Functional Specification

Project Title: MirAI Local Assistant

Project Supervisor: Gareth Jones
Date: 22nd November 2024

Group Members:

Name	Student Number
Georgijs Pitkevics	19355266
Chee Hin Choa	21100497

1. Introduction 1
2. General Description 2
3. Functional Requirements 3
3.1 User Interaction 3
3.1.2 Mobile Interface 4
3.1.2 Admin Panel: API Module Configuration 5
3.1.3 Admin Panel: LLM Performance 5
3.1.4 Admin Panel: Chat Now 6
3.2 NLP Module 7
3.3 RAG 7
3.4 Data Privacy and Security 8
3.5 LLM 8
3.5.1 Technical Difficulties: Hallucinations 9
3.5.2 Technical Difficulties: Information validity 9
3.5.3 Technical Difficulties: Limited Reasoning Skills 9
3.5.3 Technical Difficulties: Hardware Limitations 10
3.6 Additional Features 10
3.6.1 Custom Voice Line 10
3.6.2 Custom personality prompt 10
3.6.3 Various LLM Model Options 10
3.6.4 Custom API Modules 11
4. System Architecture 11
4.1 Backend 11
4.2 Frontend 11
4.3 Database 12
4.4 LLM & TTS/STT 12
5. High-Level Design 12
5.1 Data flow diagram 13
5.1.1 General LLM Query 13
5.1.2 Question/Trivia LLM Query 13
5.1.3 API Module Query 14
5.2 Object Model 15
6. Preliminary Schedule 16
7. References 16

1. Introduction

1.1 **Overview**

MirAI is a local hosted smart assistant developed to provide an alternative option for privacy concerns associated with cloud-based assistants, such as Google Home, or Amazon Alexa, by running all data processing within the user’s local environment.

MirAI is designed to be extensible by allowing community-driven customization and interaction with user specific services. This bankend can be run on any UNIX/Windows based hardware for stronger processing power, while the frontend can be run on any device that supports a modern browser. This provides a great opportunity to give old devices a new chance to shine while not having to worry about processing power, this leaves a lot of flexibility for the user facilitating a multi-platform experience.

1.2 Business Context

MirAI targets privacy conscious individuals, DIY enthusiasts and technology developers who seek full control over their personal assistant. The project aligns with the increasing demand for GDPR compliant self-hosted solutions for both personal and business settings. It focuses on open-source development to build community contributions and customizability.

1.3 **Glossary**

- **LLM (Large Language Model):** AI models trained to understand and generate responses.
- **TTS (Text-to-Speech):** Converts text input to voice spoken output.
- **STT (Speech-to-Text):** Converts voice input into text.
- **API:** Application Programming Interface, a set of functions allowing interaction between applications.
- **[the] System:** The project as a whole.
- **PromptBuilder:** An internal python class that constructs the final prompt prior to sending the prompt to the LLM.
- **NLP (Natural Language Processing):** Responsible for any natural language related operations, like understanding user input to determine the task.
- **RAG (Retrieval-Augmented Generation):** Enhance LLM generation capabilities by recalling information from an up-to-date source.
- **WWW:** World Wide Web

2. General Description

2.1 Product / System Functions

MirAI provides a local hosted assistant capable of:

- Speech recognition and voice response generation using STT and TTS.
- Handling both predefined and natural language commands.
- Holds memory of conversations for better response and awareness.
- Integrating with custom APIs, enhancing its functions through a custom API gallery.

2.2 User Characteristics and Objectives

Users are expected to be technically adept with an interest in privacy and DIY technology. Users who seek the same equivalent functionality of popular cloud based assistants but with all processing happening locally. They aim to control smart devices, manage schedules and automate tasks without sharing data externally.

2.3 Operational Scenarios

Here are some examples of what the user could use MirAI for:

1. Home Automation:

   Users can command, “MirAI, good morning” which initiates a series of tasks like turning on lights, reporting weather and playing morning news.

2. Information Retrieval:

   A user asks, “What is today's weather like and should I wear a jacket?” MirAI will respond using weather information integrating with personal insights.

3. Query:

   Users could ask, “When was Barack Obama born?”. MirAI would answer “August 4, 1961” while maintaining information validity using RAG.

2.4 Constraints

1. Hardware:
   A device or server with sufficient processing power is necessary for low latency responses.

2. Software Compatibility:
   Only supports systems that can host Dockerized applications and Python-based applications.

3. Performance:
   The degree of LLMs that can be run depends on local hardware performance.

3. Functional Requirements

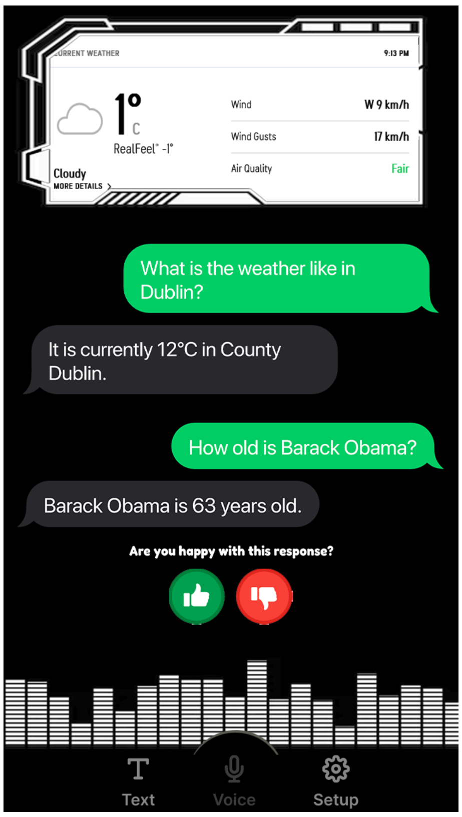
3.1 User Interaction

- **Description:**
  The system provides two interfaces:
  1. **Frontend Interface:**
     - Browser-based, as shown in figure 1
     - Allows users to interact with MirAI via speech or text input.
     - Provides feedback options (positive or negative) for LLM tuning.

2. **Admin Panel:**
    - As shown in figure 2, 3 and 4.
    - Includes pages for API module configuration, LLM performance tuning and advanced testing via the “Chat Now” page.

- **Criticality**: High
- **Technical Issues**:
    - Ensuring low-latency communication between the frontend and backend for real-time interactions.
    - Compatible for different browsers.
- **Dependencies**:
    - Local STT for speech to text conversion
    - Backend API for processing user queries.

3.1.2 Mobile Interface



What is the weather like in Dublin?

It is currently 12°C in County Dublin.

How old is Barack Obama?

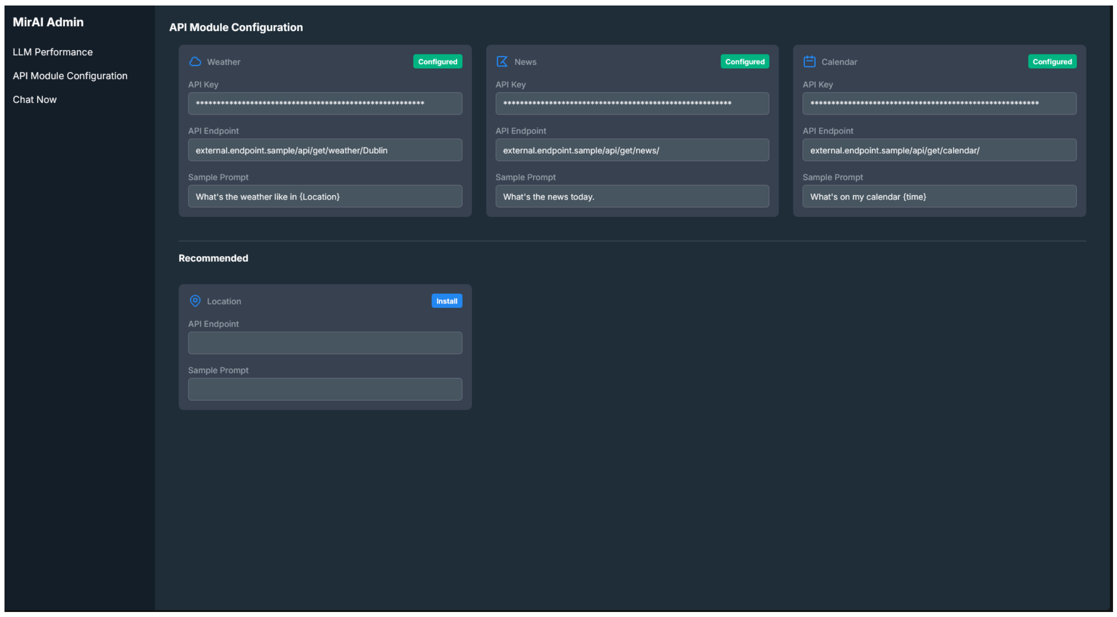
Barack Obama is 63 years old.

Are you happy with this response?

Figure 1

The mobile interface (figure 1) will consist of the speech detection and response from the assistant. The user will be able to initialise the conversation either from voice or simply typing into the chat. At the bottom of each conversation will display negative and positive feedback buttons to further tune the LLM’s responses. For further configuration, the user would need to go into the “Setup” on the bottom right, which will lead them to a simplified mobile view of the the admin panel.

3.1.2 Admin Panel: API Module Configuration



API Module Configuration

Weather
API Key: ••
API Endpoint: external.endpoint.sample/api/get/weather/Dublin
Sample Prompt: What's the weather like in {Location}

News
API Key: ••
API Endpoint: external.endpoint.sample/api/get/news/
Sample Prompt: What's the news today.

Calendar
API Key: ••
API Endpoint: external.endpoint.sample/api/get/calendar/
Sample Prompt: What's on my calendar {time}

Recommended

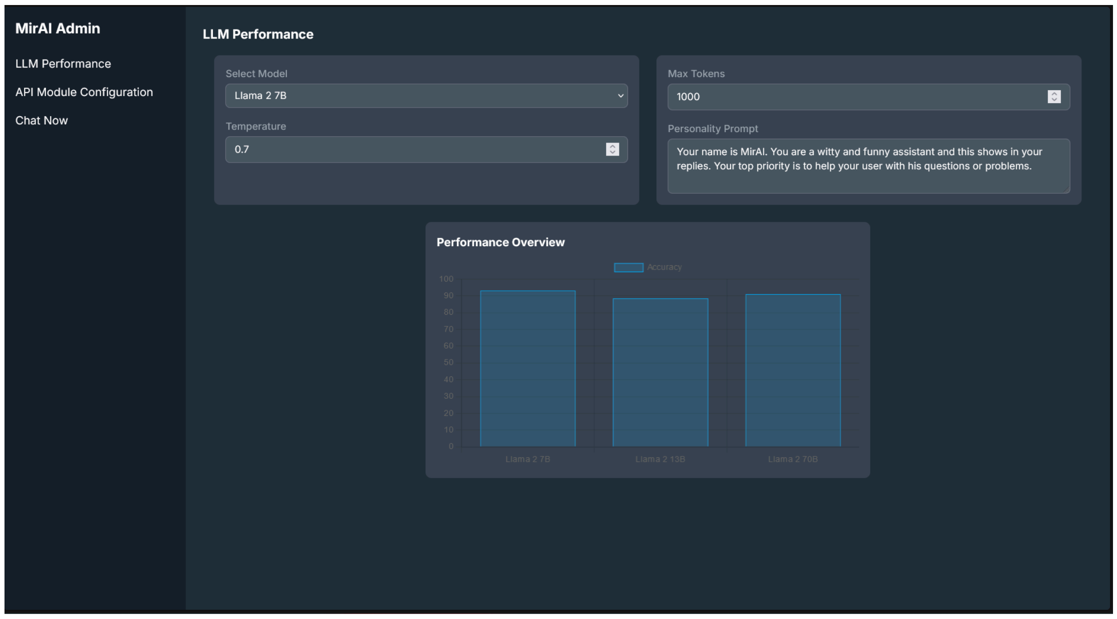
Location
API Endpoint:
Sample Prompt:

Figure 2

Within the admin page taskbar (figure 2), users can navigate between the LLM performance page, API module configuration page or back to the Chat Now main page.

Within the API module configuration, there will be a pre-populated API for different features such as weather, news or calendar. The user will be able to swap out the API to their desired endpoint for a more personalised result.

3.1.3 Admin Panel: LLM Performance



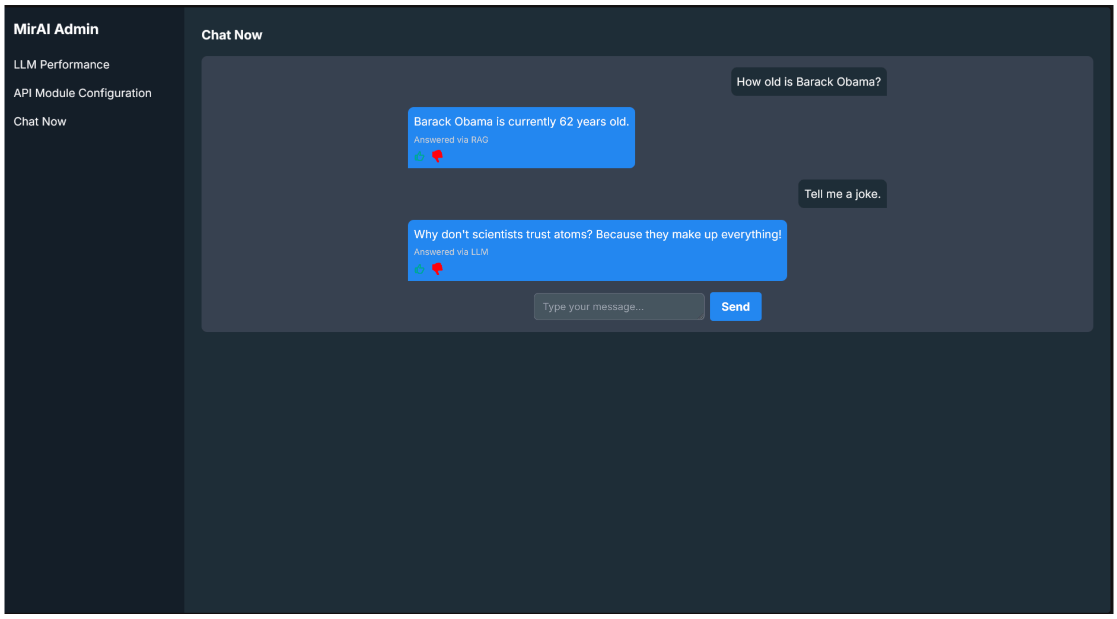
LLM Performance

Select Model: Llama 2 7B
Temperature: 0.7
Max Tokens: 1000
Personality Prompt: Your name is MirAI. You are a witty and funny assistant and this shows in your replies. Your top priority is to help your user with his questions or problems.

Figure 3

In the LLM Performance page (figure 3), users are given options to tuning the LLM model to desired, such as swapping out for a LLM model of choice, setting different temperature for preferable response, maximum token input, and even customised personality prompts for the model. Below we will have a graph for all the included models and performance based on previous user feedback.

3.1.4 Admin Panel: Chat Now



How old is Barack Obama?

Barack Obama is currently 62 years old.
Answered via RAG

Tell me a joke.

Why don't scientists trust atoms? Because they make up everything!
Answered via LLM

Figure 4

Finally the Chat Now page (figure 4) is for user interaction and testing of the user's configuration. It will be functionally similar to the client side UI, along with providing additional information about the response, for example; if it was achieved using RAG or an LLM below each response from MirAI.

Note: Designs are not final, they only serve as prototypes to visualise what the end result is expected to look/function like.

3.2 NLP Module

The NLP Module handles the query classification and determines which query type will be executed.

Regular expressions will be used to detect predefined API Module functions ie; commands like "turn on the light" or "set light brightness to 50%", the simplified regex for this can looking something like so: (turn on|set brightness to) (the)?(light|\w+ light)

To distinguish between a ***General LLM Query*** and ***Trivia LLM Query***, a simplified implementation of the spaCy python library will be used to enhance the classification accuracy. This is done by detecting nouns of importance, ie *"Who is Barack Obama?"*. In this example, the ***NLP Module*** will detect that *"Barack Obama"* is a noun (person) of importance in the sentence, which will trigger the ***Trivia LLM***

Query data flow which will utilise an **RAG** methodology to retrieve up-to-date information.

Criticality: High

Technical Issues:
- Balancing accuracy and processing speed for best suited classification.

Dependencies:
- spaCy library for natural language processing.
- RAG module for enhanced information retrieval.
- API Module for predefined commands.

3.3 RAG

RAG (Retrieval-Augmented Generation) is used in order to optimise the LLM's knowledge base by providing a relevant article(s) from an external search engine, like Google, or Bing.

- **RAG** is important because we want to prevent the LLM from hallucinating and/or giving outdated information.
- If a model was trained in 2020, and the user were to ask it "How old is Barack Obama", the LLM's outdated knowledge base may provide an inaccurate value from 2020.

RAG is a crucial factor for improving the user experience, as it will be able to provide up to date information without relying on the LLM's knowledge base that it was previously trained on.

We acknowledge that the RAG functionality contradicts the Data Privacy specifications, therefore we are going to have it turned off by default and allow the user to turn this feature on or off.

Criticality: Medium
RAG is essential for addressing the limitations of LLMs but is an optional feature due to privacy considerations.

Technical Issues:
- Managing the trade off between data retrieval speed and result quality.

Dependencies:
- External search engines for retrieval tasks, such as Google, Firefox.
- Backend for query augmentation before LLM processing.

3.4 Data Privacy and Security

The project's key goal is to create an alternative home assistant which is self-hosted and allows the user to have complete control over their data. This is done by allowing the user to host the LLM, TTS and STT models locally with ease. Any interaction that the user has with the system is performed within the user's local network and no external sources are accessed, unless explicitly allowed by the user, for example when allowing API Modules to access external API endpoints.

This is critical to the System and the user's privacy. All logs are stored in a database on the user's machine and can only be accessed by the user. No external LLM endpoints like OpenAI or Gemini are being accessed to perform the LLM functionality.

Criticality: High
Data privacy is a core principle of this system, separating it from conventional cloud-based assistants.

Technical Issues:

- Implementing secure encryption for sensitive user data (e.g, API keys).
- Ensuring no unintended external communications occur.

Dependencies:

- Database for local data storage.

3.5 LLM

- An ***LLM*** is used in order to handle Natural Language Processing within the System, as well as provide reasoning ability and allow MirAI to respond in a human-like manner.
- Conventional home/personal assistants, do not have the ability to reason as they rely on if-else based logic and "hardcoded" features, which do not allow the user to add their own functionality or ask "thinking" questions, which can be open to reasoning or decision making.

Our LLM integration will consist of an internal API integration layer using ***Flask*** for API functionality and the ***LangChain*** python library for handling prompt building, LLM pipelines and conversation history.

This functionality is vital, as it allows the user to interact with MirAI in a conversational manner that cannot be done with current conventional home assistants.

There are many technical issues associated with LLMs that we need to tackle in order to achieve the ideal user-system interaction.

Some of which are:

- **Hallucinations:** No matter how complex or intelligent a model is, it will always hallucinate.
- **Information validity:** The model's knowledge base is limited, and could be inaccurate depending on the age of the data.
- **Limited Reasoning Skills:** LLMs struggle with multi-step maths problems, or puzzles as they can get confused or hallucinate throughout the process.

- **Hardware Limitations:** Often LLMs require very expensive and powerful hardware to run.

3.5.1 Technical Difficulties: Hallucinations

To prevent hallucinations, we will test and configure suitable temperature values for each of the models. Additionally, we will test different prompts to determine which work best per each model to decrease the rate and possibility of hallucinations.

For testing, we will incorporate a manual testing procedure, where we gather a number of prompts and LLM configurations. Using an external LLM like Gemini or ChatGPT we will generate a dataset of questions to include in the prompt, these will be reviewed and sorted manually. We will then perform A/B testing and to compare and rank different combinations of prompts and configurations, and how they affect the LLM's response.

3.5.2 Technical Difficulties: Information validity

A side-effect of hallucinations and/or a limited knowledge base is information invalidity. By implementing an RAG methodology, we can circumvent or lower the possibility of the LLM directly hallucinating facts and verify the validity of the information by acquiring it from up-to-date sources using search engines like Google Search, or Bing.

A technical drawback of this, is we are unable to verify the validity of external sources like Google Search, or Bing as they can also be doctored or incorrect. This is something that we will monitor during our LLM testing phase.

3.5.3 Technical Difficulties: Limited Reasoning Skills

LLMs are exceptionally good at understanding scenarios, generating natural language response and solving minor problems.

A similar approach to the solution to hallucinations is to be implemented to overcome/contain the Limited Reasoning Skills.

Within our prompt testing, we will identify and adhere to the prompt to add a disclaimer within the system's response, in cases where the LLM attempts to solve a complex problem (ie; maths), the system will provide a disclaimer to the user that the solution may not be correct.

3.5.3 Technical Difficulties: Hardware Limitations

By implementing RAG, we are offloading some of the more complex tasks like fact-checking and the requirement of an up-to-date knowledge base away from the LLM. This gives us the opportunity to test multiple different models and determine a minimum specification required to run the corresponding models.

This gives the user the ability to:

- Configure their hardware to the specifications pertinent to the model of their choosing.

  OR

- Choose a model that corresponds to their current hardware specifications.

3.6 Additional Features

Here are some additional features that will be implemented.

3.6.1 Custom Voice Line

- **Description**: User would be able to swap out the assistant's default voice line for another voice line. Users could pick between a few options or even choose to upload their own desired voice line.
- **Criticality**: Low
- **Technical Issues**: Expand current TTS model compatibility with different voiceline files and clearly isolated out for easy transition to other TTS models.
- **Dependencies**: depending on TTS model compatibility

3.6.2 Custom personality prompt

- **Description**: Users have the freedom to customise their assistant personality, not just the LLM response but also affecting TTS voice personality, such as a cheerful response, a grumpy reply or even a sad emotion.
- **Criticality**: Medium
- Technical Issues: giving personality to LLM is not a big issue but having a TTS model to detect emotion and generate a voice with emotion could lower the accuracy and slower processing time.
- **Dependencies**: LLM model and TTS model performance

3.6.3 Various LLM Model Options

- **Description**: Users can change the current LLM model for another included model or external model seamlessly. This would give MirAI a lot of flexibility to adapt on different devices with various processing levels and user's needs.
- **Criticality**: High
- **Technical Issues**: Swapping models or having multiple models could take up time and disk space
- **Dependencies**: Hardware

3.6.4 Custom API Modules

- **Description**: We will allow the user to add custom API Modules, either created by the user, or recommended modules supplied with the system. This allows MirAI to be more helpful as a personal assistant and customizable to the user's needs.
- **Criticality**: High
- **Technical Issues**: Exploring way to get data and interact with different public API endpoints
- **Dependencies**: API models

4. System Architecture

The system architecture involves a client-server setup outlined as follows:

4.1 Backend

Flask-based API server for NLP processing and API integration.

Each part of the backend will be broken down into different classes/objects as expressed in Figure 6.

LangChain[1] and its other libraries will be used within the PromptBuilder, they will handle the conversation memory, temperature and other flags, as well as the piping of the prompt and conversation history to the LLM.

The backend will be used as the core processing engine for MirAI. It will handle all of the query logic associated with the different types of queries outlined in section 5.1.

4.2 Frontend

The frontend serves as the user interface where the user can interact with MirAI via both text and voice. It will also be running the STT component, handling speech-to-text locally to minimise latency.

Technology:

- ReactJS, for a dynamic interface. It also supports cross platform compatibility allowing it to be run on Android, desktop or any modern browser.

Modules:

- **Chat Interface:**
  - Shows user queries and system responses in a conversational manner.
  - Provide positive or negative feedback buttons for user input on response satisfactory.
- **Performance Monitoring Page:**
  - Allows tuning for LLM settings such as models, temperature and token limits.
  - Display graphs for system performance based on user’s feedback
- **API Configuration Page:**
  - Showing available API endpoints and hidden API keys
  - Allows adding new API endpoints and activate key words
- **Audio Output:**
  - Receives audio from the backend via TTS and plays it back for the user.

4.3 Database

The database will serve as the central storage system for MirAI, it will persist data essential for the operation and customization. It is designed to focus on performance, data privacy and scalability.

Technology: MySQL or SQLite for lightweight deployments.

Data Storage:

- Stores user preferences, logs, and conversation history.
- API configuration details, for example API keys and endpoint
- Records TTS/STT processing metadata for performance analysis.

Data Privacy:

- All data is stored locally on the user's system to comply with the project privacy focused goals.
- Only allowed external interactions with other cloud/WWW based services such as RAG queries to store limited data, like query logs.
- Encrypted storage for sensitive information such as API keys and passwords.

4.4 LLM & TTS/STT

Integrated via oobabooga/text-generation-webui, Langchain, and Coqui TTS libraries.

For the STT model, we will test different methods of doing this like whisper.tflite[2] and react-speech-recognition[3]. This allows us to test the different models such as whisper, for best performance and the ability to further fine tuning to suit our needs.

For streaming the TTS audio we will test and research different methods such as KoljaB/RealtimeTTS[4] paired with a Coqui. We will process the TTS model on the server-side for maximum performance and reduced latency that can occur with running TTS models on low-power devices. Additionally, this will allow for testing of different advanced TTS models and determining which can closely resemble a human voice.

5. High-Level Design

- **Command Processor**: Detects commands and routes them.
- **API Integration Layer**: Manages API calls based on user-activated modules.
- **Speech Module**: Handles TTS/STT functions.
- **User Interface Module**: A ReactJS front end for user interactions.

5.1 Data flow diagram

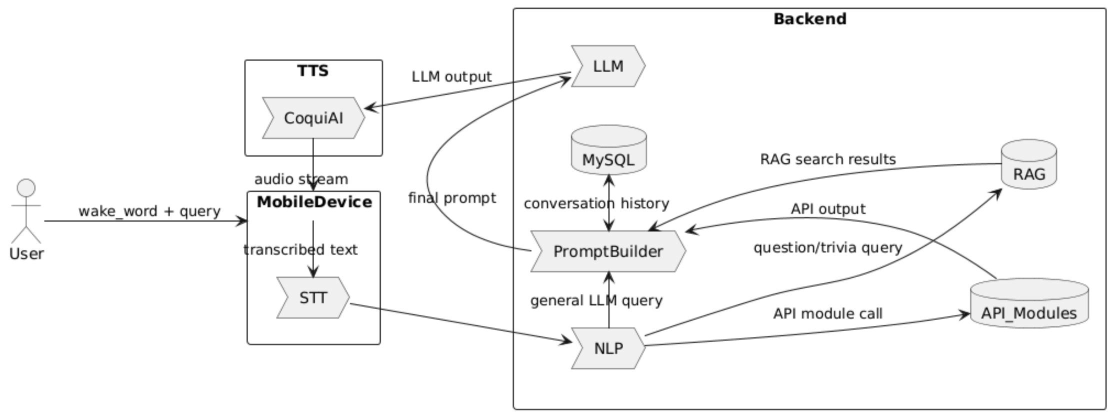


Figure 5.

- Figure 5 describes the three different data flow variants that can occur when the user interacts with the System.
- Throughout all the data flows, they all start with the user's query via their voice which takes the format of wake word + query.

The wake word is predefined and the same in every interaction. The data flow between the ***user***, ***MobileDevice***, and ***TTS*** will not differ. A modified conversation history is given to the PromptBuilder with every System interaction to provide the LLM context towards the user's previous queries.

The data flow within the backend between the NLP Module and LLM differs depending on the ***query type***, the ***query type*** is defined by three different variations; ***General LLM Query***, ***Trivia LLM Query***, and ***API Module Query***.

The ***NLP Module*** will define the query type based on the intent classification via ***Regular Expressions*** and ***NLP*** methodologies using the ***spaCy*** python library.

5.1.1 General LLM Query

- Example; wake work + "Tell me a joke".
- This general query data flow is the simplest, where the query does not require any external resources to be resolved.
- The query is given directly to the ***PromptBuilder*** and then passed to the ***LLM.***

5.1.2 Question/Trivia LLM Query

- Example; wake work + "How old is Barack Obama".
- This ***Trivia LLM Query*** data flow uses ***NLP*** to determine key nouns, which will trigger ***RAG*** methodology to provide additional information about the query.
- The original query and ***RAG*** output is then passed to the ***PromptBuilder*** and after passed to the ***LLM.***

5.1.3 API Module Query

- Example; wake work + "What's the weather like in Dublin".
- The ***API Module Query*** data flow uses ***Regular Expressions*** to determine the API Module type, in this case it is the Weather API Module.
- The external API call is resolved and the output of the API Call, as well as the original query, is passed to the ***PromptBuilder*** and then to the ***LLM***.

5.2 Object Model

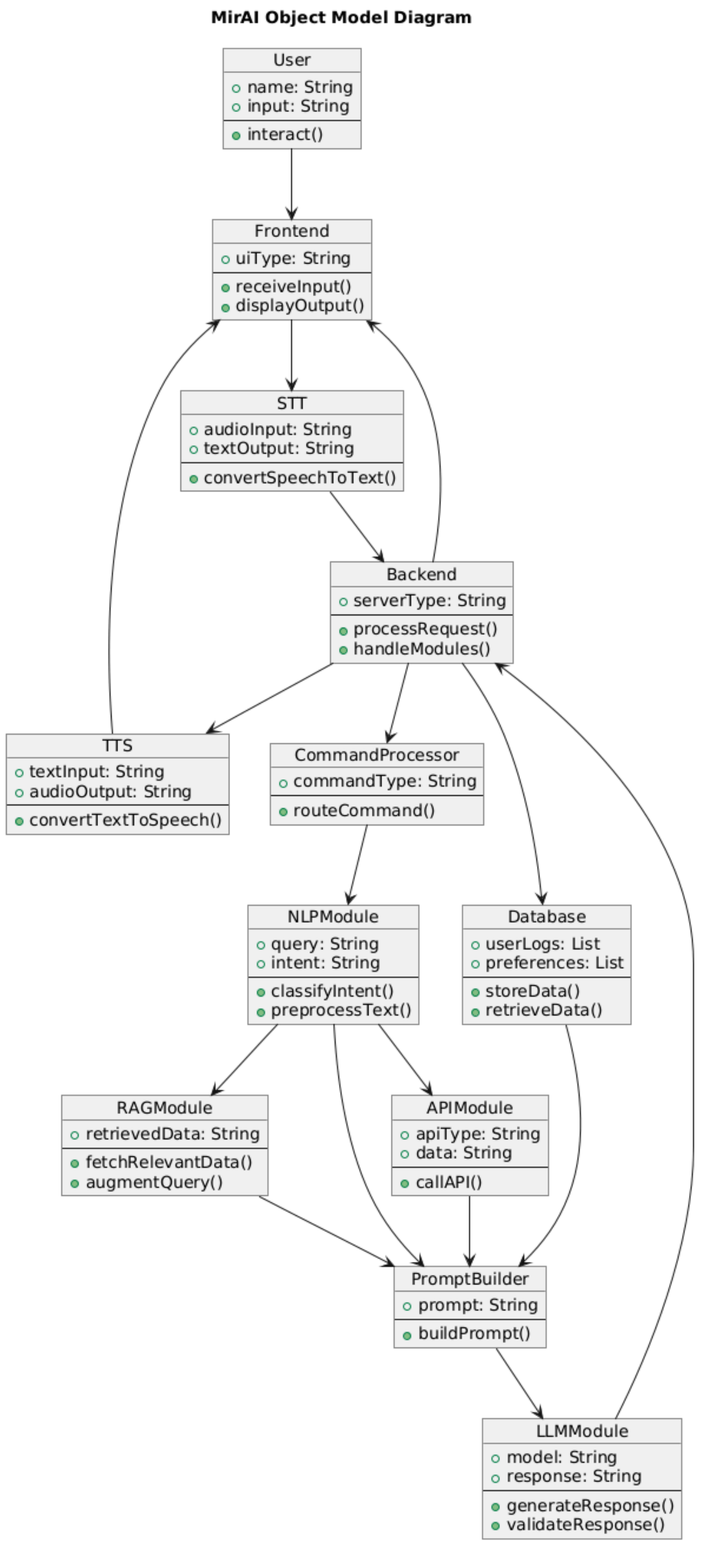
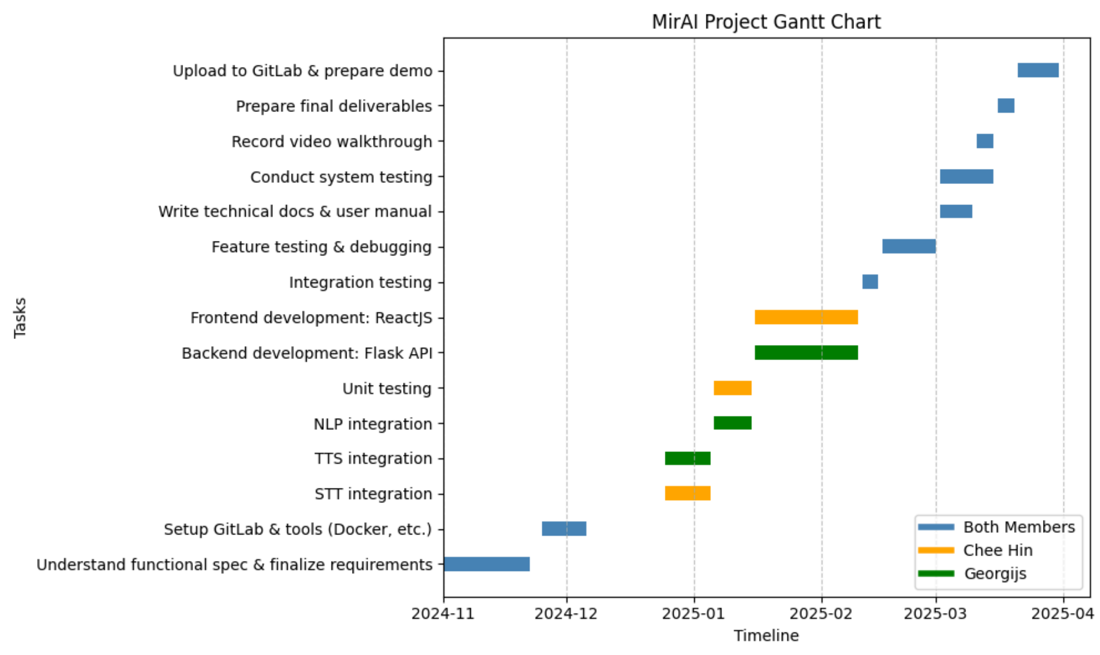


Figure 6

The object model in figure six is a visual representation of the key components of the MirAI system and the interactions between each class/component.

6. Preliminary Schedule



This gantt table shows our initial version of the project plan from now until the end of March as that may be the deadline for submission. This plan showcases the major tasks needed to be completed, team workload balance and their tentative schedule.

- Due to the exam period, there will be a break between 06 December until 24 December.
- We will be focusing on our own study during that break.
- Our goal is to be able to prepare everything by the end of March for final submission.

For any tasks that we are working on independently, highlighted in green and yellow, we will incorporate a pair programming approach with regular meetings to ensure that we understand how each member's code interacts with the other's.

For project, task and deadline planning we have implemented an Agile and Scrum based workflow.

For version control we will utilise git flow[5] which will allow each member to keep track of their work without interfering with the other's. With peer reviewed pull requests, we promote collaboration and a common understanding of each other's new or changed code.

7. References

[1] https://www.langchain.com/: LangChain python library
[2] https://github.com/nyadla-sys/whisper.tflite: Whisper Enhanced Quantized TFLite Model
[3] www.npmjs.com/package/react-speech-recognition: react-speech-recognition
[4] https://github.com/KoljaB/RealtimeTTS: KoljaB/RealtimeTTS

[5] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow:
git flow