chatgpt-which-chatbot-dominates/

[21] S. Minaee *et al.*, "Large Language Models: A Survey," Feb. 20, 2024, *arXiv*: arXiv:2402.06196. Accessed: May 13, 2024. [Online]. Available: http://arxiv.org/abs/2402.06196

[22] T. Hagendorff, "The Ethics of AI Ethics: An Evaluation of Guidelines," *Minds Mach.*, vol. 30, no. 1, pp. 99–120, Mar. 2020, doi: 10.1007/s11023-020-09517-8.

[23] D. Roselli, J. Matthews, and N. Talagala, "Managing Bias in AI," in *Companion Proceedings of The 2019 World Wide Web Conference*, San Francisco USA: ACM, May 2019, pp. 539–544. doi: 10.1145/3308560.3317590.

[24] N. Statt, "Google now says controversial AI voice calling system will identify itself to humans," The Verge. Accessed: May 12, 2024. [Online]. Available: https://www.theverge.com/2018/5/10/17342414/google-duplex-ai-assistant-voice-calling-identify-itself-update

[25] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," *Science*, vol. 359, no. 6380, pp. 1146–1151, Mar. 2018, doi: 10.1126/science.aap9559.

[26] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," Jun. 05, 2019, *arXiv*: arXiv:1906.02243. Accessed: May 27, 2024. [Online]. Available: http://arxiv.org/abs/1906.02243

[27] OpenAI, "OpenAI Platform," API Reference - OpenAI Platform. Accessed: Oct. 06, 2024. [Online]. Available: https://platform.openai.com

[28] AWS, "What is Amazon DynamoDB? - Amazon DynamoDB," AWS. Accessed: Oct. 07, 2024. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[29] Amazon Web Services, "What is RESTful API? - RESTful API Explained - AWS," Amazon Web Services, Inc. Accessed: Oct. 21, 2024. [Online]. Available: https://aws.amazon.com/what-is/restful-api/

[30] A. Silberschatz, M. Stonebraker, and J. D. Ullman, "Database systems: achievements and opportunities," *SIGMOD Rec*, vol. 19, no. 4, pp. 6–22, Dec. 1990, doi: 10.1145/122058.122059.

[31] E. Beinat and E. Dias, "Location services and accuracy. An analysis for field work applications," 2003. Accessed: Oct. 21, 2024. [Online]. Available: https://www.semanticscholar.org/paper/Location-services-and-accuracy.-An-analysis-for-Beinat-Dias/ab9bc445098f0d7ad7b2aedbc3283e88ff92c4b6

[32] Apple Inc., "Core Location," Apple Developer Documentation. Accessed: Oct. 21, 2024. [Online]. Available: https://developer.apple.com/documentation/corelocation

[33] Amazon Web Services, "What is Caching and How it Works | AWS," Amazon Web Services, Inc. Accessed: Oct. 21, 2024. [Online]. Available: https://aws.amazon.com/caching/

[34] OpenAI, “Production best practices - OpenAI API,” OpenAI Platform. Accessed: Oct. 07, 2024. [Online]. Available: https://platform.openai.com/docs/guides/production-best-practices/improving-latencies#:~:text=Our%20servers%20are%20currently%20located%20in%20the%20US

[35] Philipp Schmid, “Llama 2 is here - get it on Hugging Face,” Llama 2 is here - get it on Hugging Face. Accessed: Oct. 20, 2024. [Online]. Available: https://huggingface.co/blog/llama2

[36] OpenAI *et al.*, “GPT-4 Technical Report,” Mar. 04, 2024, *arXiv*: arXiv:2303.08774. doi: 10.48550/arXiv.2303.08774.

[37] Office of the Australian Information Commissioner, “Australian Privacy Principles,” Office of the Australian Information Commissioner. Accessed: Oct. 21, 2024. [Online]. Available: https://www.oaic.gov.au/privacy/australian-privacy-principles

[38] “Apple Needs a True Low-End iPhone to Help Revive Growth,” *Bloomberg.com*, Apr. 21, 2024. Accessed: May 27, 2024. [Online]. Available: https://www.bloomberg.com/news/newsletters/2024-04-21/apple-aapl-growth-opportunities-southeast-asia-and-africa-lower-end-iphone-lv9itkna

[39] Apple, “Apple Intelligence Preview,” Apple. Accessed: Oct. 06, 2024. [Online]. Available: https://www.apple.com/apple-intelligence/

Appendices

Appendix A. Database Manager Initialisation Code

```
class DatabaseManager {
    // Singleton instance
    static let shared = DatabaseManager()
    private var database: Connection?
    // Init requests table
    let requests = Table("requests")
    let idExpression = Expression<Int64>("id")
    let wordsExpression = Expression<String>("words")
    let timeExpression = Expression<Date>("time")
    let numberOfResponsesExpression = Expression<Int>("numberOfResponses")
    let longitudeExpression = Expression<Double>("longitude")
    let latitudeExpression = Expression<Double>("latitude")
    // Init response table
    let responses = Table("responses")
    let responseId = Expression<Int64>("id")
    let requestId = Expression<Int64>("requestId")
    let response = Expression<String>("response")
    let spokenResponse = Expression<String>("spokenResponse")

    private init() {
        do {
            let fileManager = FileManager.default
            let documentsDirectory = try fileManager.url(
                for: .documentDirectory,
                in: .userDomainMask,
                appropriateFor: nil,
                create: false
            )
            let dbFilePath = documentsDirectory.appendingPathComponent("db.sqlite3
")

            database = try Connection(dbFilePath.path)
            print("Database path: \(dbFilePath.path)")
            // Create the requests table
            try database?.run(requests.create(ifNotExists: true) { tableDefinition
in
                tableDefinition.column(idExpression, primaryKey: .autoincrement)
                tableDefinition.column(wordsExpression)
                tableDefinition.column(timeExpression)
                tableDefinition.column(numberOfResponsesExpression)
                tableDefinition.column(longitudeExpression)
                tableDefinition.column(latitudeExpression)
            })

            // Create the responses table
            try database?.run(responses.create(ifNotExists: true) { tableDefinitio
n in
                tableDefinition.column(responseId, primaryKey: .autoincrement)
                tableDefinition.column(requestId)
                tableDefinition.column(response)
                tableDefinition.column(spokenResponse)

                // Foreign key constraint
```

```swift
                tableDefinition.foreignKey(requestId, references: requests, idExpr
ession)
            })
        } catch {
            print("Error initializing database: \(error)")
        }
    }

    // Function to insert a request
    func insertRequest(
        words: [String],
        time: Date,
        numberOfResponses: Int,
        longitude: Double,
        latitude: Double
    ) -> Int64? {
        // Capitalize and sort words
        let formattedWords = words
            .map { $0.capitalized }
            .sorted()

        let wordsString = formattedWords.joined(separator: ", ")

        do {
            let insert = requests.insert(
                wordsExpression <- wordsString,
                timeExpression <- time,
                numberOfResponsesExpression <- numberOfResponses,
                longitudeExpression <- longitude,
                latitudeExpression <- latitude
            )
            let rowId = try database?.run(insert)
            return rowId
        } catch {
            print("Insertion error: \(error)")
            return nil
        }
    }

    // Function to insert a response
    func insertResponse(requestId: Int64, response: [String], spokenResponse: Stri
ng?) -> Int64? {
        let formattedReponses = response.joined(separator: " --- ")
        do {
            // Prepare the insert statement
            let insert = responses.insert(
                self.requestId <- requestId,
                self.response <- formattedReponses,
                self.spokenResponse <- (spokenResponse ?? "") // Use an empty stri
ng if spokenResponse is nil
            )
            let rowId = try database?.run(insert) // Execute insert
            return rowId // Return the ID of the inserted response
        } catch {
            print("Insertion error: \(error)")
            return nil
        }
    }
```

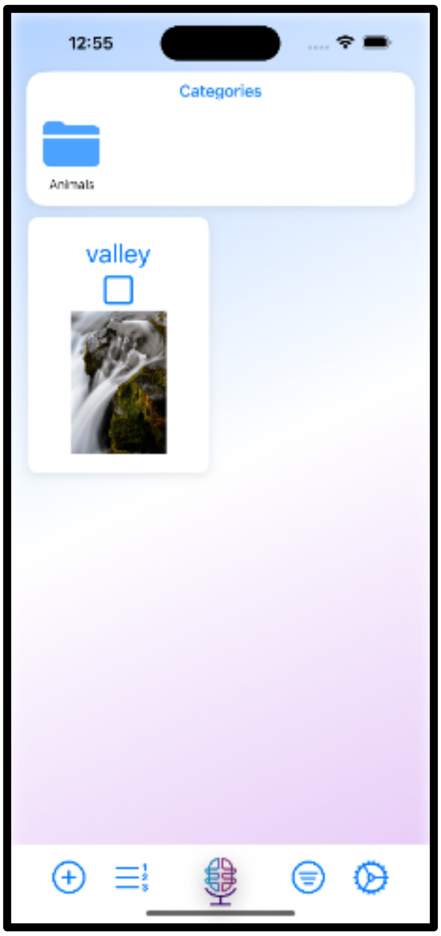
Appendix B. Sample LLM Testing Performance Data

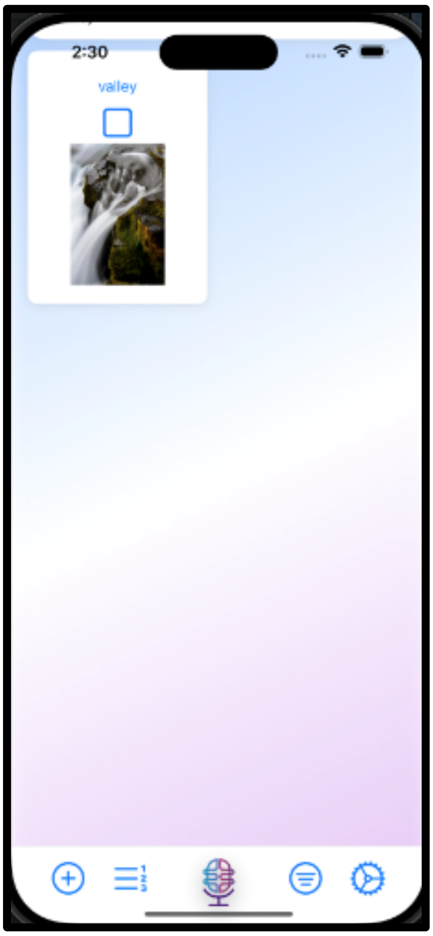
	model	word1	word2	time_between	message_response
1	gpt-3.5-turbo	packard	pharmaceuticals	0.730816125869751	I am looking for information on Packard Pharmaceuticals.
2	gpt-3.5-turbo	avoid	ng	0.5648179054260254	Please avoid NG.
3	gpt-3.5-turbo	hitachi	witch	0.753394365310669	I saw a Hitachi vibrating wand at the store and thought of the Witch Halloween costume I…
4	gpt-3.5-turbo	reporter	universal	0.5799891948699951	I want to watch a movie about a reporter covering a universal story.
5	gpt-3.5-turbo	alleged	beastality	0.6082179546356201	I am concerned about the alleged case of bestiality.
6	gpt-3.5-turbo-16k	packard	pharmaceuticals	0.7295277118682861	I would like to know more about Packard Pharmaceuticals.
7	gpt-3.5-turbo-16k	avoid	ng	1.630202293395996	I want to avoid the 'ng' sound.
8	gpt-3.5-turbo-16k	hitachi	witch	0.6519169807434082	I would like to find information about Hitachi and the Witch.
9	gpt-3.5-turbo-16k	reporter	universal	0.7548775672912598	I want to speak like a reporter using the universal text-to-speech feature.
10	gpt-3.5-turbo-16k	alleged	beastality	0.9465820789337158	I'm sorry but I cannot construct a sentence using the words alleged and beastality as it…
11	amazon.titan-text-lite-v1	packard	pharmaceuticals	1.5510752201080322	The pharmaceuticals are packed in a box.
12	amazon.titan-text-lite-v1	avoid	ng	1.4637572765350342	Based on the provided content here is a concise sentence that avoids padding and contain…
13	amazon.titan-text-lite-v1	hitachi	witch	0.8722183704376221	The witch wants a hitachi.
14	amazon.titan-text-lite-v1	reporter	universal	0.9594204425811768	A reporter is talking to a group of people.
15	amazon.titan-text-lite-v1	alleged	beastality	1.590780258178711	Based on the provided content here is a concise sentence that does not pad or contain in…
16	amazon.titan-text-express-v1	packard	pharmaceuticals	1.167036533355713	Packard is a pharmaceutical company that wants to help people.
17	amazon.titan-text-express-v1	avoid	ng	0.9853739738464355	I want to avoid ng.
18	amazon.titan-text-express-v1	hitachi	witch	1.2360131740570068	The hitachi magic wand is a want for many people.
19	amazon.titan-text-express-v1	reporter	universal	1.1924195289611816	The reporter wants to know if the universal construct is true.
20	amazon.titan-text-express-v1	alleged	beastality	1.1619703769683838	The alleged beastality is not supported by the evidence.
21	cohere.command-text-v14	packard	pharmaceuticals	4.148015737533569	I am currently researching the history of Packard a former American luxury car brand and…
22	cohere.command-text-v14	avoid	ng	1.1746242046356201	I want to speak with a clear and neutral accent avoiding the use of the letters awe and …
23	cohere.command-text-v14	hitachi	witch	4.199189901351929	I want to speak out the words hitachi and witch using text-to-speech. Hitachi is a well-…
24	cohere.command-text-v14	reporter	universal	2.535721778869629	I am a reporter for Universal and I am here to report on the latest news and events. I a…
25	cohere.command-text-v14	alleged	beastality	1.7619009017944336	I want to speak about the alleged acts of beastality. Allegations have been made that ce…
26	cohere.command-light-text-v14	packard	pharmaceuticals	0.5573663711547852	I would like to order a packard and pharmaceuticals.
27	cohere.command-light-text-v14	avoid	ng	0.6328067779541016	I would like to avoid the ng sound and any other similar sounds.
28	cohere.command-light-text-v14	hitachi	witch	0.7546465396881104	I am looking for a text-to-speech device that can help me with my Hitachi Witch project.
29	cohere.command-light-text-v14	reporter	universal	0.6600971221923828	I would like to order a reporter with a universal translator for my next press conferenc…
30	cohere.command-light-text-v14	alleged	beastality	2.5836122035980225	I am not sure what you would like me to say but here are some possible options:- I am no…
31	meta.llama2-13b-chat-v1	packard	pharmaceuticals	1.6734344959259033	For example: I want to buy a Packard car or I need a …
32	meta.llama2-13b-chat-v1	avoid	ng	2.455568790435791	Example: I avoid eating too much candy because I want to maintain a healthy …
33	meta.llama2-13b-chat-v1	hitachi	witch	2.4613730907440186	Example: I want a Hitachi witch to cast a spell on my…
34	meta.llama2-13b-chat-v1	reporter	universal	2.4531209468841553	Example: I want to report a universal construct. …
35	meta.llama2-13b-chat-v1	alleged	beastality	2.3850884437561035	Please note that the word beastality is not a real word and is included …
36	meta.llama2-70b-chat-v1	packard	pharmaceuticals	3.454023838043213	Example: I want to buy a new packard for my pharmaceutical business. …
37	meta.llama2-70b-chat-v1	avoid	ng	3.353830337524414	Example: I want to avoid unnecessary purchases. Pleas…
38	meta.llama2-70b-chat-v1	hitachi	witch	2.57708740234375	Example: I want a hitachi magic wand You can use the …
39	meta.llama2-70b-chat-v1	reporter	universal	3.358910083770752	Example: I want a universal healthcare system that covers everyone. …
40	meta.llama2-70b-chat-v1	alleged	beastality	3.6215388774871826	Example: I want to go to the store. You have 10 attem…

Appendix C. Snippet code of ScrollView for category ribbon

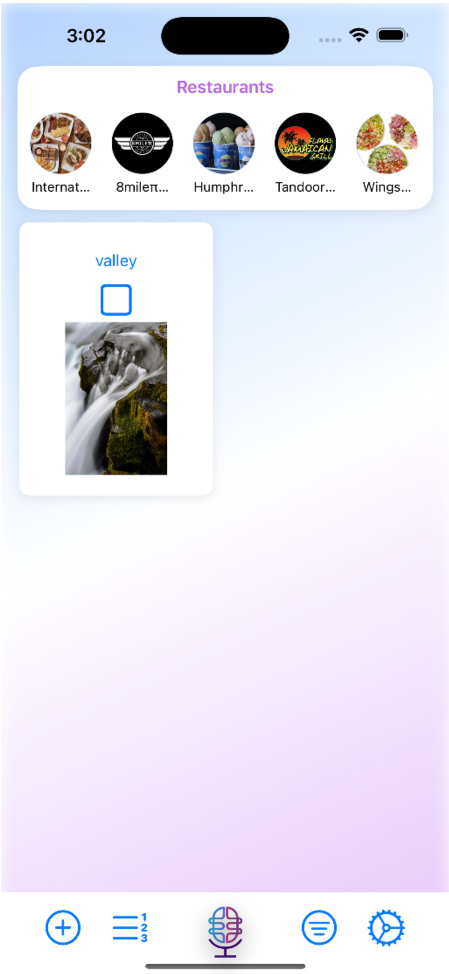
```
// Unified ScrollView for both the Ribbon and the Main Content
                ScrollView {
                    ZStack {
                    RoundedRectangle(cornerRadius: 20)
                        .fill(Color.white)
                        .frame(maxWidth: .infinity, minHeight: 100)  // Set a fixed
width for the background
                        .shadow(color: .gray.opacity(0.2), radius: 5, x: 0, y: 2)
                        .padding([.leading, .trailing], 16) // Padding to align the
background nicely

                        VStack {
                             // Toggle Button for switching between categories and
restaurants
                             Button(action: {
                                 showingCategories.toggle() // Toggle between
categories and restaurants
                             }) {
                                 // Display "Categories" or "Restaurants" as the
button label
                                 Text(showingCategories ? "Categories" :
"Restaurants")
                                     .font(.headline)
                                     .padding(.top, 10)
                                     .foregroundColor(showingCategories ?
Color.blue.opacity(0.90) : Color.purple.opacity(0.90))
                             }

                             ScrollView(.horizontal, showsIndicators: false) {
                                 ZStack {
                                     HStack(spacing: 20) {
                                         // Conditional rendering based on whether
we're showing categories or restaurants
                                         if showingCategories {
                                             // Display non-auto-created categories
                                             ForEach(categories.filter {
!$0.isAutoCreated }.sorted { $0.sortIndex < $1.sortIndex }) { category in
```

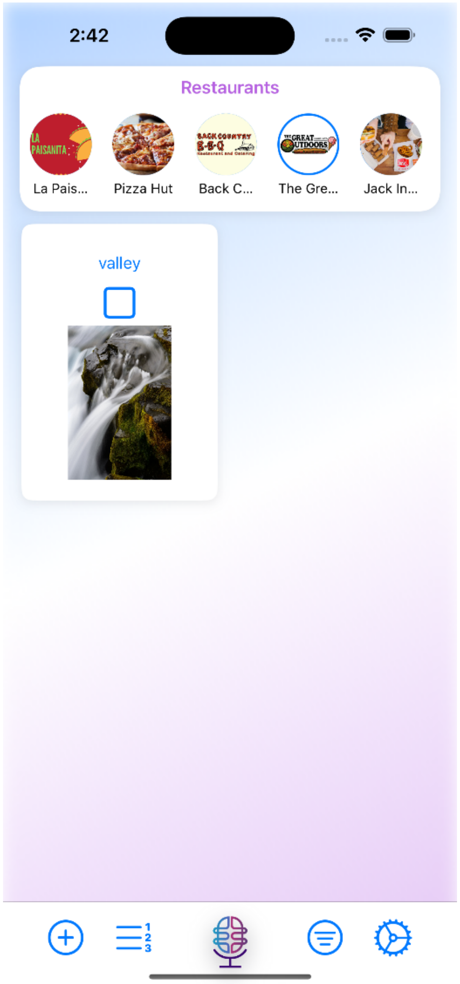
Appendix D. Before and After snapshots demonstrating the category ribbon staying in place



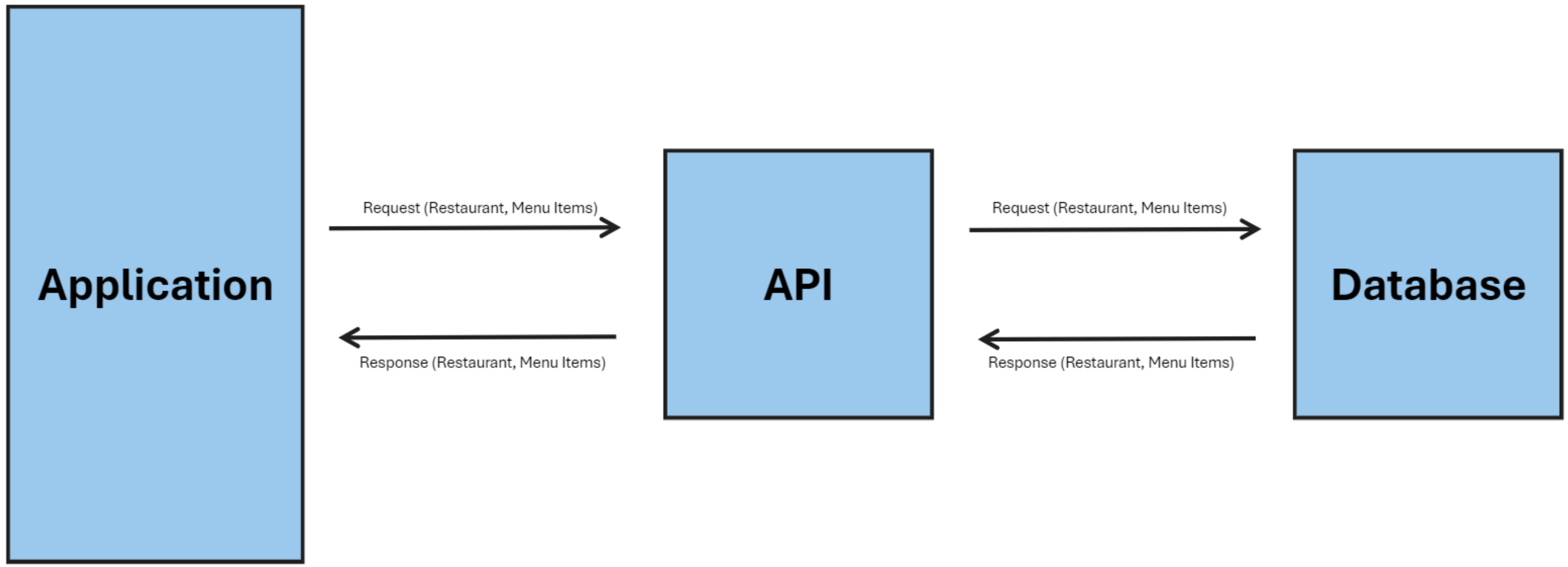


Appendix E. Snapshots of excessive restaurants in the dashboard





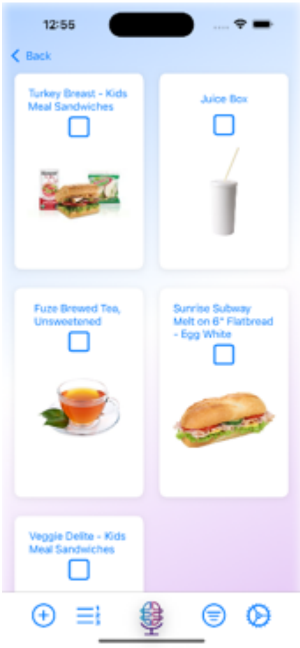
Appendix F. Mock-up of Application and API interaction

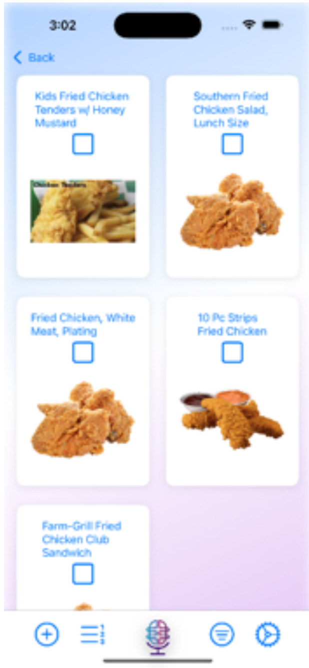


Appendix G. Snippet code of menu item fetching

```
// Fetch menu items for the restaurant (e.g., McDonald's)
            fetchMenuItems(forRestaurant: name) { result in
                DispatchQueue.main.async {
                    switch result {
                    case .success(let menuItems):
                        var delayTime: TimeInterval = 0.0
                        for (index, menuItem) in menuItems.enumerated() {
                            delayTime = TimeInterval(index) * 1 // Add a 1-second
delay between each item addition
                            DispatchQueue.main.asyncAfter(deadline: .now() +
delayTime) {
                                // Check if there's an image URL
                                if let imageURL = menuItem.image, let imageURLObject
= URL(string: imageURL) {
                                    // Asynchronously fetch image data and check for
404 error
                                    fetchImageData(from: imageURLObject) { imageData
in
                                        DispatchQueue.main.async {
                                            // If image data is nil (including 404),
proceed without an image
                                            self.addItemWithCategory(
                                                category: newCategory,
                                                title: menuItem.title,
                                                imageData: imageData
                                            )
                                        }
                                    }
```

Appendix H. Snapshot showcasing menu at Subway vs Nashville Chicken (5 item parameter input)





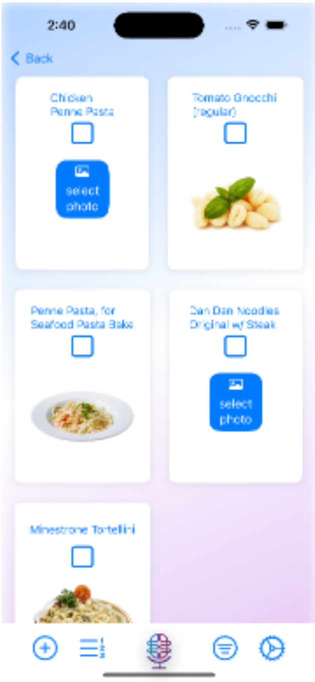
Appendix I. Snippet code of restaurant fetching and caching

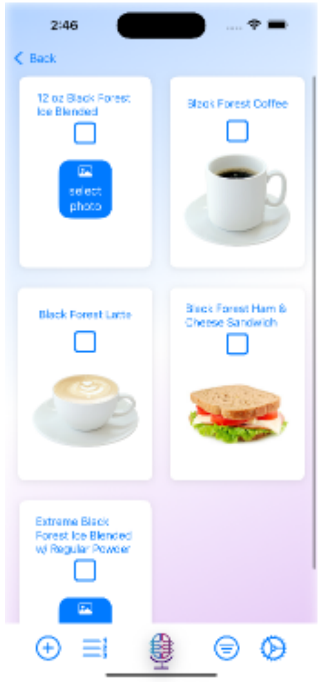
```
private func determineCategoryBasedOnLocation(currentLocation: CLLocation) {
            // Clear the list of pending categories
            pendingCategories.removeAll()

            for (restaurantName, locationData) in restaurantCache {
                let restaurantLocation = CLLocation(latitude: locationData.latitude,
longitude: locationData.longitude)
                let radius: CLLocationDistance = 100.0 // Radius in meters for
proximity

                // Calculate the distance between the user's current location and the
restaurant
                let distance = currentLocation.distance(from: restaurantLocation)

                if distance <= radius {
                    // Check if the category already exists to avoid duplicates
                    if !categories.contains(where: { $0.name == restaurantName }) {
                        print("Found nearby restaurant within radius:
\(restaurantName)")
                        pendingCategories.append(PendingCategory(
                            name: restaurantName,
                            latitude: locationData.latitude,
                            longitude: locationData.longitude,
                            logoPhotos: locationData.logoUrl
                        ))
                    } else {
                        print("Category \(restaurantName) already exists, skipping.")
                    }
                }
            }
```

Appendix J. Snippet code of Location Conversion to 6 decimal places

```
// Round latitude and longitude to 6 decimal places
let roundedLatitude = Double(round(restaurant.latitude * 1_000_000) / 1_000_000)
let roundedLongitude = Double(round(restaurant.longitude * 1_000_000) / 1_000_000)
```

Appendix K. Snapshot showcasing menus at Local North American Cafes (5 item parameter input)



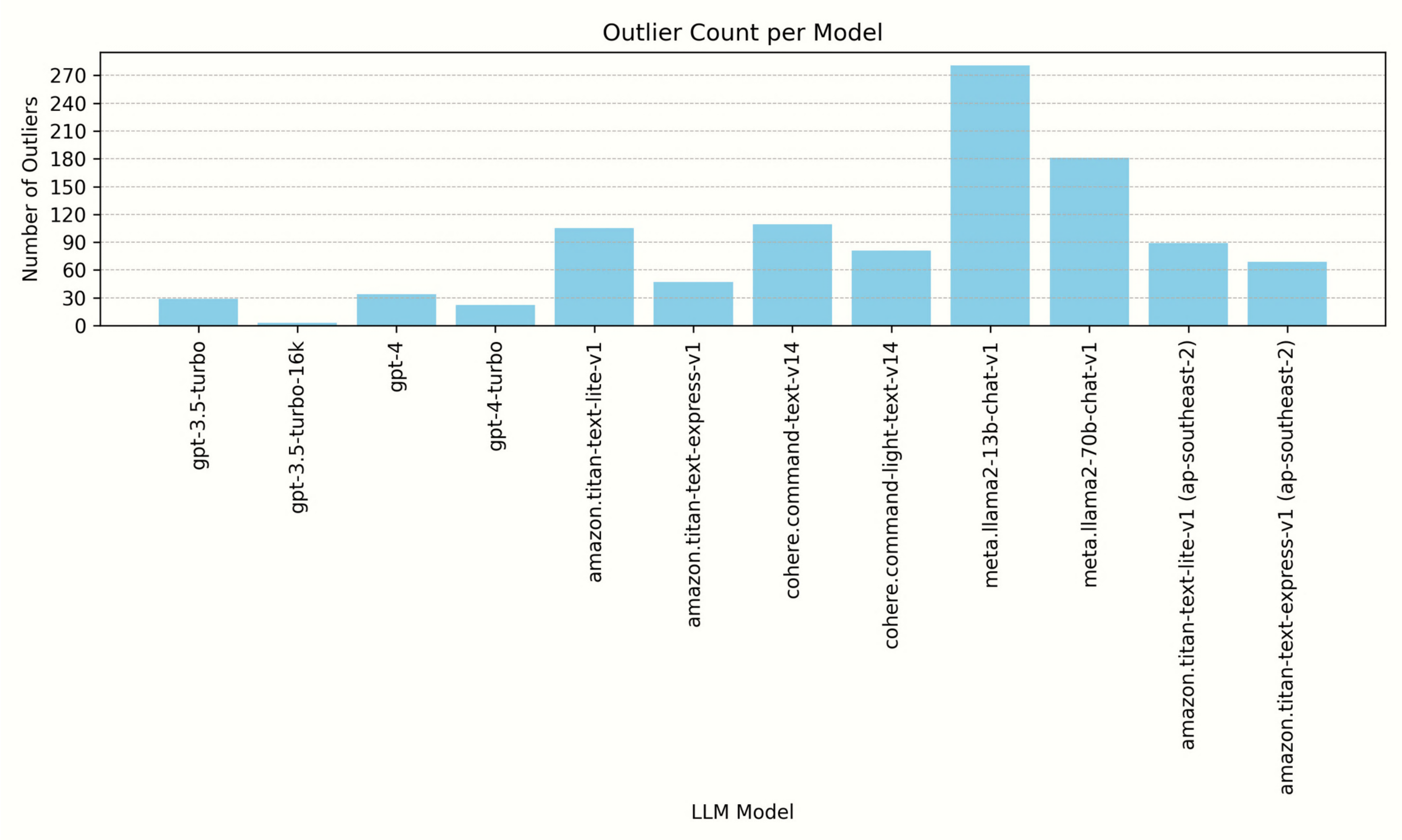


Appendix L. Query Code for Location-based Sorting

```
func getMostRecentResponses(limit: Int = 15) -> [String] {
        guard let location = LocationManager.shared.currentLocation else {
            print("Current location is not available.")
            return []
        }

        do {
            // Fetch the 15 most recent responses,
            let responsesQuery = responses
                .join(requests, on: responses[requestId] == requests[idExpression]
)
                .order(responses[responseId].desc)
                .limit(limit)

            let allResponses = try database?.prepare(responsesQuery)
            var wordsCount: [String: Int] = [:]

            for responseRow in allResponses ?? AnySequence([]) {
                let requestWords = responseRow[requests[wordsExpression]]
                let requestLongitude = responseRow[
                    requests[longitudeExpression]
                ]
                let requestLatitude = responseRow[
                    requests[latitudeExpression]
                ]

                // Calculate distance from current location to the request locatio
n
                let requestLocation = CLLocation(latitude: requestLatitude, longit
ude: requestLongitude)
                let distance = requestLocation.distance(to: location)

                if distance <= 500 { // within 500 meters
                    let words = requestWords.split(separator: ", ").map { String($
0) }

                    for word in words {
                        wordsCount[word, default: 0] += 1
                    }
                }
            }

            // Find the top 3 most used words
            let topWords = wordsCount.sorted(by: { $0.value > $1.value }).prefix(3
).map { $0.key }

            return Array(topWords)

        } catch {
            print("Query error: \(error)")
            return []
        }
    }

    func getRecentResponsesWithFallback() -> [String] {
        let primaryWords = getMostRecentResponses()

        if primaryWords.count >= 3 {
```

```
            return primaryWords
        }

        guard let targetTime = LocationManager.shared.currentDateTime else {
            print("Current date and time is not available.")
            return []
        }

        do {
            let oneHour: TimeInterval = 60 * 60
            let startTime = targetTime.addingTimeInterval(-oneHour)

            let fallbackQuery = responses
                .join(requests, on: responses[requestId] == requests[idExpression]
)
                .filter(requests[timeExpression] >= startTime)
                .order(responses[responseId].desc)

            let allFallbackResponses: AnySequence<Row> = try database?.prepare(fal
lbackQuery) ?? AnySequence([])

            var fallbackWordsCount: [String: Int] = [:] // Initialize words count
for fallback

            for responseRow in allFallbackResponses {
                let requestWords = responseRow[response] // Assuming you're fetchi
ng the response words here
                let words = requestWords.split(separator: ", ").map { String($0) }
                for word in words {
                    fallbackWordsCount[word, default: 0] += 1
                }
            }

            var wordsCount: [String: Int] = [:]

            for word in primaryWords {
                wordsCount[word, default: 0] += 1
            }

            for (word, count) in fallbackWordsCount {
                wordsCount[word, default: 0] += count
            }

            let topWords = wordsCount.sorted(by: { $0.value > $1.value }).prefix(3
).map { $0.key }

            return Array(topWords)

        } catch {
            print("Query error: \(error)")
            return []
        }
    }
```

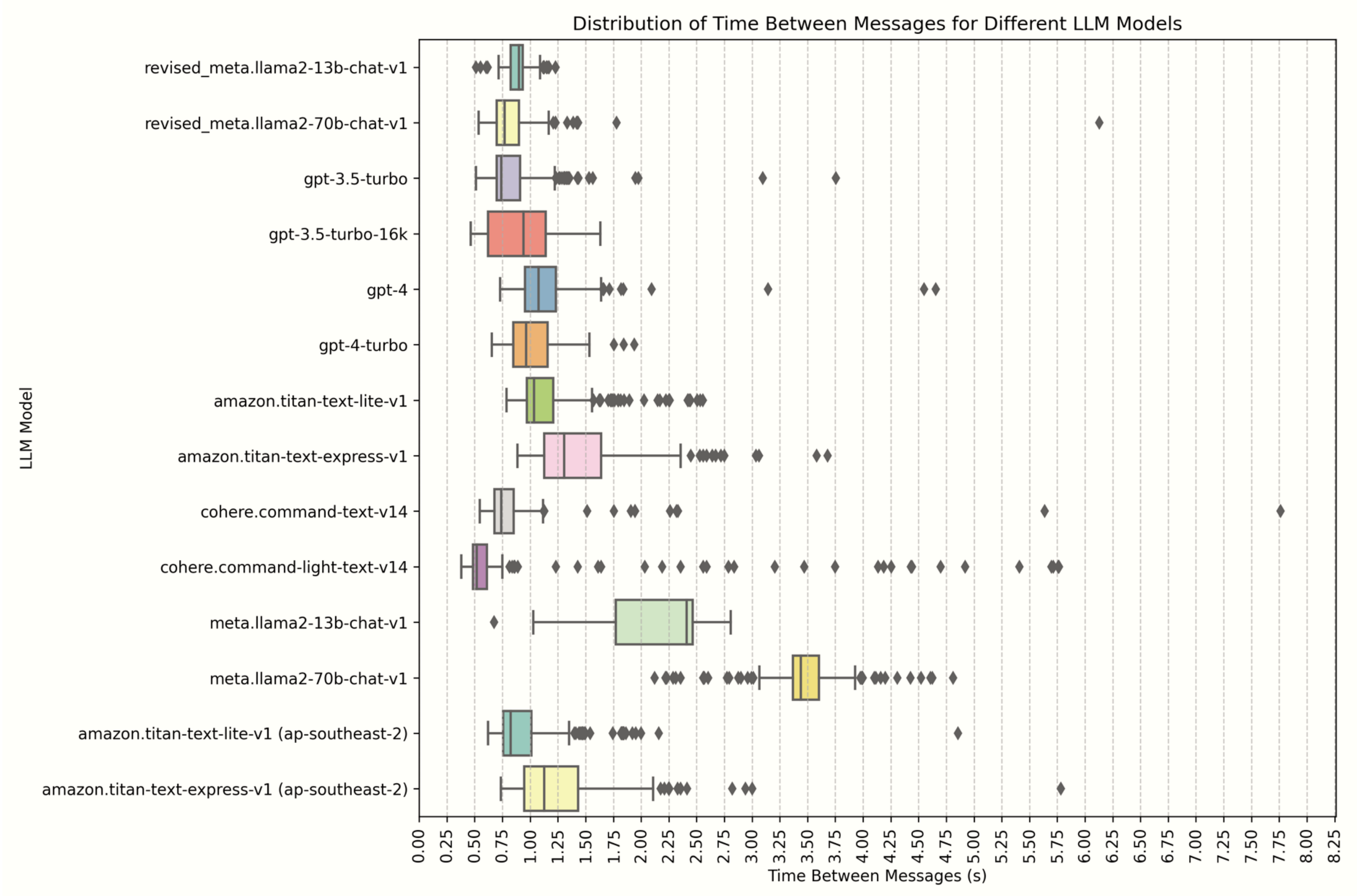
Appendix M. Outliers above the mean, for all models on all datasets



Appendix N. Example of Preface on Revised Prompt Meta 13b Model

message_response
Sure! Here is the sentence using the words degree and furniture:The couch requires a degree angle adjustment to fit properly in the living roo…
Sure! Here is the sentence using the words degree and furniture:The couch requires a degree angle adjustment to fit properly in the living roo…
Sure! Here is the sentence using the words degree and furniture:The couch requires a degree angle adjustment to fit properly in the living roo…
Sure! Here is the sentence using the words party and music:The upbeat music filled the party atmosphere.
Sure! Here is the sentence using the words party and music:The upbeat music filled the party atmosphere.
Sure! Here is the sentence using the words party and music:The upbeat music filled the party atmosphere.
Sure! Here is the sentence using the words exploration and cloud:The cloud-based platform enabled the team's exploration of new markets.
Sure! Here is the sentence using the words exploration and cloud:The cloud-based platform enabled the team's exploration of new markets.
Sure! Here is the sentence using the words exploration and cloud:The cloud-based platform enabled the team's exploration of new markets.
Sure! Here is the sentence using the words pet and jacket:My pet dog loves to wear his favorite jacket.
Sure! Here is the sentence using the words pet and jacket:My pet dog loves to wear his favorite jacket.
Sure! Here is the sentence using the words pet and jacket:My pet dog loves to wear his favorite jacket.
Sure! Here is the sentence using the words meeting and run:I need to run to the meeting.
Sure! Here is the sentence using the words meeting and run:I need to run to the meeting.
Sure! Here is the sentence using the words meeting and run:I need to run to the meeting.
Sure! Here is the sentence using the words study and peace:Seeking peace through study.
Sure! Here is the sentence using the words study and peace:Seeking peace through study.
Sure! Here is the sentence using the words study and peace:Seeking peace through study.
Sure! Here is the sentence using the words event and poem:The poem was the highlight of the event.

Appendix O. Boxplot of results including revised meta models



Contribution Declaration

Group member name	Contribution			
	Approach	Technical output	Writing	Project management
Matthew Fowler	Conceptualisation, Methodology, Formal analysis	Software, Investigation, Validation	Visualization, Writing (Draft and Review), Formal analysis	Project Administration
Keefe Zebastian Dela Cruz	Conceptualisation, Methodology, Formal analysis	Software, Investigation	Structure Conceptualisation, Writing (Draft and Review), Formal analysis	Project Administration
Addy Dhingra	Conceptualisation, Methodology, Formal analysis	Software, Investigation	Writing (Draft and Review), Formal analysis	Project Administration

 # Note: The group did not assign roles at any point due to the small size of the group

Group member name	**Comments (provide details about each contribution)**
Matthew Fowler	• Chapter 1. (Executive Summary) – Written and Reviewed • Section 3.1.2 (Multiple & Past Responses) – Written, Technical Work, and Reviewed. • Section 3.2.2 (Location Based Sorting) – Written, Technical Work, and Reviewed. • Section 2.3 ALL (Large Language Models) – Written, Technical Work, and Reviewed. • Section 4.1.2 (Multiple & Past Responses) - Written, Technical Work, and Reviewed. • Section 3.2.2 (Location Based Sorting) – Written, Technical Work, and Reviewed. • Section 3.3 ALL (Large Language Models) – Written, Technical Work, and Reviewed. Along with all visualization of data. • Section 4.3 ALL (Large Language Models) – Written and Reviewed • Chapter 5: (Suggestions for Future Work) Introduction & Section 5.2.4 (Shared Recommended Terms) & Section 5.3 (Large Language Models) – Written and Reviewed • Chapter 6: (Conclusion) – Written and Reviewed

	• Appendices A-B, & L-O along with formatting of all Appendices • Integration of all components into a complete package (Technical Verification). • Formatting and Layout upkeep of the document • Formatting and upkeep of document's referencing • Drafting of document (with Ms Dorothy Missingham)
Keefe Zebastian Dela Cruz	• Chapter 1:Introduction – Written and Reviewed • Section 2.2 (Location-based Systems) – Written and Reviewed • Section 3.2.1 (Location Aware Restaurants and Menu Items) – Written and Reviewed • Section 4.1.5 (Categories Ribbon) – Written and Reviewed • Section 4.2.1.1 (Location Aware) – Written and reviewed • Section 4.2.2 (Location Aware Restaurants) – Written and reviewed • Section 5.2.1 (Location-based Privacy Concerns)– Written and reviewedSnippet code of ScrollView for category ribbon • Appendix C (Snippet code of ScrollView for category ribbon) – Addition • Appendix D (Before and After snapshots demonstrating the category ribbon staying in place) – Addition • Appendix E (Snapshots of excessive restaurants in the dashboard) – Addition • Appendx F (Mock-up of Application and API interaction) – Addition • Appendix G (Snippet code of menu item fetching) – Addition • Appendix H (Snapshot showcasing menu at Subway vs Nashville Chicken (5 item parameter input)) – Addition • Appendix I (Snippet code of restaurant fetching and caching) – Addition • Appendix J (Snippet code of Location Conversion to 6 decimal places), – Addition • Appendix K (Snapshot showcasing menus at Local North American Cafes (5 item parameter input)), – Addition • Editing and Proofreading – Conducted comprehensive editing and proofreading of the document, refining grammar, enhancing clarity, and ensuring consistent structure, flow and readability throughout the entire report. • Supported in conceptualization of document content.
Addy Dhingra	• Section 2.1.1 (User-Interface Literature review) - Witten and Reviewed • Section 2.1.2 (User-Interface Relevance to Future Work) - Witten and Reviewed • Section 3.1.1 (User-Interface Approach) - Witten and Reviewed • Section 4.1 (User-Interface Outcomes) - Witten and Reviewed • Section 4.1.1 (User-Interface Evaluation of outcomes) - Witten and Reviewed

	• Section 5.1 (User-Interface Suggestions for Future Work) - Witten and Reviewed • Editing and Proofreading • Supported in conceptualization of document content.

Acknowledgement

By signing below students are acknowledging that the contributions recorded above are a true representation of the contribution made by everyone.

Name: Matthew Fowler	**Student ID:** a1803076	**Signature:**
Name: Keefe Zebastian	**Student ID:** a1793854	**Signature:**
Name: Addy Dhingra	**Student ID:** a1803893	**Signature:** suvir dhingra