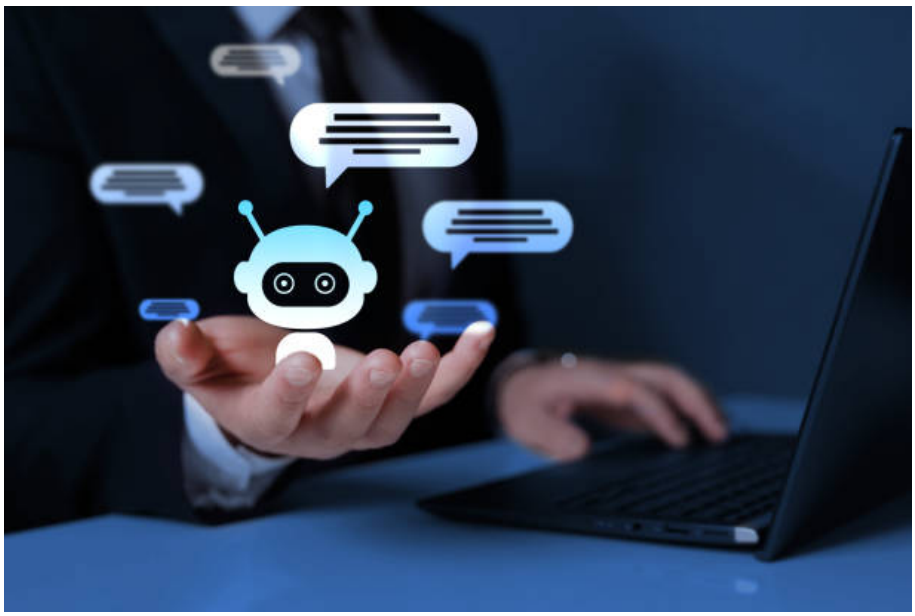

Digital Assistant with Artificial Intelligence Techniques

DINIS RODRIGUES DIAS
(Licenciado)

Dissertação para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientadores:
Doutor Artur Jorge Ferreira
Doutor Nuno Miguel da Costa de Sousa Leite

Júri:
Presidente: Doutor Nuno Miguel Machado Cruz
Vogais:
Doutor Paulo Manuel Trigo Cândido da Silva
Doutor Nuno Miguel da Costa de Sousa Leite

Outubro de 2025

Digital Assistant with Artificial Intelligence Techniques

DINIS RODRIGUES DIAS
(Licenciado)

Dissertação para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientadores:
Doutor Artur Jorge Ferreira, IPL/ISEL
Doutor Nuno Miguel da Costa de Sousa Leite, IPL/ISEL

Júri:
Presidente: Doutor Nuno Miguel Machado Cruz, IPL/ISEL
Vogais:
Doutor Paulo Manuel Trigo Cândido da Silva, IPL/ISEL
Doutor Nuno Miguel da Costa de Sousa Leite, IPL/ISEL

Outubro de 2025

Acknowledgements

I would like to express my deepest gratitude to my advisers, Doctor Artur Ferreira and Doctor Nuno Leite, for their constant guidance, availability, and encouragement throughout this work. Their support was not only essential for the development of this thesis, but also instrumental in giving me the opportunity to attend conferences and present the articles that emerged from it.

I am also thankful to my colleague António Goulão, with whom I shared many discussions and ideas. Our works had strong parallels, and his collaboration and co-authorship of the related articles enriched both the research and the personal experience.

To my family, I owe the greatest thanks for making it possible for me to pursue higher education and for their unwavering support at every stage of this journey. To my girlfriend, I am profoundly grateful for her encouragement, patience, and care during the thesis process, I love her deeply.

Finally, I would like to thank my friends, as well as my family and my girlfriend once more, for the enthusiasm and belief they always showed in me and in this project. Their confidence was a constant source of motivation.

Statement of integrity

I declare that this dissertation is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author

Assinado por: Dinis Rodrigues Dias
Num. de Identificação: BI30244003
Data: 27-11-2025 01:27:40 +00:00

CHAVE MÓVEL

Abstract

The design, implementation, and assessment of a modular Digital Assistant (DA), developed in Python, that can process natural language in speech and text, being optimized for the Windows desktop environment are presented in this dissertation. The DA performs tasks like retrieving weather data and launching applications, where the system combines Large Language Models (LLM) to interpret user requests and dynamically choose between conversational responses and function execution. To ensure modularity, extensibility, and maintainability, a layered architecture was used to organize the functionality, reasoning engine, conversation handling, and graphical user interface modules. To maintain responsiveness and user control even during lengthy operations, the assistant uses asynchronous execution, supports both text and voice input, and can output speech synthesis.

The implementation places a strong emphasis on sound software engineering techniques, such as modular contracts, interface-first design, and reliable error handling. The secure handling of Application Programming Interface (API) keys and the lack of persistent memory protect privacy are also addressed. Experimental evaluation shows near real-time responses from contemporary LLM backends, sub-second latency for functionality modules, and high accuracy in differentiating between function calls and conversations. Additionally, qualitative validation verifies that the system satisfies its non-functional requirements for modularity, robustness, and user experience, and that the Graphical User Interface (GUI) is responsive and the speech features are usable.

In conclusion, the project produces a useful, expandable, and intuitive digital assistant that connects conversational Artificial Intelligence (AI) and desktop task automation, providing a solid basis for upcoming improvements like cross-platform deployment, sophisticated speech recognition, and runtime model selection.

Keywords

Digital Assistant, External Function Calling, Large Language Models, Natural Language Processing

Resumo

Esta dissertação apresenta o projeto, implementação e avaliação de um Assistente Digital (AD) modular, desenvolvido em Python e otimizado para o ambiente Windows, capaz de processar linguagem natural em texto e voz. O sistema realiza tarefas como a obtenção de dados meteorológicos e o lançamento de aplicações, combinando Modelos de Linguagem de Grande Escala (LLM) para interpretar pedidos do utilizador e escolher dinamicamente entre respostas conversacionais e a execução de funcionalidades. Para assegurar modularidade, extensibilidade e facilidade de manutenção, foi adotada uma arquitetura em camadas que organiza os módulos de funcionalidades, motor de raciocínio, tratamento de conversação e interface gráfica. Para manter a responsividade e o controlo do utilizador mesmo durante operações prolongadas, o assistente utiliza execução assíncrona, suporta entrada por texto e voz e gera saída através de síntese de fala.

A implementação dá particular ênfase a boas práticas de engenharia de software, incluindo contratos modulares, design orientado a interfaces e tratamento robusto de erros. A privacidade é salvaguardada através do manuseamento seguro de chaves de Interfaces de Programação de Aplicações (API) e da ausência de memória persistente. A avaliação experimental demonstra respostas quase em tempo real dos LLM modernos, latência inferior a um segundo nos módulos de funcionalidades e elevada precisão na distinção entre conversação e chamadas de função. A validação qualitativa confirma que o sistema cumpre os requisitos não funcionais de modularidade, robustez e experiência do utilizador, verificando-se ainda que a interface gráfica é responsiva e as funcionalidades de voz são utilizáveis.

Em suma, o projeto apresenta um assistente digital útil, expansível e intuitivo, que alia a inteligência artificial conversacional à automação de tarefas em ambiente desktop, constituindo uma base sólida para futuras melhorias, como a portabilidade entre plataformas, reconhecimento de fala avançado e seleção dinâmica de modelos.

Palavras-chave

Assistente Digital, Chamada Externa de Funções, Modelos de Linguagem de Grande Escala, Processamento de Linguagem Natural

Contents

List of Figures

List of Tables

Glossary

Digital Assistant	A software agent that can perform a range of tasks or services for a user based on user input such as commands or questions, including verbal ones. Can be used as an umbrella term for Virtual Assistant, Conversational Agent or chatbot.
Rogers' therapy	A Person-centered Therapy, which was developed by Carl Rogers.

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ASR	Automatic Speech Recognition
CLI	Command Line Interface
DA	Digital Assistant
DL	Deep Learning
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
IR	Information Retrieval
IVR	Interactive Voice Response
JSON	JavaScript Object Notation
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
NLU	Natural Language Understanding
NN	Neural Network
OS	Operating System
PDA	Personal Digital Assistant
REST	Representational State Transfer
SUS	System Usability Scale
TTS	Text-to-Speech
UI	User Interface
VA	Virtual Assistant

Chapter 1

Introduction

This first chapter details the context, drive, and objectives behind this thesis. Section 1.1 focus on explaining the background and motivation of the thesis. Section 1.2 showcases the goals of the thesis. Section 1.3 details the thesis contributions. Lastly, Section 1.4 gives an overview of the structure of this document.

1.1 Background and Motivation

Rapid advances in Artificial Intelligence (AI) over the last decade have significantly transformed human-computer interaction. AI-driven systems, including Digital Assistants (DA), resort to techniques and technologies such as Automatic Speech Recognition (ASR), Natural Language Processing (NLP), and Machine Learning (ML) to improve user engagement and functionality. With applications on devices such as smartphones, smart speakers, and desktops, DA have become integral to modern workflows. These agents help users by answering their questions, be it through information retrieval or the execution of certain functionalities incorporated in them. Unlike Siri, Alexa, ChatGPT, and many others, which cater to mobile, smart home, and cross-platform environments, there are not many solutions for the desktop environment, which focus on task automation and application management in that domain.

1.2 Thesis Objective

This thesis aims to address the desktop solutions gap by developing and demonstrating a DA that will be designed and implemented specifically for the desktop environment, specially for the Windows Operating System (OS), capable of responding to voice or text commands.

The key features to be developed in this thesis include the implementation of a modular architecture, achieved through the definition of contracts via interfaces. This design enables seamless integration and abstraction between the DA and its own components. Another feature is the integration of open-source technologies to implement these components. The system also extends beyond conversational interaction by providing specific functionalities, namely retrieving weather updates and launching applications on the OS. Finally, the use of Large Language

Model (LLM) technology allows for more advanced language understanding and richer user interaction compared to older approaches.

Figure 1.1 depicts, in a broad sense, the way in which the DA functions. It is important to note that the DA only sends one answer, which can be written or voiced, and this answer can come from the LLM or the functionality.

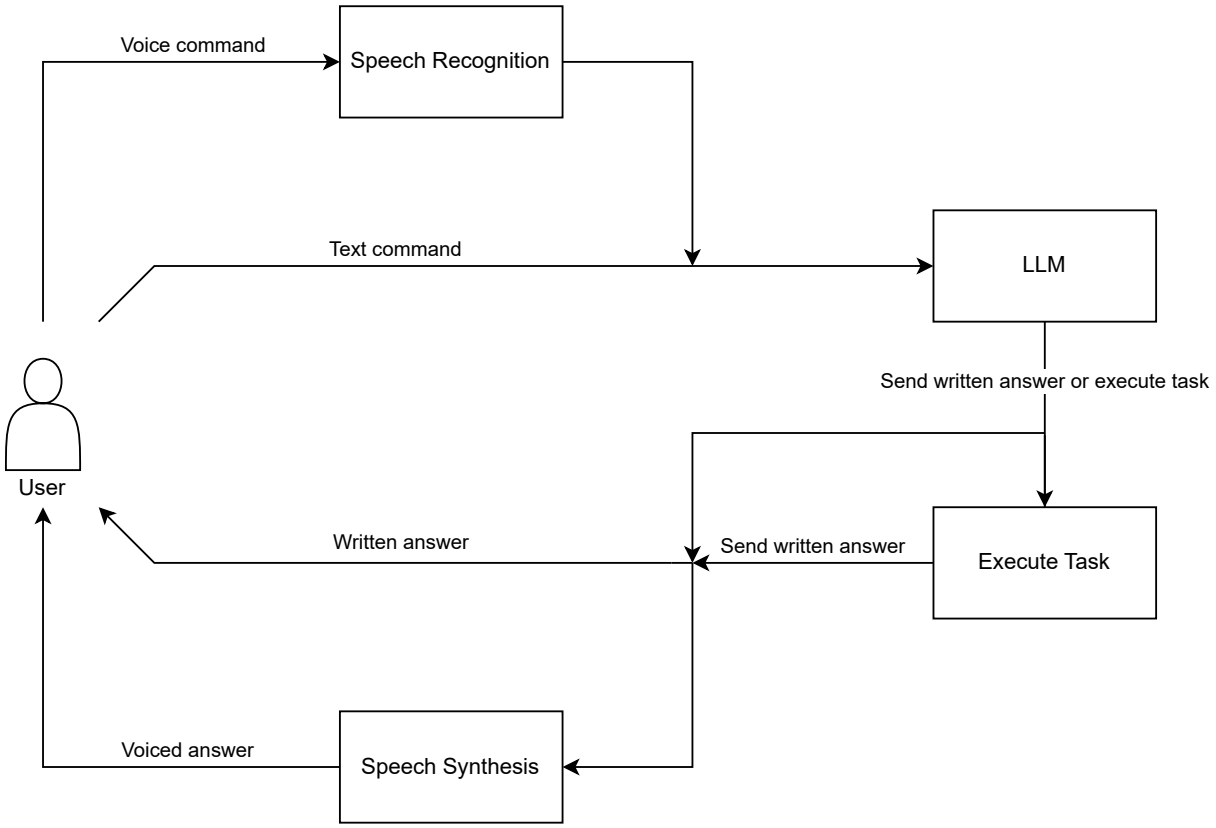

Figure 1.1 General functioning idea for the Digital Assistant

1.3 Thesis Contributions

The full source code of the project is publicly available at:

`https://github.com/pataponjak3/Digital-Assistant-Project`

Two scientific papers were written, presenting parts of this work, which contribute to its dissemination within the scientific community.
The following papers were published:

António Goulão, Dinis Dias, Artur Ferreira, and Nuno Leite, “A Personal Digital Assistant Enhanced with Artificial Intelligence Techniques”, Simpósio em Informática (INForum), September 2025, Évora, Portugal.

António Goulão, Dinis Dias, Artur Ferreira, and Nuno Leite, “A Modular Digital Assistant for Windows with Large Language Models”, Portuguese Conference on Pattern Recognition (RECPAD), October 2025, Aveiro, Portugal.

The system introduces a modular, layered architecture with a clear separation between the graphical interface, the core reasoning engine, and dynamically loaded modules. This structure promotes extensibility, maintainability, and the seamless integration of new functionalities without modifying the core system. Lastly, a key contribution of this thesis is the design of this architecture itself, which provides a flexible and future-proof foundation for intelligent assistant development.

1.4 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 examines the state of the art for DA, giving an understanding of their development over time, the tools and techniques used to implement them, and some challenges found in their development and usage. Chapter 3 presents the proposed solution, explaining objectives, defining requirements, presenting principles, and detailing the approach taken for the implementation of the project. Chapter 4 details the implementation of the proposed solution, through the modeling and definition of the architecture, the tools and technologies utilized, the implementation choices and design criteria chosen, and the discussion of key implementation aspects. Chapter 5 evaluates the implemented solution through unit and manual testing, analyzing its performance, usability, and ability to meet the requirements defined for this solution. Finally, Chapter 6 summarizes the main findings of this thesis, discusses its contributions and limitations, and outlines potential directions for future work.

Complementary materials supporting the analysis and validation of the system are provided in the annexes. Annex A presents detailed performance evaluation data, including parameters, test phrases, and raw results used in the quantitative assessment of the assistant. Annex B lists the external API endpoints employed by the system, covering integrations with meteorological and language model services. Lastly, Annex C reproduces the usability questionnaire used in the qualitative validation phase, including all task-related and post-test items administered to participants.

Chapter 2

State of the Art

DA have witnessed a significant advance with the evolution of AI in the last decade, although these have a history dating back to the 1960s. This chapter aims to present its history of evolution, the technologies that enabled this progress, some challenges related to the development of DA, and future directions.

Section 2.1 focus on explaining what DA are and present its historical development in a general sense, while also focusing on desktop environments, specifically on the Windows OS. Section 2.2 gives an overview of tools and techniques commonly used in the development of DA. Finally, Section 2.4 shows the main challenges of developing DA solutions.

2.1 Digital Assistants and its history

A Digital Assistant (DA) or Virtual Assistant (DA) is a software agent that performs tasks for a user based on their input, typically via speech or text. These tasks can include answering questions, managing personal data (for example, setting reminders or calendar events), or even controlling other devices through multi-device experiences [1].

The development of DA reflects the evolution of NLP and AI, as first shown in Table 2.1, which summarizes key systems from the rule-based era. These systems started with ELIZA [2] in the 1960s, demonstrating the potential for machine-driven dialogue, despite being limited to simple rule-based pattern matching, related to the fact that NLP had its roots in the 1940s [3]. Later systems, including PARRY [4] and chatbots like A.L.I.C.E. and SmarterChild, pushed these ideas forward, gradually improving natural language interaction [5, 6].

In the 1990s, Personal Digital Assistants (PDA) [7] such as the Apple Newton and Palm Pilot were introduced. Although not VA themselves, they contributed to the DA concept by showcasing features such as handwriting recognition, personal scheduling, and memo creation, where some of these elements later became standard in intelligent assistants. Around the same time, Interactive Voice Response (IVR) systems gained traction in call centers, bringing ASR and Text-to-Speech (TTS) into practical use and laying the foundation for voice-driven interaction.

The early 2010s marked the arrival of mainstream VA like Siri, Google Now, Cortana, and Alexa, which popularized speech interaction, real-time assistance, and used the improvement

Table 2.1 Rule-based Conversational Systems

Year - Mention	Underlying technology	Notable Features / Contributions
1966 - ELIZA	Pattern-matching, scripted rules	First widely known chatbot; simulated a psychotherapist exercising Rogers' therapy.
1972 - PARRY	Rule-based, state machine	Modeled paranoid schizophrenia; early attempt at simulating mental states.
1988 (early dev), 1997 (public) - Jabberwacky	Rule-based and learning database	Focused on humorous, playful conversation; precursor to Cleverbot.
1995 - ALICE (A.L.I.C.E.)	AIML (Artificial Intelligence Markup Language)	Won Loebner Prize multiple times; inspired many early chatbots.
2001 - SmarterChild	Pattern-matching, scripted responses	Deployed on AIM/MSN/Yahoo Messenger; offered weather, trivia, small talk.
2011 - IBM Watson (DeepQA)	NLP pipeline, Information Retrieval (IR) and ML ranking	Famous for winning Jeopardy!; answered questions from structured/unstructured data.
2011 - Siri	Intent recognition and rule-based dialogue	First major voice assistant on smartphones; integrated with iOS.
2012 - Google Now	Search and rule-based context cards	Provided proactive info (weather, traffic); predecessor of Google Assistant.
2014 - Cortana	Intent-based NLU	Windows and Xbox assistant; deep integration with Microsoft ecosystem.
2014 - Alexa	Intent recognition and skill-based system	Popularized smart speakers (Echo); extensible via third-party skills.
2017 - Bixby	Intent-slot filling and rule-based tasks	Deep integration with Samsung devices; supported routines and automation.

of speech recognition technologies on consumer devices [8]. A major breakthrough came in 2017 with the Transformer architecture, described in Attention Is All You Need [9], which enabled more context-aware and conversational AI models such as BERT [10] and GPT [11]. Since then, transformer-based architectures and LLM have become central to DA, enabling natural conversation and complex task handling [9, 12, 13]. This is observable in Table 2.2, that illustrates diverse models based on this architecture.

In the Windows ecosystem, Microsoft's first experiments with assistant-like features included Microsoft Bob (1995) [14] and Microsoft Agent (1997) [15], both of which used animated characters to guide users. The launch of Cortana in 2014 introduced Windows with a true VA, although it was eventually discontinued due to limited adoption [16]. Microsoft's current solution, Copilot [17], is powered by LLM and is integrated in Windows, Microsoft 365, GitHub, and others. The release of Microsoft Mu [18], a compact on-device LLM, represents Microsoft's latest move toward low-latency, privacy-friendly, locally executed assistance.

Together, these developments highlight the steady progression from simple scripted systems to powerful, context-aware assistants that are becoming an integral part of modern computing.

2.2 Tools and Techniques

Modern DA are the result of the combination of several complementary technologies, many of which were introduced in the historical overview of Section 2.1. Whereas early systems relied on handcrafted, rule-based approaches, current assistants are predominantly data-driven and powered by Machine Learning (ML) [19, 20] and Deep Learning (DL) [21, 22]. These approaches allow assistants to learn from examples, generalize to new situations, and improve over time.

At the core of most assistants lies ML, which supports tasks such as intent recognition, entity extraction, and personalization. By analyzing previous user interactions, DA can adapt to individual preferences and provide more relevant responses. DL approaches, which use multilayered Neural Networks (NN), have been particularly successful in processing unstructured data such as speech and text, enabling breakthroughs in Natural Language Understanding (NLU) [6, 23], Automatic Speech Recognition (ASR) [24, 25], and Text-to-Speech (TTS) synthesis [26].

Natural Language Processing (NLP) [6, 27] is another essential component that binds human language and machine interpretation. Modern NLP pipelines handle tokenization, part-of-speech tagging, sentiment analysis, and dialogue management, with neural architectures, especially those based on the Transformer [9], now dominating the field. These models, summarized in Table 2.2, are capable of capturing long-range context, enabling assistants to generate coherent, context-aware responses in multi-turn conversations [12, 13].

Speech technologies are equally critical. ASR systems convert spoken input into text, often using neural encoders such as Wav2Vec 2.0 to achieve near-human transcription accuracy [28]. Conversely, TTS systems transform text back into natural-sounding speech, using sequence-to-sequence models such as Tacotron 2 and neural vocoders like WaveNet [29, 30]. Together, these components allow for seamless multimodal interaction.

Finally, assembling these technologies into a usable assistant requires careful system design. Modern DA adopt modular, API-driven architectures, enabling developers to add new capabilities (“skills” or “actions”) without altering the core logic. This extensibility has been key to the success of assistants like Alexa and Google Assistant, which can grow continuously through third-party integrations [31, 32].

As introduced earlier, the Transformer architecture [9] has become the foundation for nearly all state-of-the-art NLP systems and, by extension, DA. Its self-attention mechanism allows every token in an input sequence to consider all others simultaneously, enabling the modeling of complex relationships and long-range dependencies. This capability is particularly valuable for digital assistants, which must handle multi-turn conversations and maintain context.
A typical Transformer consists of multiple layers with the following key components:

- Positional Encoding - Injects order information into token embeddings, since the model processes tokens in parallel rather than sequentially.
- Multi-Head Self-Attention Enables the model to focus on different types of relationships

between tokens simultaneously (e.g., syntactic, semantic).

- Feed-Forward Networks - Fully connected layers that transform the attention outputs, adding non-linearity and expressiveness.
- Residual Connections and Normalization - Facilitate stable training and support deeper networks.
- Masked Attention (for decoders) - Ensures that text generation happens autoregressively by preventing access to future tokens during training.

This architecture offers three major benefits that explain its dominance:

1. Global Context Awareness - Captures long-range dependencies crucial for coherent dialogue.
2. Parallelism and Scalability - Processes tokens simultaneously, enabling efficient training on large datasets and scaling to billions of parameters.
3. Flexibility - Can be used for understanding tasks (BERT-style encoders), generation (GPT-style decoders), or both (T5-style encoder-decoder models).

The adoption of Transformer-based models, as summarized in Table 2.2, has enabled modern assistants to move beyond scripted responses and provide dynamic, human-like interaction, capable of reasoning over user input and producing contextually appropriate output.

2.3 Technologies, Approaches, and Platforms

Several platforms and frameworks support the creation of digital assistants. Among the most relevant are:

- Rasa – An open-source framework for intent recognition, entity extraction, and dialogue management using ML and DL models [31].
- Dialogflow – Google's platform integrating NLP and speech processing via cloud services.
- Microsoft Bot Framework – A comprehensive SDK and cloud integration for conversational AI on Azure.
- IBM Watson Assistant – Provides cloud-based NLP, intent recognition, and multi-channel deployment capabilities [33].
- Alexa Skills Kit – Enables developers to extend Amazon Alexa with custom "skills" [32].

Recent advances also include LLM-based frameworks (e.g., OpenAI GPT, Gemini, and LLaMA) that allow assistants to reason, generate text, and integrate external tools through structured "function calling" API [34]. In contrast with these cloud-first solutions, the system developed in this work focuses on a local, desktop-oriented architecture that ensures privacy, low latency, and extensibility through modular components.

2.4 Challenges

Because DA are assigned to do the tasks that the user requires and are present on the user's machine, they collect data for training and personalization, and some might be personal data. This data, if not handled correctly, might be used for others advantage. An example of this is the possibility that someone using the VA on the user's machine may buy things using their account [35]. With this, security and privacy are one of the challenges present when developing a VA, where, for example, in Europe, software applications must follow the General Data Protection Regulation (GDPR).

Because DA are composed of different technologies, they will inherently acquire the challenges present in those technologies. These range from moral issues, like the impact that these tools will have on people's jobs or the possibility of they achieving singularity. We may also face technical problems, like the introduction of bias in training, computational costs, failure of understanding the context given by the user or failure to cater the needs of the user [19, 27, 36, 37].

Table 2.2 Transformer-Based Conversational AI Systems

Year - Mention	**Underlying technology**	**Notable Features / Contributions on Release**
2018 - GPT-1	Decoder-only Transformer	First large-scale generative transformer trained on unsupervised text.
2018 - BERT	Encoder-only Transformer	Bidirectional pretraining (masked language modeling); revolutionized NLP understanding tasks.
2019 - GPT-2	Decoder-only Transformer	Multi-paragraph coherent text generation; sparked debate on large model release.
2019 - T5	Encoder-Decoder Transformer	Unified NLP tasks into text-to-text format; strong benchmark performance.
2020 - GPT-3	Decoder-only Transformer	175B parameters, few-shot learning capabilities; foundation for ChatGPT.
2021 (preview), 2022 (public) - GitHub Copilot	Decoder-only Transformer (OpenAI Codex)	Coding assistant; generates code in IDEs using natural language prompts.
2022 - ChatGPT	GPT-3.5 (fine-tuned)	General-purpose conversational AI; multi-turn dialogue; RLHF fine-tuning.
2023 - GPT-4	Multimodal Decoder-only Transformer	Handles text and images; powered ChatGPT Plus; improved reasoning.
2023 - Claude	Constitutional AI + Transformer	Alignment-focused LLM; safer responses; general-purpose conversational AI.
2023 (rebrand in 2024) - Gemini (Bard)	Multimodal Transformer	Combines text, image, code reasoning; successor to Bard.
2023 - Microsoft Copilot	GPT-4 based	Productivity assistant integrated into Microsoft 365 apps (Word, Excel, PowerPoint); drafts documents, summarizes content, performs data reasoning.
2023 - Llama	Decoder-only Transformer	Open-weight LLM series for research; spawned models like Alpaca, Vicuna.
2023 - Mistral 7B / Mistral	Sparse Mixture-of-Experts Transformer	Efficient, open-source LLMs with competitive performance.
2025 - GPT 4.5	Decoder-only Transformer	Enhanced reasoning, multimodal abilities, supports up to 128K tokens.
2025 - Gemini 2.5	Multimodal Transformer	Advanced reasoning, live multimodal APIs, tool integration.
2025 - Phi-4	Transformer-based	Open-source, developer-friendly, supports fine-tuning and local deployment.
2025 - DeepSeek-R1	Transformer-based	Efficient text processing and generation using self-attention mechanisms.
2025 - Grok 4	Mixture-of-Experts Transformer	Scientist-level reasoning, coding mode, natural voice, content interpretation.
2025 - GLM-4.5	Transformer-based	Intelligent agent applications; prominent in China's AI ecosystem.
2025 - GPT-5	Unified Transformer	Enhanced reasoning, multimodal capabilities, reduced hallucinations, dynamic model selection.
2025 - Microsoft MU	330M parameter Encoder-Decoder Transformer	On-device LLM for real-time Windows assistance; low-latency system integration.

Chapter 3

Proposed Solution

Evaluating all the points mentioned above, with respect to history, tools, techniques, and challenges regarding DA, it is observed that there are not many solutions for the Windows desktop environment capable of performing the tasks users need. Moreover, when such tasks are not supported, existing solutions often lack the accessibility of easily adding new features, a limitation that the proposed solution aims to address.

In this chapter, the objectives, requirements, theoretical principles, and approach of the proposed solution are defined. Section 3.1 outlines the objectives that the solution must have. Section 3.2 shows the defined requirements of the solution. Section 3.3 gives an overview of the theoretical and technical principles followed in the development. Finally, Section 3.4 presents the approach taken for the development of the solution.

3.1 Objectives

The primary goal of this project is to develop a modular and extensible DA tailored for the Windows desktop environment. Unlike existing VA focused on mobile or smart home platforms, this assistant aims to integrate with third-party applications and services, using natural language commands, spoken or written, to perform concrete tasks, such as launching applications, retrieving weather updates, or even responding with general knowledge using a LLM. The proposed solution should be structured to support both flexibility in feature extension and adaptability to future improvements in the different components (be it the LLM, the services, or other components) that incorporate the DA.

3.2 Requirements

To implement the project with the aforementioned objectives, requirements, descriptions of the expected behavior and characteristics of the system are defined to serve as the foundation of the design and implementation of the DA. These are organized into two categories: functional requirements, which define what the system must do, and non-functional requirements, which describe how the system should behave in terms of quality attributes such as usability,

performance, and scalability. The functional requirements are detailed in Table 3.1 and the non-functional requirements are described in Table 3.2.

Table 3.1 Functional requirements of the project

Ref.	Description	Priority
FR1	The system shall present a GUI where users can interact with the DA through text or speech and observe the conversation as it goes.	Must
FR2	The system shall include speech related functions that are provided as:	Must
FR2.1	Speech recognition for user input.	Must
FR2.2	Speech synthesis for DA output.	Must
FR3	The system shall include speech configurations that are provided as:	Must
FR3.1	Enable speech output.	Must
FR3.2	Select input device (microphone) for users.	Must
FR3.3	Configure input device sensitivity.	Should
FR4	The system shall interpret user input utilizing a LLM to generate responses or determine whether a functionality should be executed.	Must
FR5	The system shall provide these functionalities to be executed:	Must
FR5.1	Meteorology functionalities:	Must
FR5.1.1	Current weather conditions.	Must
FR5.1.2	Five-day weather forecast.	Must
FR5.1.3	Current air pollution levels.	Must
FR5.2	OS functionalities:	Must
FR5.2.1	Launch applications.	Must
FR5.2.2	Search files.	Should

3.3 Principles

The proposed solution is grounded on a set of theoretical concepts and practical technologies that together enable intelligent human–computer interaction. This section outlines the core principles that guided the project's development, including language understanding, modular design, speech interfaces, and REST-based integration.

3.3.1 Language Model Reasoning and Function Calling

At the core of the assistant lies a conversational reasoning system driven by a LLM. These models have the benefit of being able to generalize across tasks and domains, which makes them ideal for interpreting a variety of user commands and requests in various contexts.

The system's use of function calling through natural language is one of its defining principles. Instead of using pre-programmed rule-based logic, the assistant uses the LLM to understand user intent and decide whether to execute an available functionality or provide a direct, natural response. The LLM serves as a mediator between the user and the internal operations of the system, introducing a dynamic and appropriate method of matching human input with system

Table 3.2 Non-functional requirements of the project

Ref.	Description	Priority
NFR1	The GUI shall be intuitive, easy to navigate, and responsive to user actions.	Must
NFR2	The system shall respond to user inputs within a reasonable time frame, recognizing that the latency depends on external services (e.g. the LLM or functionalities that require external access).	Must
NFR3	The system shall handle errors gracefully, such as unavailable API or unrecognized commands, by providing informative messages to the user.	Must
NFR4	The architecture shall be modular to support interchangeable modules for functionalities, speech recognition, speech synthesis, and LLM.	Must
NFR5	The system shall be able to integrate new modules without requiring refactoring of the existing core.	Must
NFR6	The system shall run on the Windows OS, with the possibility for other OS provided that dependencies are met.	Should
NFR7	The use of interfaces and layered design shall ensure that the system can be maintained and extended with minimal impact on existing code.	Must
NFR8	The GUI shall remain responsive during heavy operations (e.g., LLM calls, TTS synthesis) by delegating tasks to background execution.	Must
NFR9	The system shall not store sensitive user input or transmit it to third parties beyond what is required for external services (e.g., weather API).	Should

capabilities.

By using this method, the assistant can still provide precise, structured control when calling internal operations while maintaining a conversational, adaptable interface.

This design principle takes inspiration from recent developments in open frameworks such as OpenAI's Function Calling [34] and Rasa's modular architecture [31], which promote extensibility and decoupling between dialogue reasoning and functionality execution. The same modular logic is adopted in this work to ensure scalability and flexibility of the system's core.

3.3.2 Modular Design and Software Architecture Principles

The system follows a modular architecture, where the assistant is made up of separate but interchangeable parts. This aligns with foundational software engineering principles such as:

- Encapsulation – Only clearly defined interfaces are exposed; each module retains its own logic and data.
- Separation of Concerns – Different system layers handle different tasks, including language processing, function execution, and user interaction.
- Pluggability/Modularity – By incorporating new modules, new features can be added without requiring modifications to the system's other parts.
- Scalability – The system's ability to increase in complexity without sacrificing readability or maintainability.

Every feature (like weather updates, or launching applications) can develop independently and be reused or replaced as needed thanks to the modular principle.

An interface-driven design that uses abstract base classes to specify explicit contracts for every component supports this architectural vision. This method enables the smooth interchange of modules such as the speech recognizer, functionality provider, and LLM interface. It adheres to the principles of software architecture described by Bass et al. [38], which emphasize component modifiability, separation of concerns, and maintainable design through clearly defined interfaces.

3.3.3 Human–Computer Interaction and Speech Interfaces

To ensure natural and accessible interaction, the system incorporates both text-based and speech-based input and output. This supports a multimodal interface that aligns with the expectations of modern users.

From a theoretical point of view, the ASR and TTS systems rely on probabilistic models and acoustic to linguistic mappings to bridge the gap between speech and text. These technologies enable the assistant to process both verbal input and communicate responses vocally, enhancing usability in hands-free scenarios and for users with limited mobility or literacy.

The integration of these speech systems complements the natural language reasoning of the LLM, creating a fluid loop between spoken command and the corresponding response.

3.3.4 Integration with External Information Sources

Modern assistants are expected to provide relevant and timely information by tapping into external knowledge sources. The proposed solution incorporates the principle of external data integration through standardized web service protocols, such as Representational State Transfer (REST) API.

This enables the assistant to interact with real-world services (for example, retrieving weather data, calendar events, or web-based content) and adapt to changing user contexts. Theoretical foundations for this integration lie in client-server communication models, data serialization (e.g., JSON or XML), and the abstraction of service interactions through API contracts.

This design principle ensures that the assistant is not a static entity, but a dynamic one, capable of extending its knowledge and capabilities through connectivity.

3.3.5 Visual Interface Design and Usability

The assistant features a graphical interface that facilitates user-friendly interaction. Based on the principles of usability, responsiveness, and clarity, the interface design emphasizes direct manipulation, minimal input effort, and clear visual feedback.

The GUI serves as the primary communication bridge between the user and the reasoning backend of the system. It reflects the assistant's responses, offers tools for selecting input/output modes, and maintains the conversational context in a readable format. Theoretical guidance

for the design of this component is drawn from user interface design heuristics, event-driven architecture, and model–view–controller (MVC) architecture.

By aligning the behavior of the interface with the expectations of the user, the assistant promotes an intuitive and accessible interaction experience.

3.4 Approach

In terms of the approach to the proposed solution development, seven main phases were defined to implement the project: research, design, development, testing, documenting, refactoring, and packaging.

For research, information on how to construct a DA and what languages, tools, libraries, and best practices to use for the construction of the project were obtained. In the design, understanding the structure of a DA was the key to formulate the design into three main layers, the GUI, the backend and the functionality features. The design phase also raised questions like *"How can everything be modular and interchangeable without code changes?"*, *"How should the LLM execute the functionality features and interact with the GUI?"*, *"How can certain details of the project be prominent in the GUI?"*, and *"How can the features give their information regarding execution to the LLM?"* which were then answered as the next four phases of the approach happened.

For the development of the code itself, *Python* was chosen as the programming language because of its ease of use, readability, and rich library ecosystem, which facilitates web integration, GUI development, and natural language processing. Its high-level syntax and community-driven ecosystem make it particularly suitable for scientific computing and rapid prototyping, which are key enablers for developing AI-oriented systems [39]. Additionally, its dynamic typing facilitates quick iteration cycles during design and testing, and its cross-platform nature permits future scalability beyond Windows.

As for the GUI, *PyQt5* was used to implement it because of its robust support for event-driven and multithreaded programming, integration with Qt Designer for visual prototyping, and sophisticated widget toolkit. It provides a versatile and aesthetically refined environment appropriate for scalable desktop applications [40]. The interface was developed through an iterative refinement process in which layout composition, color schemes, and text styling were evaluated for visual balance and clarity. This guaranteed a visually harmonious and responsive setting that complemented the assistant's conversational style.

By the end, the project would be packaged as an executable, for it to be then distributed to users who want to utilize it.

With all of this done, an online questionnaire was made, and the proposed solution was made available for the users to evaluate, while answering the questionnaire. This was done to evaluate the degree of usability of the solution.

Chapter 4

Implementation of the Solution

This chapter provides a comprehensive breakdown of how the proposed digital assistant was transformed into a functional system. Following the direction outlined in the previous chapter, we describe the development of the solution in four main sections.

Section 4.1 introduces the modeling and architectural structure of the solution, describing the high-level organization of the system and the interaction between its main layers: the Presentation, the Core, and the Module Layer. Section 4.2 indicates the tools, frameworks, and libraries adopted throughout the implementation, justifying their choice based on performance, compatibility, and suitability for the tasks at hand. Section 4.3 discusses the key implementation choices and design criteria, including considerations for modularity, language model integration, and responsiveness. Finally, Section 4.4 highlights and analyzes the most critical aspects and challenges of the implementation, providing insight into technical solutions and their impact on the overall behavior of the system. Additional technical information regarding the external API integrations, including meteorology and language model services, is presented in Annex B.

4.1 Modeling and Architecture of the Solution

At the core of the implementation lies a layered architecture with characteristics of a plugin-based design pattern. Its structure separates concerns while ensuring extensibility through modular components that are dynamically loaded through the usage of interfaces and dependency injection.

To contextualize the assistant's scope and main user interactions, Figures 4.1 and 4.2 present the global and detailed use-case diagrams of the system.

Conceptually, the system is organized into three main layers: the Presentation Layer, the Core Layer, and the Module Layer (which is organized into two sub layers, namely Base Modules and Functionality Modules).

In Figure 4.3, an overview of the system's architecture is shown, with a clear separation between these distinct layers. The Presentation Layer provides a responsive environment for user interaction through the GUI, supporting both textual and speech communication. Its main responsibilities include displaying the conversation history, capturing user input (typed

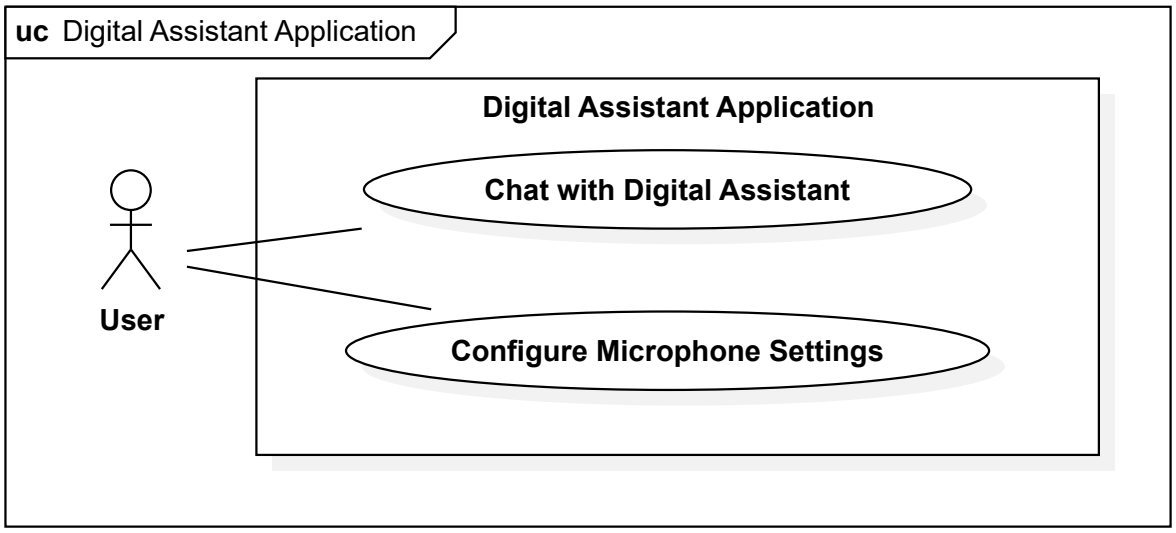

Figure 4.1 Use Case Scenarios General Diagram

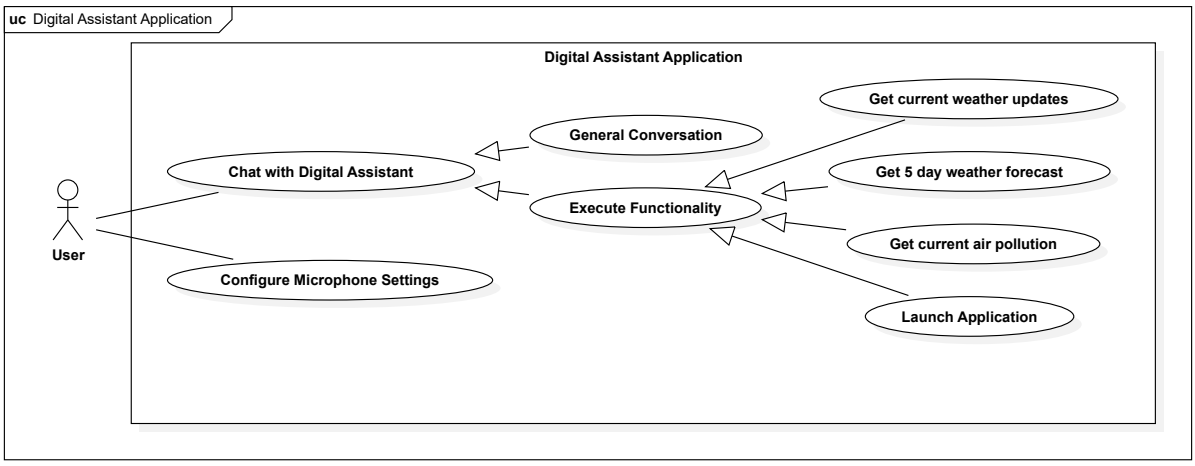

Figure 4.2 Use Case Scenarios Detailed Diagram

or spoken), displaying buttons to trigger actions, and allowing the user to toggle voice output or configure microphone settings. To enforce a clean separation from the backend logic, the Presentation Layer does not interact with the Core directly. Instead, all requests are routed through an intermediary component, the chat handler, which acts as a facade that executes actions on different parts of the system.

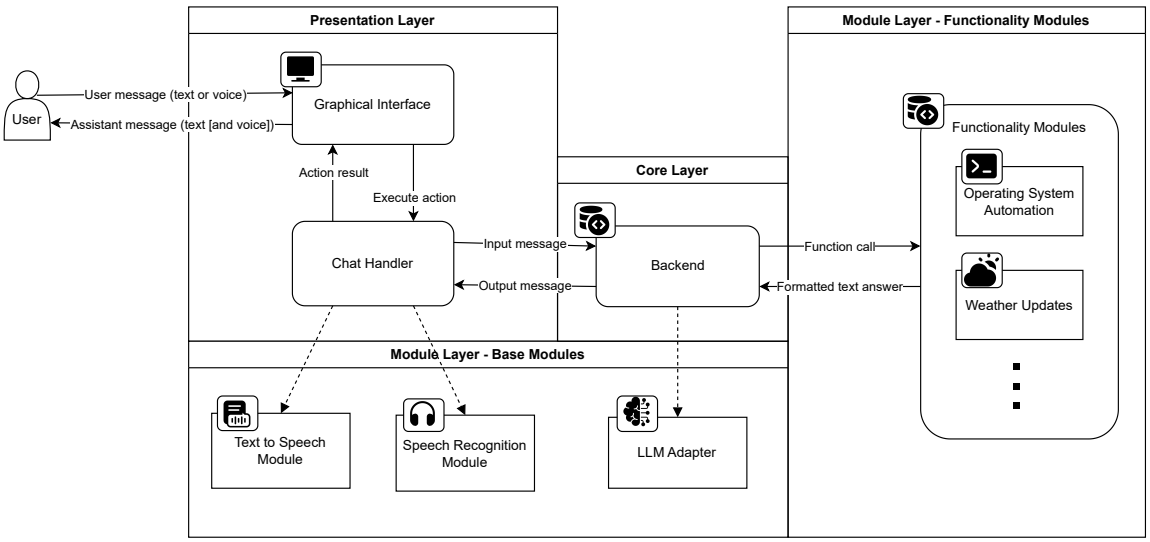

Figure 4.3 System Architecture Diagram of the Digital Assistant

The Core Layer coordinates the logic and control flow of the system through the backend. It receives input from the Presentation Layer and mediates with the LLM, acting as a reasoning engine, where it interprets requests and determines whether a response should be generated directly or whether a functionality needs to be executed through a function handler. A key characteristic of this layer is its reliance on abstract interfaces that decouple central logic from concrete implementations and delegate how components must behave. This ensures that the core remains lightweight and stable while still enabling the integration of different modules, like the LLM, speech services, or functionality providers.

Finally, the Module Layer provides all the extensible and specialized capabilities of the assistant. It encapsulates independent modules that are self-contained, that implement functionalities such as weather services, system automation, or speech processing. In this project, there are two types of modules, base modules and functionality modules, where, regardless of its type, a module adheres to predefined contracts so that it can be dynamically discovered and integrated without altering the core functionality of the DA. In the case of base modules, integral components to the normal functioning of the DA, their contracts only expose the functions necessary to the execution of a certain functionality. However, the contracts of functionality modules do not adhere to this; instead, they expose two special functions, one that must define descriptive metadata for each function that the module wants to execute, and another that executes these functions based on the provided metadata.

The dynamic discovery available in these modules is possible through the usage of a plugin

mechanism, which enables new functionality modules to be introduced, and both module types to be replaced without modifying the main architecture. This makes the solution scalable and adaptable to evolving requirements.

To complement the architectural overview, Figures 4.4 through 4.11 depict the internal organization of the main packages and classes.

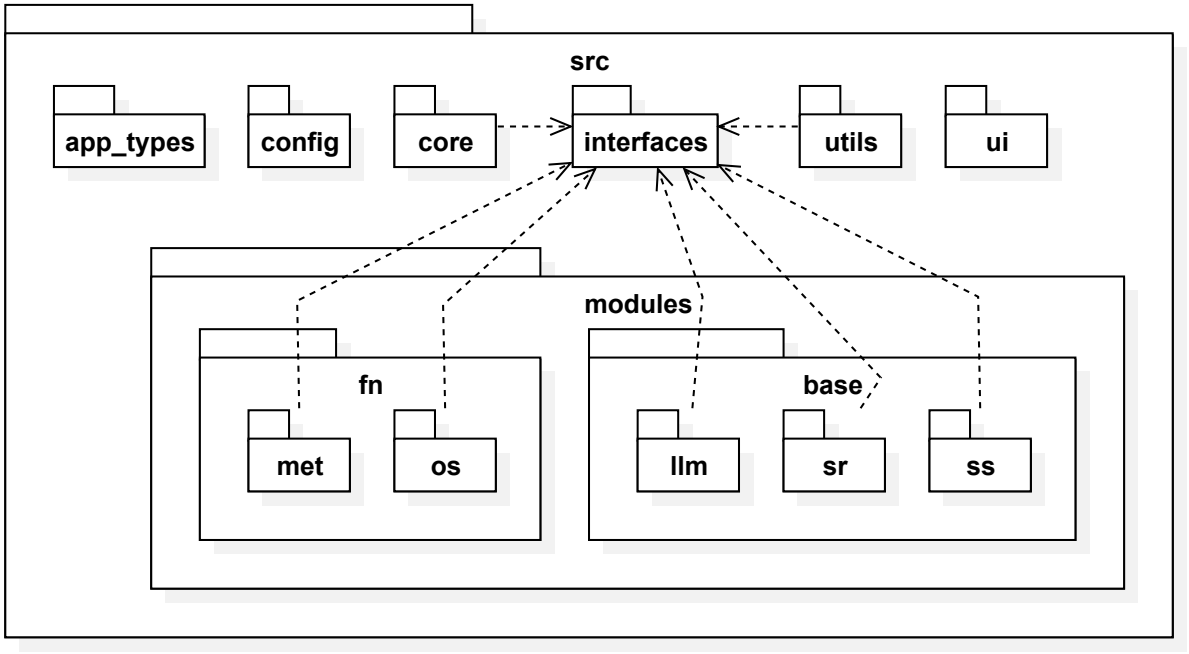

Figure 4.4 Project Package Diagram

This is carried out with interfaces that act as design contracts. They ensure that different components can interoperate while remaining loosely coupled. Some contracts exist to enable interchangeability of modules (e.g., different speech or functionality providers), while others are defined simply to enforce a clear separation of roles (e.g., between user interaction and backend coordination).

The use of contracts, combined with the plugin mechanism, allows the solution to maintain a lightweight and stable core while enabling dynamic extension of its capabilities.

This architecture thus balances clarity of responsibility (via separation into layers) with flexibility (via plugin-based extensibility), making the digital assistant both robust and adaptable.

4.2 Tools and Technologies Used

The implementation of the digital assistant relies on a set of tools such as the programming language, frameworks, and external services that together enable modular extensibility, speech processing, and user interaction. This section presents the main tools and technologies used in the development of the solution, justifying their choice and role in the project.

For the programming language, *Python* 3.10 was employed to implement the solution. As introduced in the previous chapter, *Python* was selected for its readability and strong ecosystem

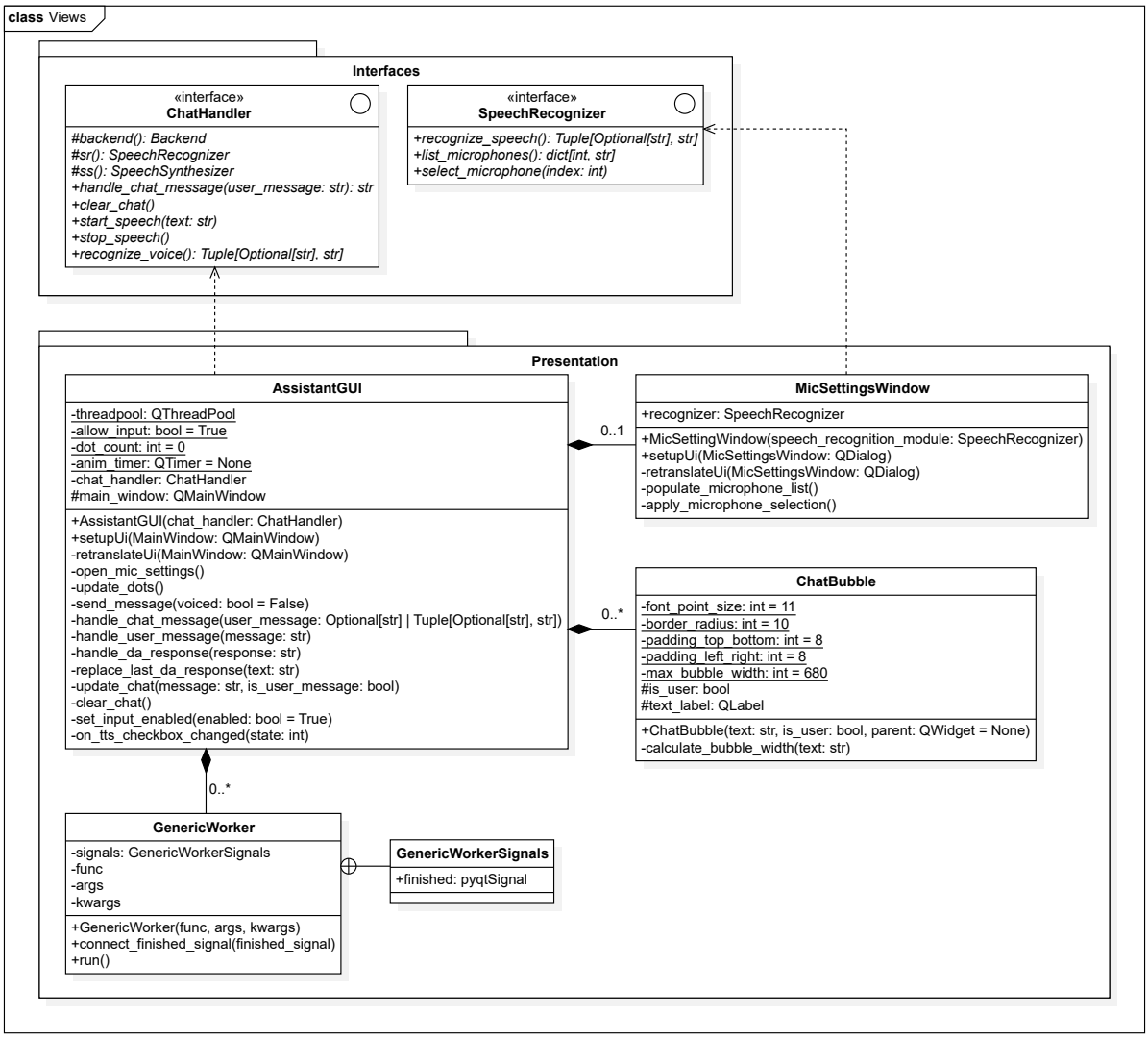

Figure 4.5 Presentation Layer Class Diagram

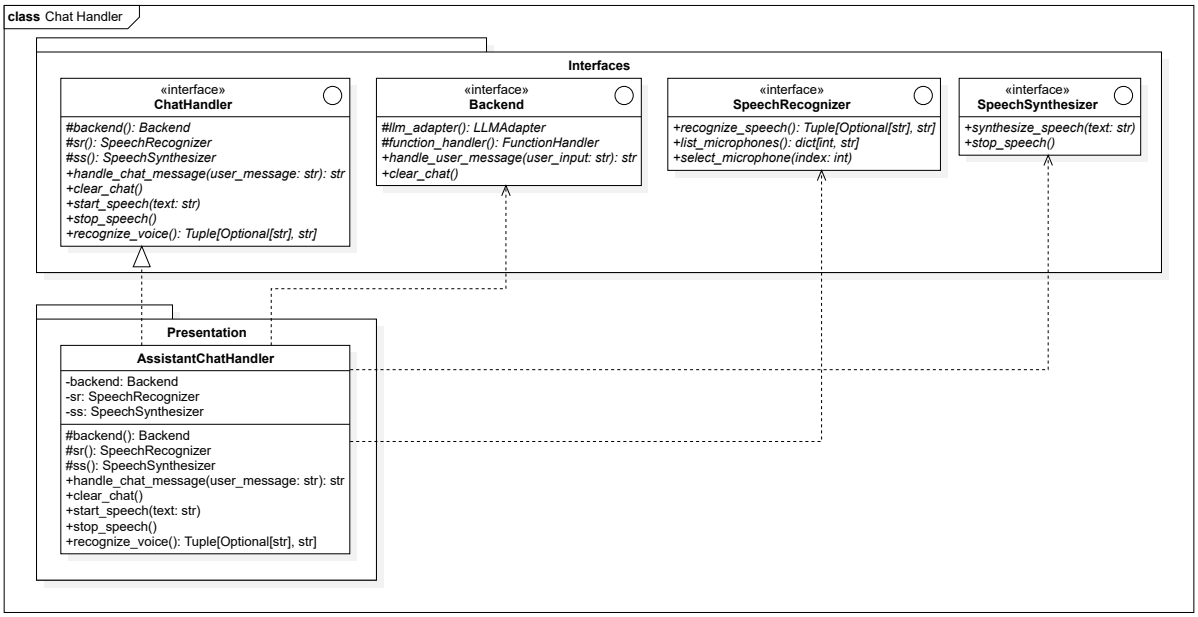

Figure 4.6 Chat Handler Class Diagram

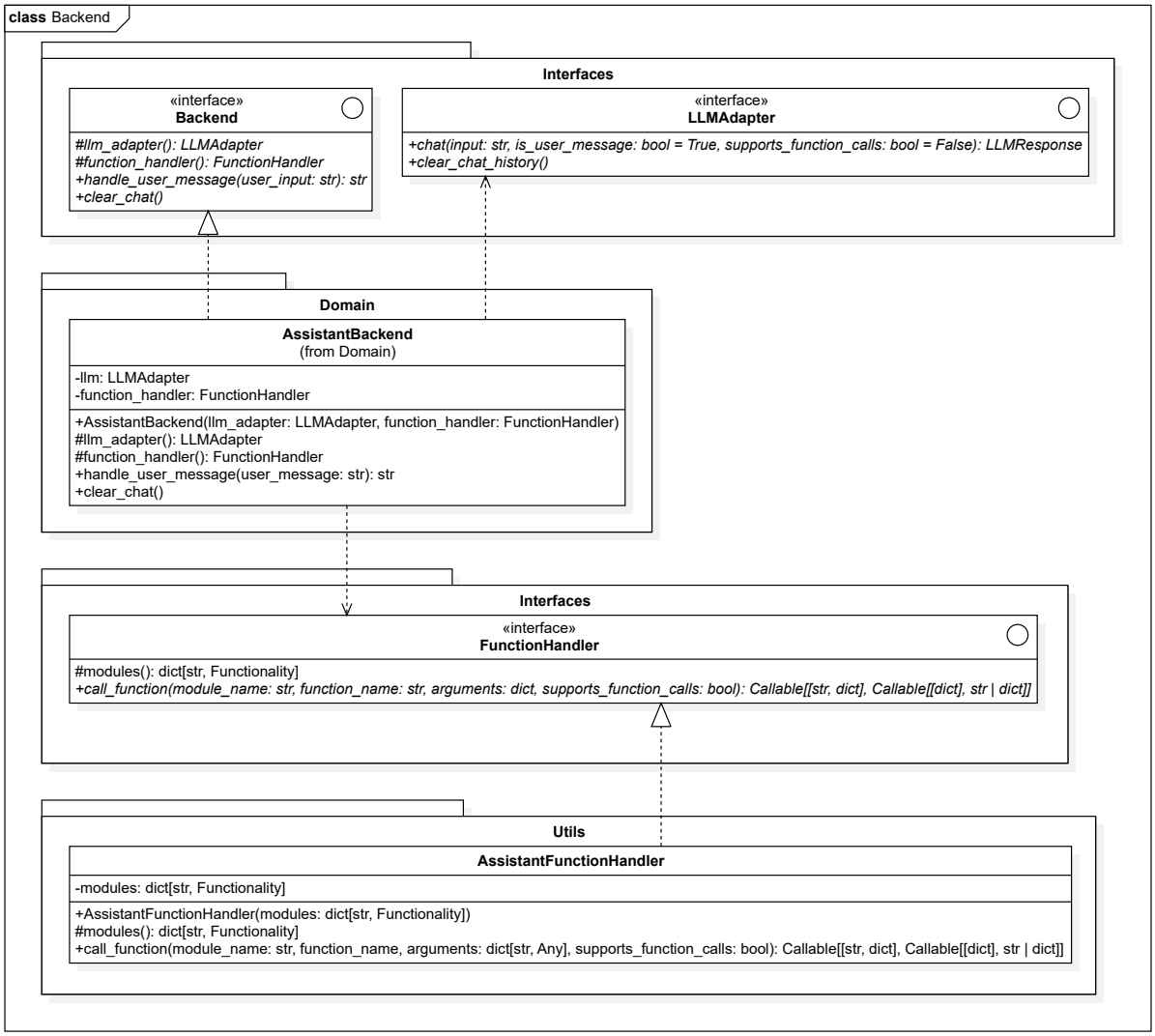

Figure 4.7 Backend Class Diagram

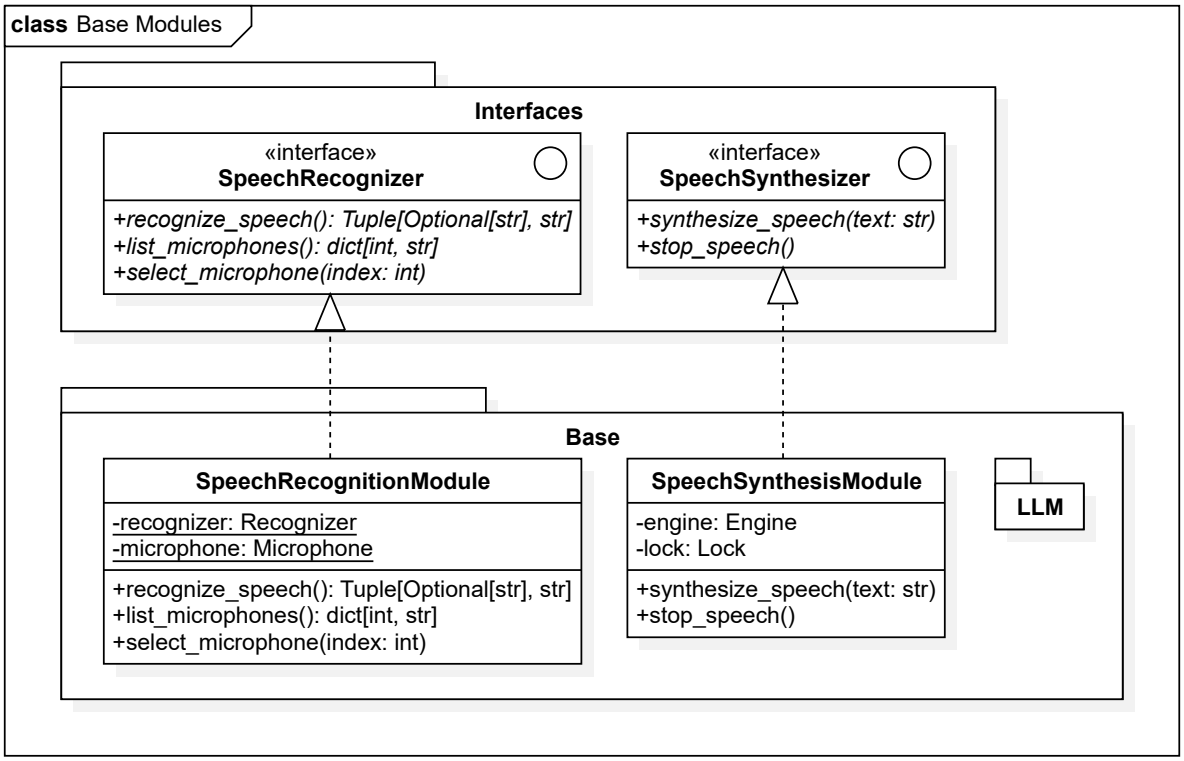

Figure 4.8 Base Modules Class Diagram

for artificial intelligence and interface development. In practice, its modular package management and extensive standard library facilitated the integration of diverse components such as speech recognition, text-to-speech, and system automation within a single coherent environment. Its dynamic typing and interpreted nature also supported rapid iteration during debugging and testing, accelerating the development process.

In the presentation and interaction modules, *PyQt5* served as the framework for constructing the graphical user interface, providing the foundation for both the main chat window and the microphone settings view. While the previous chapter described its selection rationale, in implementation terms, *PyQt5* allowed for a clear separation between layout design (handled through Qt Designer) and event handling (implemented through Python slots and signals). Multithreading support was leveraged to ensure that background operations, such as text-to-speech synthesis and API requests, did not block the interface, maintaining a fluid user experience. This integration enabled the GUI to remain responsive and visually consistent while coordinating asynchronous interactions with the backend.

For communication with external services, the *requests* library was key to enable seamless integration with the LLM backend and third-party API like the one used in the meteorology functionality (see Annex B for a detailed description of the API endpoints employed). Its ease of use in constructing and handling HTTP requests made it an adequate tool for both the backend and the meteorology module.

To enable dynamic discovery and configuration of modules, the Python standard *importlib* library was utilized, which allowed modules to be imported at runtime. The *json* library was used

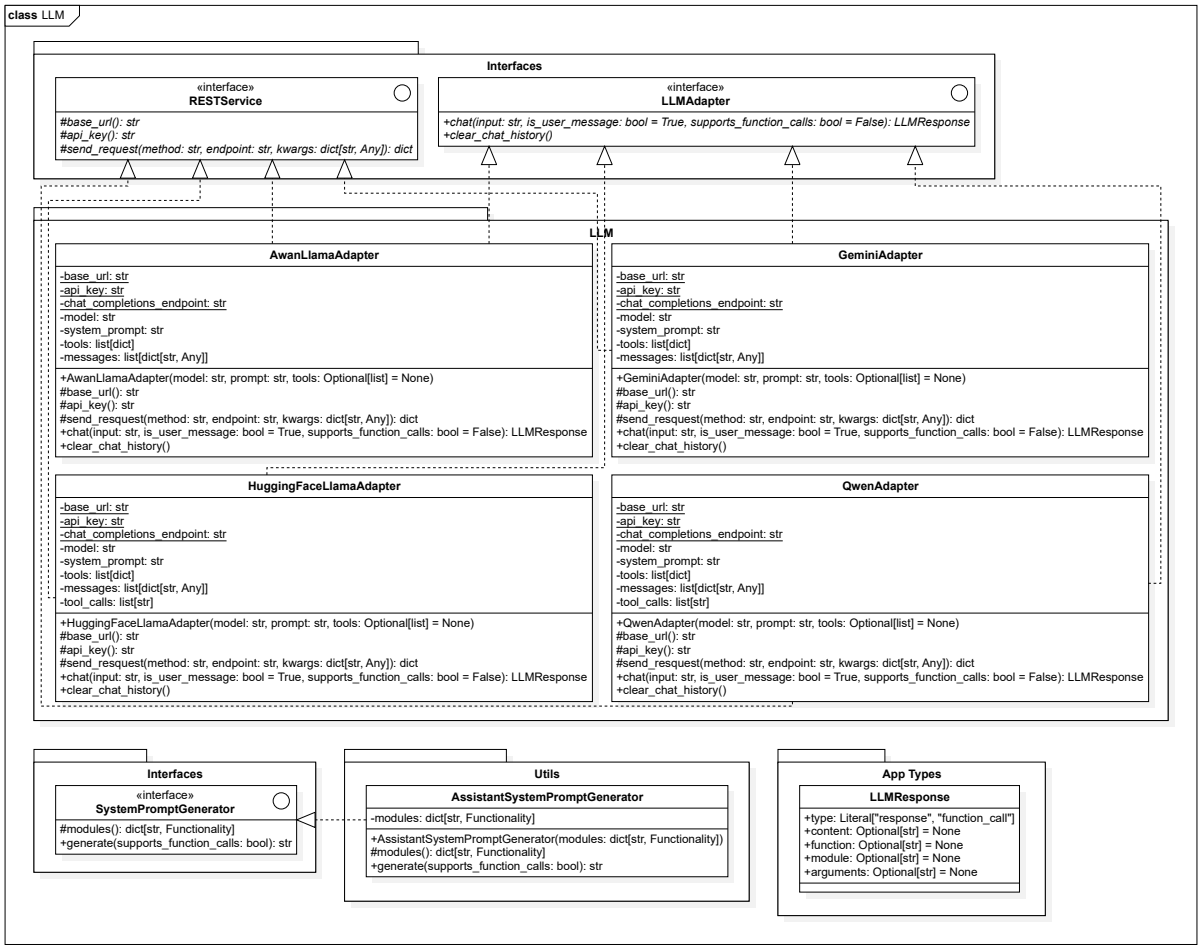

Figure 4.9 LLM Class Diagram

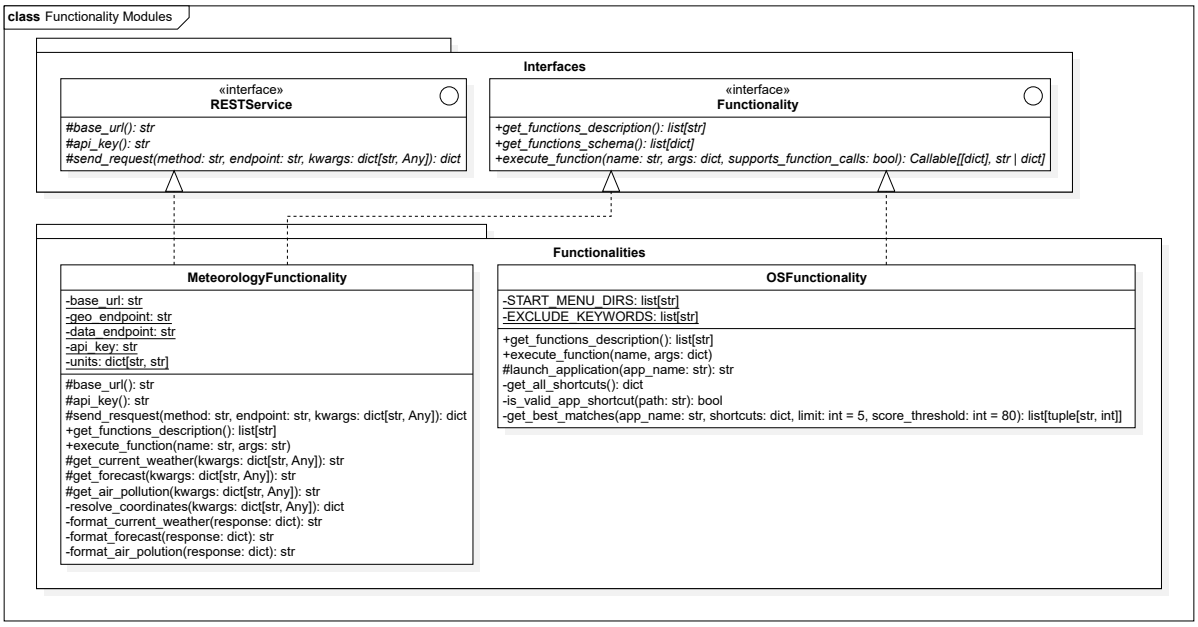

Figure 4.10 Functionality Modules Class Diagram

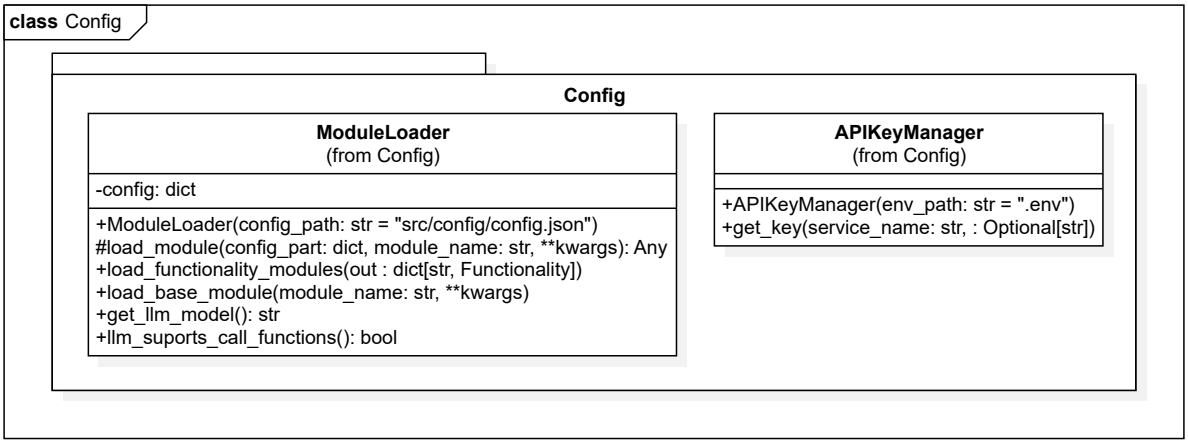

Figure 4.11 Configuration Class Diagram

to parse the configuration file that defines which modules should be loaded, while the *os* library managed filesystem-level tasks such as resolving configuration paths and interacting with the environment. Together, these ensured that the system remained flexible and easily extensible.

Regarding speech processing, the *SpeechRecognition* library was used to enable user speech recognition, capturing input, and converting it into text for further processing; while the *pyttsx3* library was chosen to allow the DA having a voice for it's answers.

Figures 4.12 and 4.13 illustrate the operational flow of the audio configuration subsystem, focusing on the microphone settings workflow.

Both were picked for their abstraction to multiple engines, providing flexibility and ease of integration with the assistant, and their ability to run locally without relying on cloud services (although possible in some recognition engines), ensuring independence from external providers and enabling immediate responses without latency introduced by network calls.

With the OS automation module, some libraries were chosen for their handling with system-level tasks, like *subprocess*, *os* and *win32com*. In addition, for the launch application functionality, type-matching was required to have better results in the discovery of apps, where the *RapidFuzz* library was used. *RapidFuzz* provides efficient fuzzy string matching algorithms implemented in C++, allowing fast and accurate comparisons between the user's query and available application names. It was selected over alternatives such as *FuzzyWuzzy* due to its higher performance, minimal dependencies, and permissive MIT license, which simplifies integration in open and closed-source environments [41].

In addition, with development and modeling tools, StarUML was used to create the design models of the system, including class diagrams, sequence diagrams, and use case diagrams. These diagrams documented the structure and behavior of the system, supporting both the implementation and the presentation of the architectural vision. For version control, Git was used throughout the project, ensuring that code changes could be tracked, reverted, and collaboratively managed. GitHub hosted the repository, providing a platform for remote collaboration, backup, and documentation.

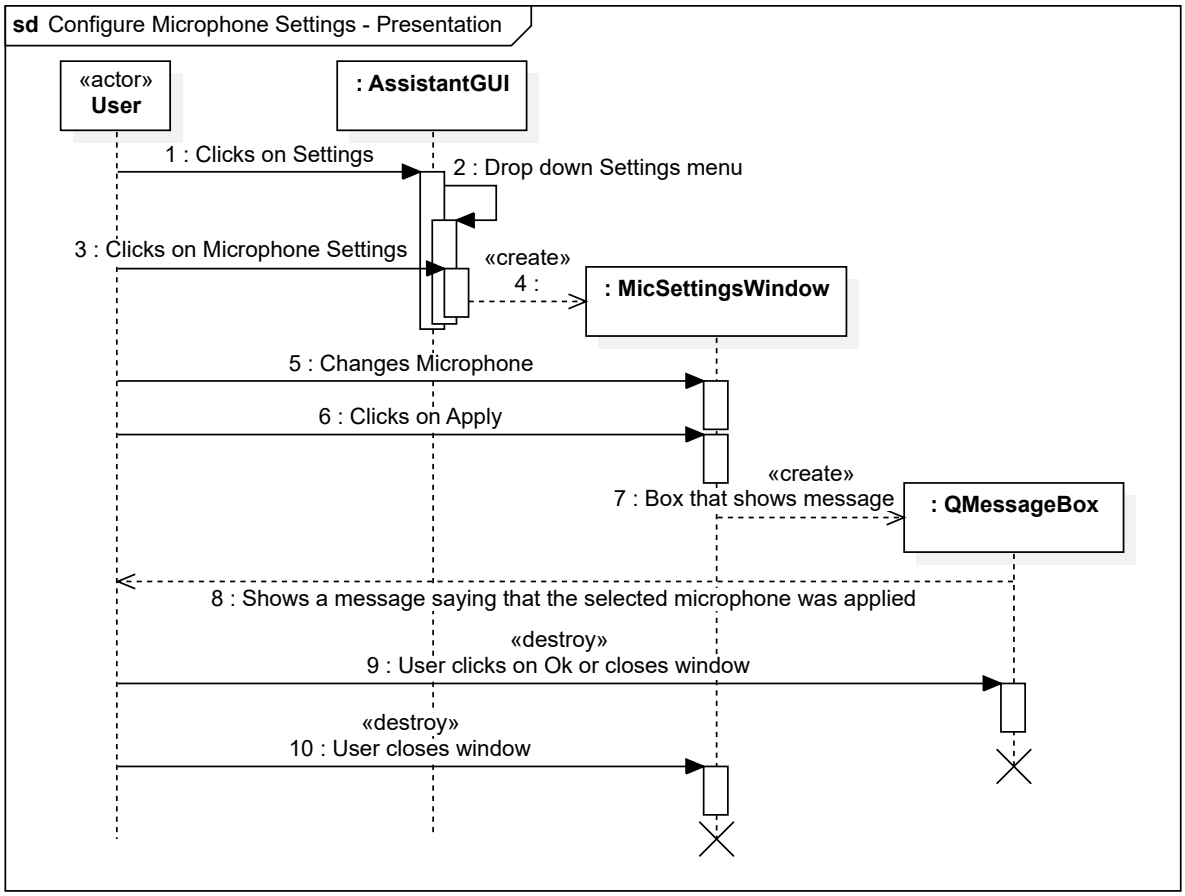

Figure 4.12 Configure Microphone Settings - Presentation Diagram

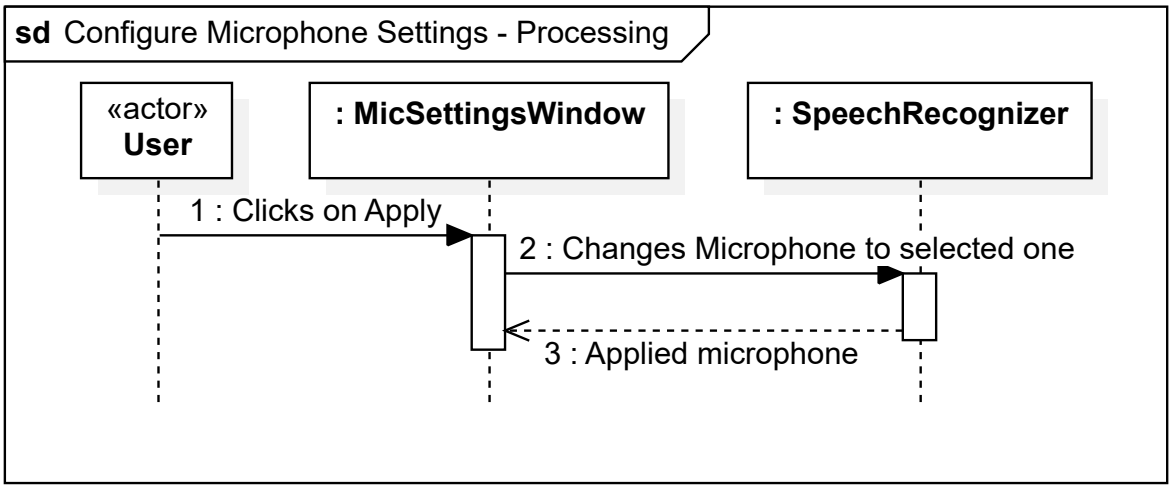

Figure 4.13 Configure Microphone Settings - Processing Diagram

4.3 Implementation Choices and Design Criteria

This section details the key engineering decisions that shaped the solution, the criteria used to evaluate alternatives, and the trade-offs accepted to meet the project's objectives. The choices are grouped by cross-cutting concerns: architecture and extensibility, interaction model, concurrency and responsiveness, external integrations, and quality attributes (robustness, safety, maintainability, and deployment).

4.3.1 Architectural Modularity and Extensibility

The first major design concern was the modularity and extensibility of the architecture. This spans over three related implementation choices: adopting a plugin-oriented modularity, enforcing interface contracts with dependency injection, and adopting a function metadata driven dispatch model.

The system was designed around a plugin-like modular structure for both base modules (e.g., speech recognition/synthesis and the LLM adapter) and functionality modules (e.g., meteorology, OS automation). Modules are not hard-wired into the core, but discovered and wired at start-up based on configuration. In practice, adopting a plugin-oriented architecture meant weighting several criteria before committing to the design: low coupling and high cohesion, so that feature additions would not require edits to core logic; dynamic evolvability, so that swapping providers (e.g., TTS/ASR engines or LLM) would not cascade changes across the codebase; and predictable contracts, so that each module type implements a clear interface. However, this approach also introduced bootstrapping complexity (discovery, and configuration validation) and required careful error reporting when a module failed to load. The trade-off was accepted because it provides long-term agility and testability.

Yet modularity alone does not ensure stability. Once the plugin structure was in place, a further concern emerged: how could modules interact in a predictable and reliable way? To address this, explicit interfaces and a dependency injection strategy were introduced. All module types adhere to abstract contracts, with the core referencing only interfaces while concrete implementations are injected at initialization. This reinforced modularity by making interaction rules explicit and was evaluated with criteria such as interchangeability, by supporting multiple implementations per role; testability, by enabling mocks and fakes for external services; and separation of concerns, by keeping the core focused on orchestration rather than vendor-specific details. These benefits came with trade-offs: stable interfaces had to be designed early, requiring anticipation of future needs; overly narrow interfaces risked rework, while overly broad ones weakened guarantees. The chosen contracts balanced minimalism with forward compatibility.

Even with modularity and contracts in place, one key challenge remained: discoverability. The system needed a way not just to plug in modules, but also to understand what each module could do. This led to the adoption of a function metadata driven dispatch model. Functionality modules describe their callable functions via metadata (name, parameters, and description), enabling the core to expose a machine-readable catalog to the LLM and provide a uniform

execution pathway for all features. This choice was guided by criteria such as alignment with the LLM, which maps user intent to structured function calls using semantic descriptions; uniformity, since all module calls share the same dispatch mechanism; and transparency, as each function advertises its contract in a consistent format. The trade-offs lay in the reliance on correct metadata definitions and the added complexity of parsing function signatures; authoring quality metadata became a new responsibility for module developers. Nonetheless, this mechanism significantly improved intent resolution and simplified routing logic.

4.3.2 Interaction Model and User Experience

The second set of design choices revolved around the interaction model and user experience, with the central question being how the assistant should communicate with the user. This led to two key implementation decisions: adopting a chat-centric interface with dual input modes, and introducing a dedicated chat handler as a facade.

The assistant was built around a chat metaphor, offering two complementary input modes: typed text or spoken utterances (converted to text internally). The output remains consistently textual, with optional text-to-speech layered on top, as shown in Figure 4.14.

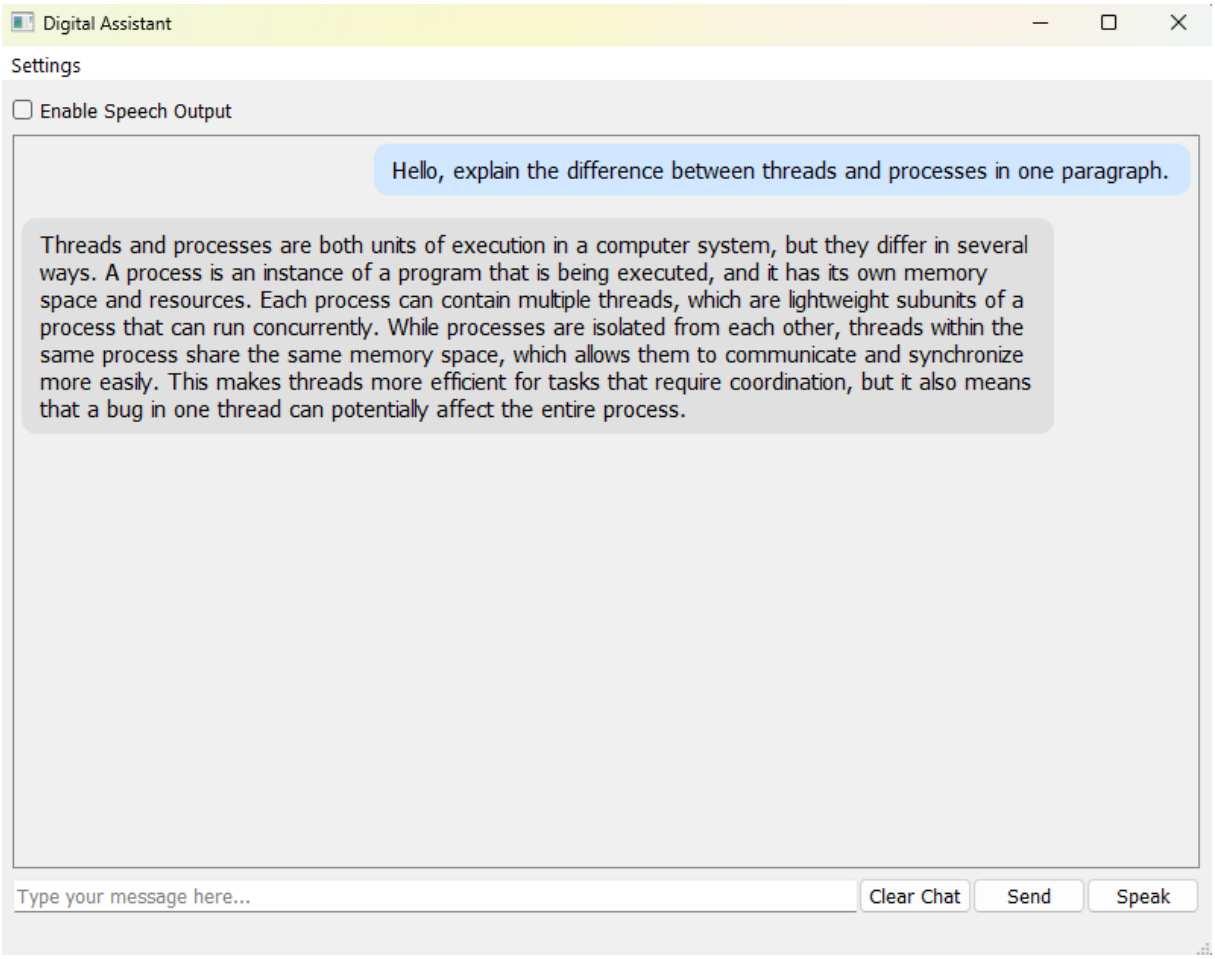

Figure 4.14 Screenshot of the GUI Main Screen

This conversational focus made a chat-style interface the natural choice, but it was not

adopted without scrutiny. It was evaluated against criteria such as accessibility, by supporting both typing and speaking to accommodate user preferences and contexts; clarity, through a single conversational timeline that avoids User Interface (UI) fragmentation; and determinism, by ensuring that regardless of input mode, the core always receives text and preserves downstream uniformity.

The following sequence diagrams (Figures 4.15 to 4.19) illustrate the internal flow of communication between the user, graphical interface, chat handler, and backend during typical interaction scenarios.

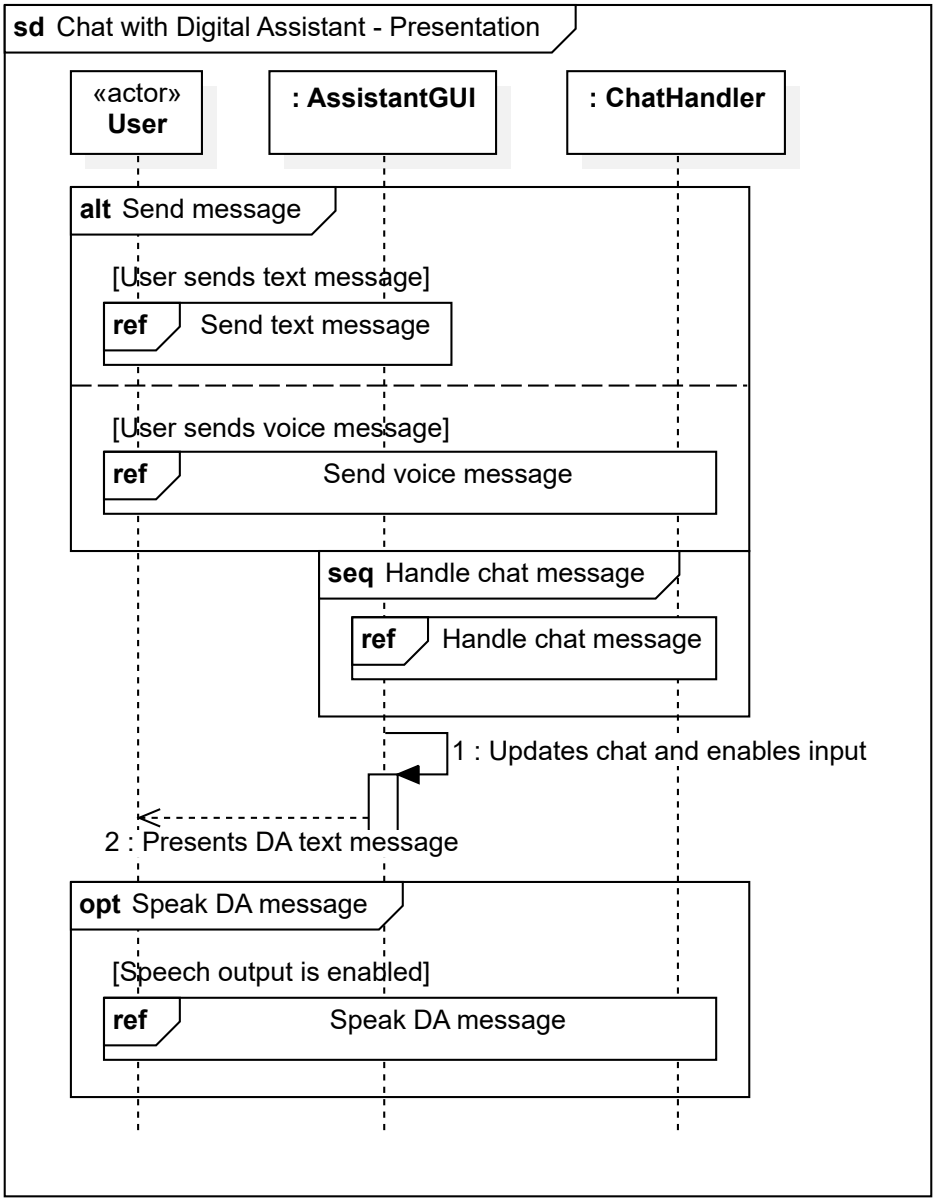

Figure 4.15 Chat with Digital Assistant - Presentation Diagram

While this approach created a familiar and straightforward interaction model, it also introduced trade-offs: richer multimodal interactions were simplified into a linear conversation flow, and voice input carried risks of latency and error in noisy environments. To mitigate these, ASR errors were surfaced as explicit user-facing messages rather than silent failures.

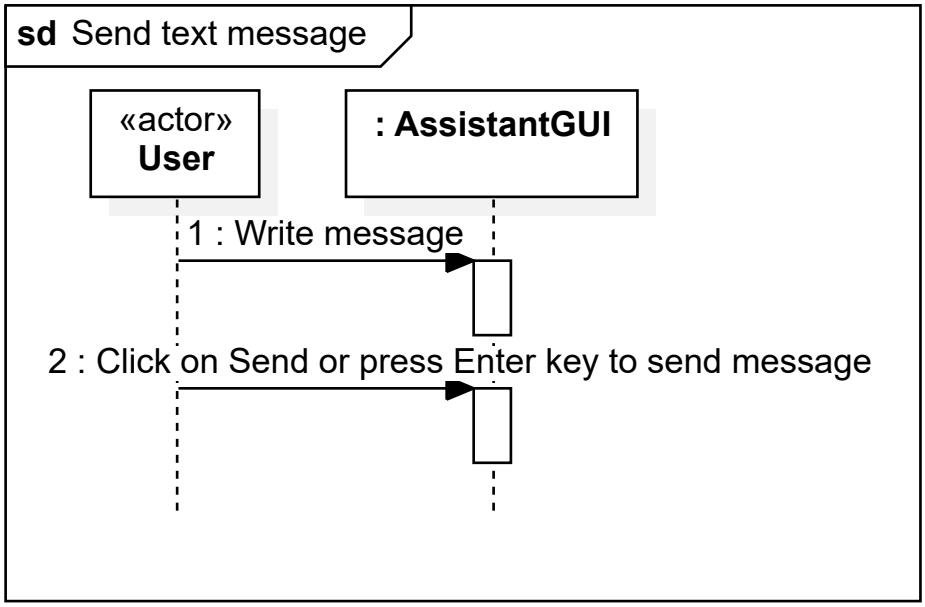

Figure 4.16 Send Text Message Diagram

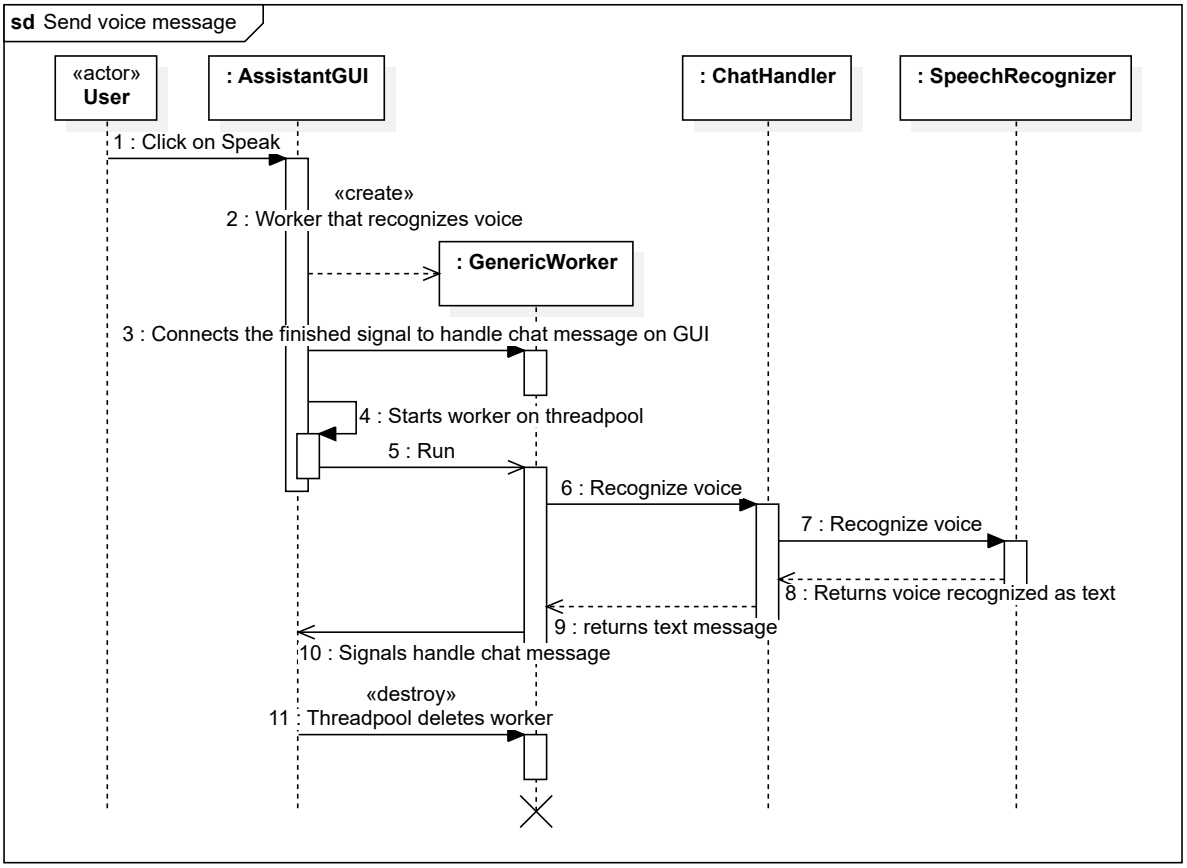

Figure 4.17 Send Voice Message Diagram

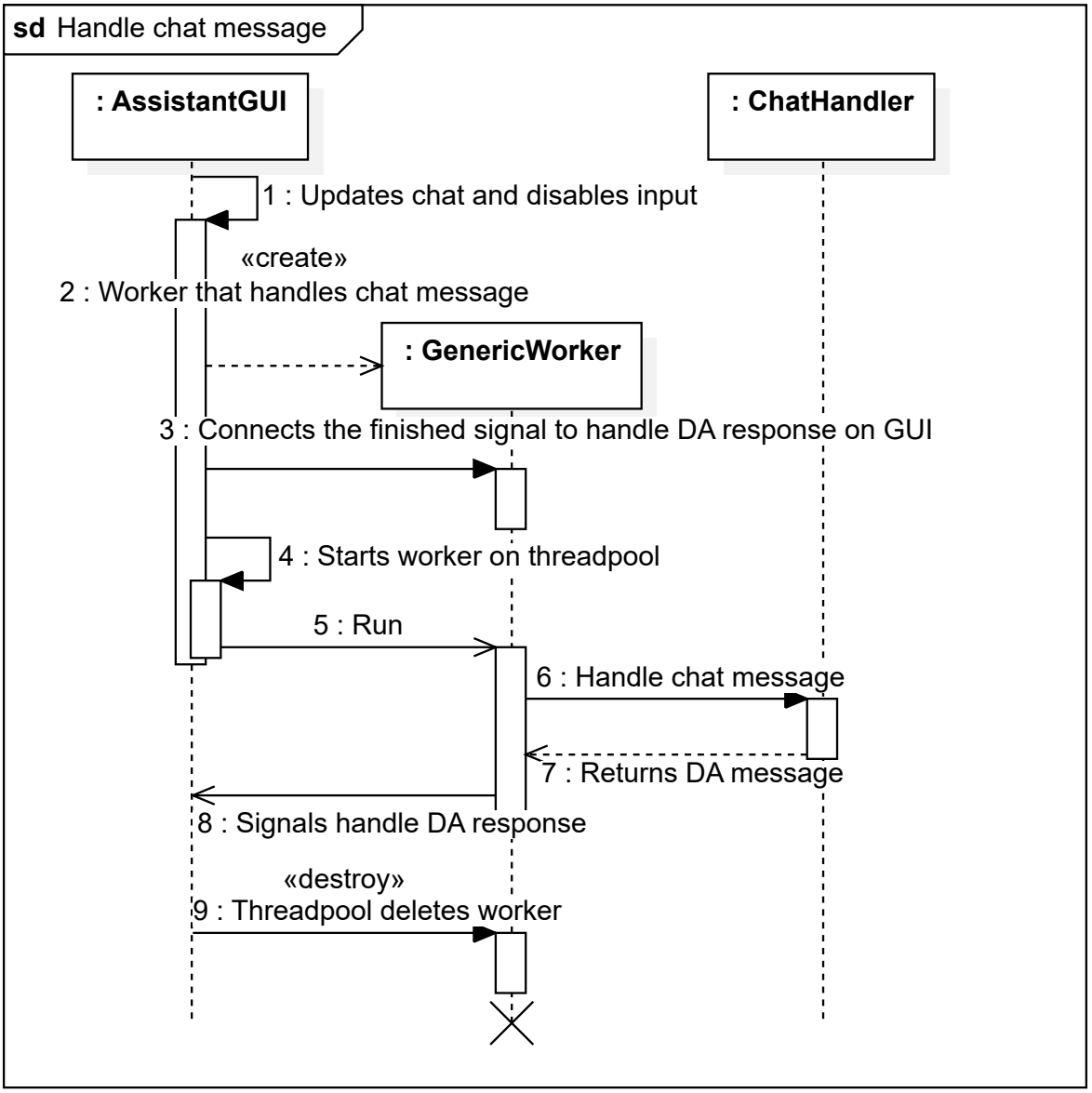

Figure 4.18 Handle Chat Message Diagram

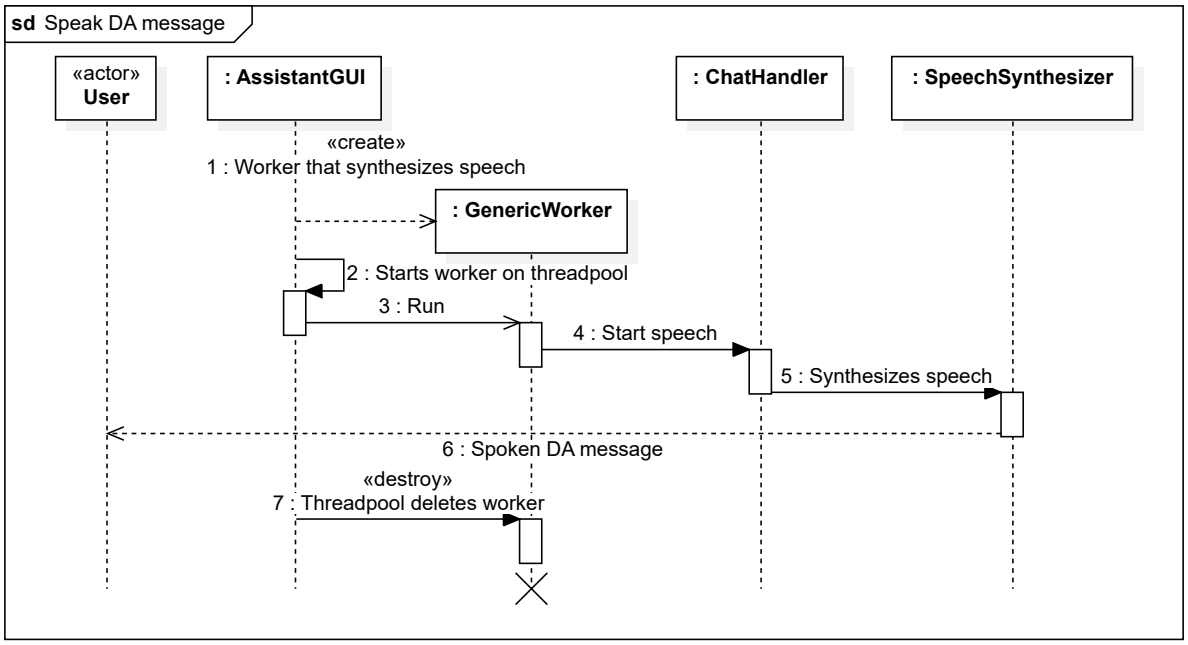

Figure 4.19 Digital Assistant Speak Message Diagram

Once the conversational interface was established, a new concern emerged: ensuring that user-facing interactions remained decoupled from backend operations. This motivated the introduction of a dedicated chat handler acting as an intermediary between the GUI and the core logic. The chat handler encapsulates all “conversation-side” operations, sending messages, triggering speech recognition, or requesting speech output, while shielding the GUI from backend complexity. The rationale was guided by criteria such as: separation of concerns, with the GUI focusing purely on presentation; consistency, by ensuring that the GUI interacts with a single, well-defined surface API; and extensibility, by enabling future UI channels, such as a Command Line Interface (CLI) or web client, to reuse the same interaction layer. While this design introduced an additional indirection layer, potentially adding overhead, it paid off by simplifying the GUI’s role, improving cohesion, and ensuring testability.

4.3.3 Concurrency, Responsiveness, and Preemption

A third design dimension concerned concurrency and responsiveness, ensuring the assistant remained fluid and controllable even when performing intensive computations or long-running tasks. This gave rise to two complementary implementation choices: asynchronous task execution in the GUI and input gating during message processing.

The first challenge was to prevent long-running operations, such as LLM calls, function execution, or speech processing, from blocking the user interface. To address this, asynchronous task execution was introduced. Any potentially blocking task is delegated to worker runnables or threads, leaving the main thread dedicated to rendering and input. This design was guided by criteria such as:

- responsiveness, with the UI updating immediately to display user input and remaining

interactive while operations are ongoing;

- isolation, by ensuring that a delay or failure in one worker does not freeze the interface;
- and simplicity, by favoring a small set of generic workers over specialized, fragmented ones.

The decision carried trade-offs, as concurrency introduced lifecycle management challenges, coordinating worker start/stop events, error propagation, and safe completion updates, but these were mitigated through consistent signaling patterns between background workers and the GUI.

While asynchronous execution protected interface responsiveness, it left open the possibility of conflicting actions being triggered simultaneously while the assistant was still processing. To ensure consistency, the design incorporated input gating: temporarily disabling inputs while a request is active, re-enabling them only once processing is complete. This mechanism was judged by criteria such as consistency, by preventing overlapping submissions that could interleave outputs; predictability, by reflecting system availability through visually disabling input fields or buttons; and simplicity, by avoiding complex reordering or queuing logic in the GUI. The trade-off was a temporary reduction in perceived interactivity: users must wait until the assistant finishes responding before issuing a new request. Nonetheless, this conservative choice prioritized state integrity and reliability over concurrency of inputs, which could have introduced race conditions and user confusion.

4.3.4 LLM-Driven Reasoning and Function Routing

Another central design concern was how the assistant interprets user input and determines whether to respond directly or to invoke a specific functionality. This aspect centered on the role of the LLM as a semantic mediator and the balance between deterministic routing and natural conversational fluency.

The first choice was to position the LLM not only as a generator of natural responses but also as a mediator capable of mapping user requests to function calls. This was made possible by exposing a catalog of available functionalities through structured metadata, which the LLM could then use to resolve intent into explicit actions. The suitability of this approach was evaluated through criteria such as generality, by handling open-ended natural language queries without rigid rules; explainability, since function calls are explicit and traceable when generated by the LLM; and adaptability, as new functionalities become immediately available through updated metadata without retraining the model. However, this introduced trade-offs: prompt engineering became critical to guide reliable function selection, and the inherent variability of LLM outputs required careful validation of chosen functions and arguments.

Figures 4.20 and 4.21 expand on this process by illustrating how the backend and the LLM collaborate to interpret user requests and route them to the appropriate functionality modules.

Robustness was reinforced by fallback strategies, such as producing clear user-facing errors when no suitable function was found.

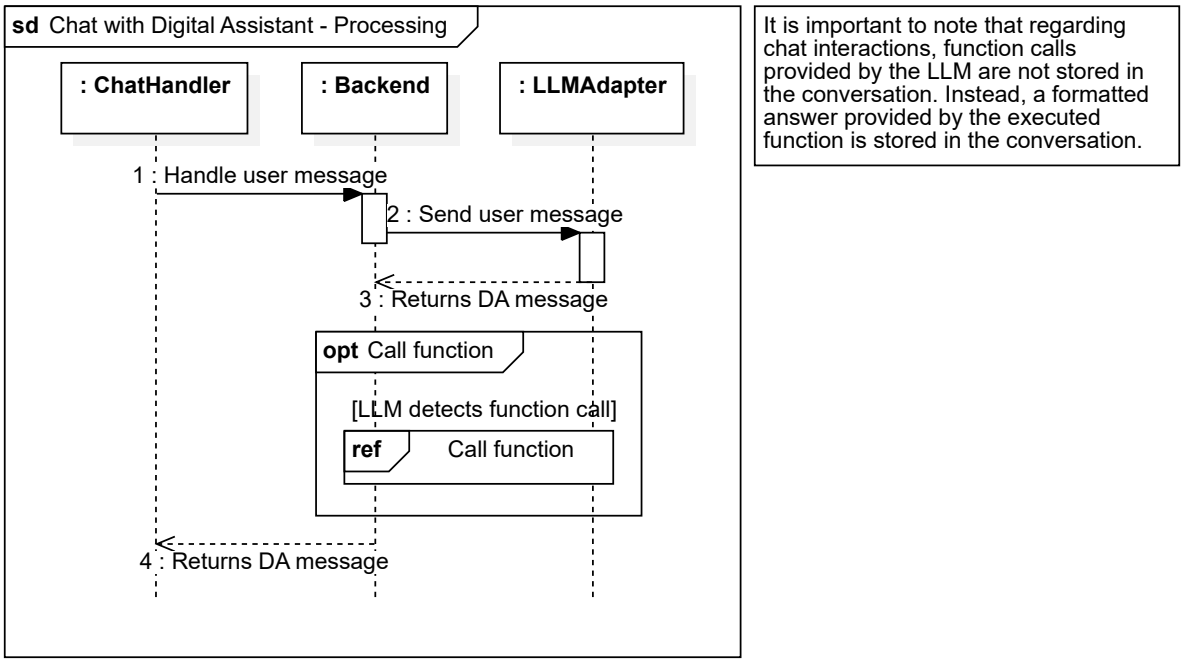

Figure 4.20 Chat with Digital Assistant - Processing Diagram

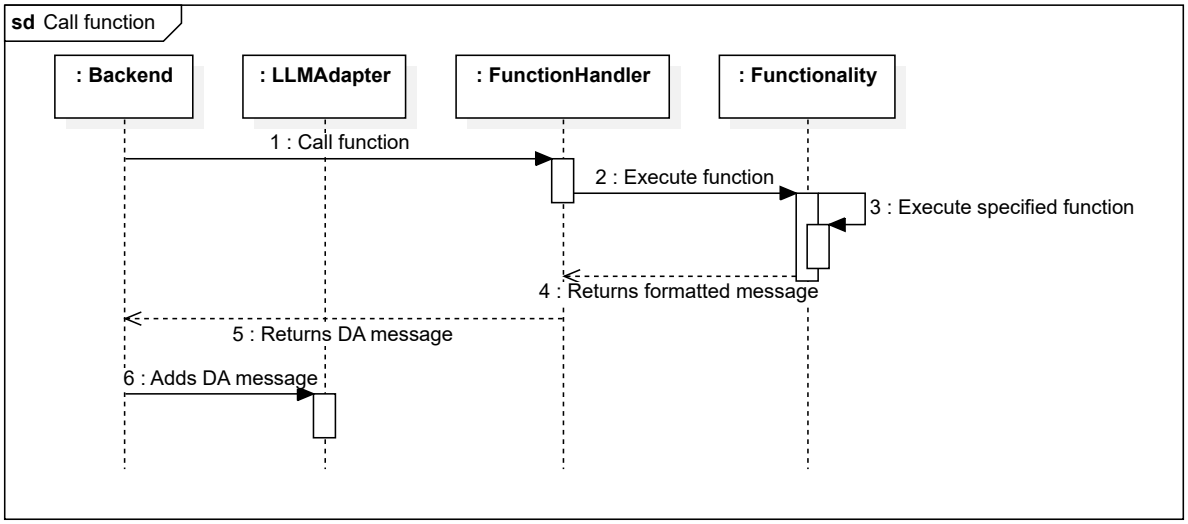

Figure 4.21 Call Function Diagram

Once the LLM was embedded as the central mediator, the next consideration was how predictable or creative its responses should be, particularly when differentiating between natural conversational replies and functional invocations. The balance leaned toward deterministic behavior in function selection, ensuring that repeated requests yield consistent routing outcomes, while allowing more flexibility and variation in natural language phrasing to maintain fluency. This balance was evaluated through criteria such as reliability, by minimizing unexpected or inconsistent behavior; user trust, by providing transparency and predictability in assistant actions; and readability, by ensuring that generated responses remain clear and helpful. The trade-offs here reflected a tension between utility and naturalness: while strict determinism may reduce the impression of spontaneity, it reinforced the assistant's role as a dependable utility-driven tool.

4.3.5 External Integrations and Data Handling

Beyond its internal orchestration, the assistant depends on external services and system-level integrations to fulfill user requests. This design dimension required careful consideration of reliability, error handling, and safety when interacting with third-party API and OS features.

For external API, the primary concern was how to manage variability in responses and potential unavailability. Services such as weather providers were accessed through a uniform HTTP request layer that normalized outputs before presenting them to the user. This decision was evaluated against criteria such as availability, ensuring that the system remains robust when external services are down or slow; schema stability, since data formats may differ or evolve and require consistent handling; and user readability, by clearly distinguishing between "no results" and "service unavailable" cases. These criteria justified the adoption of normalization and structured error messages, though they introduced trade-offs, notably the need for additional parsing logic and the unavoidable dependency on third-party service availability. Still, the benefit was clear: users receive consistent and meaningful feedback regardless of external variability.

While API raised concerns about reliability and clarity, system-level automation posed different challenges tied to safety and predictability. The assistant integrates with the OS to perform actions such as launching applications or locating files. To mitigate risks, the design constrained these tasks to safe, non-destructive operations, relying on shortcuts and controlled lookup strategies rather than direct system manipulation. Evaluation criteria included safety, by avoiding destructive or intrusive commands; predictability, by ensuring that when multiple results match, the user is explicitly informed rather than the system guessing; and tolerance to imperfect input, by accepting partial matches or misspellings through fuzzy matching. The trade-offs were acknowledged: using shortcuts may omit certain applications without menu entries, and conservative safeguards limit functionality in favor of reliability. Nevertheless, this cautious stance aligned with the goal of creating a trustworthy and user-safe assistant.

4.3.6 Quality Attributes and Cross-Cutting Concerns

Alongside feature-specific design decisions, the implementation was guided by a set of cross-cutting quality attributes that ensure the assistant is not only functional but also maintainable, observable, performant, secure, and deliverable in practice. These attributes shaped choices that cut across all layers of the architecture.

Maintainability was supported through a layered structure and interface-driven design, ensuring that modules remain self-contained with explicit boundaries. This allowed local reasoning: changes to one module rarely require edits elsewhere, and improved refactorability by enabling components to be swapped with minimal disruption. The trade-off was the added abstraction and boilerplate required for interfaces and dependency wiring, but this cost was accepted as an investment in long-term adaptability.

To complement maintainability, observability was emphasized to make the system transparent to both developers and users. Errors propagate as structured messages that surface directly in the conversational interface (e.g. “no matching applications found”), while internally, exceptions are routed through well-defined completion signs. Criteria such as user feedback, by clearly communicating what went wrong; debuggability, by linking failures to specific modules or providers; and containment, by ensuring that a module failure does not crash the UI, guided this choice. The trade-off was the additional effort needed to craft user-readable error messages consistently, improving trust and traceability.

Once robustness in error handling was established, performance and responsiveness became the next priority. Immediate UI feedback, by echoing user input before processing; background execution of heavy tasks; and decoupled voice I/O ensured a fluid user experience. The key criteria were responsiveness, since the UI never “hangs”; throughput, as the system can accept new input once a prior request finishes; and smooth concurrency, because long tasks do not block other operations. The cost was the complexity of synchronizing background tasks and handling state transitions, but this was mitigated with disciplined signal/slot coordination.

Security and privacy considerations were incorporated to safeguard user trust. Credentials (such as API keys) were externalized and loaded securely at runtime, while user utterances were kept in memory and only transmitted to external services when required. These decisions were evaluated through confidentiality, by ensuring keys are never hardcoded; least privilege, by restricting modules to access only the data they need; and transparency, by allowing users to be informed about which services were contacted. The trade-off was the operational overhead of managing secure configurations, but this ensured resilience against accidental leaks.

Finally, deployment considerations shaped the assistant’s delivery model. Targeting Windows desktop as the initial platform allowed for packaging as a standalone executable, minimizing setup requirements for end users. Criteria such as accessibility, since a user can launch the assistant without additional dependencies; consistency, as a single target environment simplifies automation reliability; and supportability, by narrowing the platform scope to reduce maintenance complexity, guided this decision. The limitation, of course, is reduced portability: cross-platform support is deferred to a future enhancement. This trade-off aligned with the immediate goal of

providing a stable, usable assistant in its target environment.

4.3.7 System Initialization Process

During the system's initialization phase, when all layers are dynamically put together to form a functional whole, these architectural and design principles actually come together. By coordinating the loading of the base and functionality modules through the module loader at startup, the entry routine confirms that the language model supports function calling and compiles all of the functions into a single schema. The language model adapter and backend, which together control reasoning and functionality execution, are then instantiated after the system prompt that specifies the assistant's operational context has been generated. In order to maintain a clear division between presentation and logic, the chat handler is then deployed to connect the backend and the graphical user interface. Lastly, the main window is displayed and the event loop that maintains constant user interaction is started when the PyQt-based graphical user interface is launched within its application environment. By connecting the presentation, core logic, and module layers into a unified, interactive digital assistant, this startup sequence operationalizes the previously discussed design goals, modularity, separation of concerns, and responsiveness.

4.4 Discussion of Key Implementation Aspects

The DA's implementation revealed a number of factors that were especially important to its success and that provide more general lessons for systems of a similar nature. These aspects cover architecture, concurrency, interaction design, external dependencies, and development practices. This section is divided into thematic subsections that each concentrate on one of the previously mentioned important technical or design aspects of the project to offer a more coherent and structured analysis. Together, they describe how engineering choices were made based on theoretical concepts, which in turn influenced the behavior of the system and the user experience.

4.4.1 Architectural Design and Modularity

One of the most impactful aspects of the implementation was the adoption of a layered, plugin-based architecture. Although conceptually straightforward, its practical effects were far-reaching. Separating the system into a presentation layer, a lightweight core, and modular plugins not only clarified responsibilities but also enabled incremental evolution throughout development.

This separation became particularly valuable when integrating diverse modules such as speech recognition, text-to-speech, weather services, and system automation. Each module could be developed, tested, and even replaced independently, without disrupting the rest of the system. The use of contracts proved decisive in enforcing clear communication between components and creating a uniform language for extension.

A key lesson learned was that interface-first design requires discipline early on. Anticipating all future module behaviors was not always possible, and some interfaces had to be iteratively refined as new functionalities were added. Still, the benefits were clear: this architectural discipline prevented tight coupling and ensured that the assistant could grow without accumulating unmanageable complexity.

This modular structure also reflects the principles of component-based software engineering [42], in which each module functions as an independent, replaceable unit governed by explicit contracts. Such an approach improves scalability and maintainability, two essential attributes for intelligent assistants that must adapt to new API, LLM models, or OS environments over time.

4.4.2 Concurrency and Responsiveness

Another crucial aspect was the management of concurrency and responsiveness. Long-running tasks such as LLM calls, API requests, or speech operations could easily block the GUI, undermining the assistant's interactivity. To address this, asynchronous task execution and worker threads were employed to delegate these tasks, keeping the main interface responsive.

This strategy allowed the conversational window to update immediately after input submission, reinforcing the perception of fluidity. However, implementing concurrency introduced challenges in thread lifecycle management, synchronization, and error propagation. Coordinating worker completion and handling user actions during ongoing operations required careful control to avoid inconsistencies.

A simple but effective solution was input gating: temporarily disabling user input while a response was being generated. Although this reduced interactivity slightly, it prevented race conditions and overlapping outputs, which would have been far more damaging to usability. This design decision aligns with human–computer interaction principles emphasizing perceived responsiveness as a key determinant of usability [43]. By providing consistent visual feedback and predictable state changes, the assistant maintained user confidence even during background computations.

Overall, responsiveness was treated not just as a technical concern but as an experiential one. Ensuring that the assistant felt "alive" and reliable required balancing concurrency mechanisms with clear communication of system state.

4.4.3 User Interaction and Experience

The adoption of a chat-centric interaction model also defined much of the assistant's identity. A unified conversational timeline, where both typed and spoken inputs are represented as text, simplified downstream processing and created a consistent user experience. This design gave users a single conversational space regardless of the input modality, making interaction more intuitive and cohesive.

This approach had several advantages: it reduced branching logic in the backend, avoided duplication of effort across input types, and provided a single place for users to monitor dialogue

history. At the same time, it highlighted the importance of transparent feedback; speech recognition results, including errors, had to be displayed clearly so that users could understand when the assistant misheard a command rather than silently failing.

The introduction of a chat handler as an intermediary between the GUI and backend also had a strong impact on usability. Although it added one more abstraction layer, it isolated the GUI from backend complexity and allowed it to focus solely on presentation and input capture. This separation was particularly valuable for managing conversational state consistently, since all user interactions passed through a single, coherent interface.

This design pattern resembles the Model–View–Controller (MVC) paradigm [44], which emphasizes decoupling presentation from logic to enhance modularity and testability. By maintaining this separation, the assistant's architecture remains extensible and adaptable to new front-end formats, such as command-line or web interfaces, without altering backend logic. This illustrates how careful software structure directly supports consistent and scalable user experiences.

4.4.4 External Integrations and Defensive Design

Integrating external services and system-level functionality introduced both opportunities and risks. The OpenWeatherMap API enriched the assistant's capabilities with weather and pollution data, while OS-level automation extended its practical utility beyond purely conversational tasks. However, these integrations also exposed the system to unreliable network connections, variable response formats, and potential ambiguity when handling user queries.

To mitigate these risks, normalization and structured error handling were implemented to ensure that inconsistencies or failures did not propagate to the user. By translating raw API errors into clear, meaningful feedback (for example, distinguishing between "no results found" and "service unavailable"), the assistant maintained robustness and transparency.

For OS-level automation, safeguards were introduced to handle ambiguous or uncertain operations. When multiple applications matched a fuzzy search query, the assistant presented the user with options rather than making assumptions. Similarly, only non-destructive commands were permitted, ensuring that automation remained predictable and safe.

A central part of this mechanism was the use of fuzzy string matching through the *RapidFuzz* library [41], which allowed the system to handle imperfect or misspelled inputs efficiently. The algorithm computes similarity scores using metrics such as Levenshtein distance, enabling robust and efficient matching even in uncertain user scenarios. This implementation reflects defensive programming principles [45], where systems are designed to anticipate errors and handle them gracefully rather than assuming ideal inputs.

Reliability, in this context, was achieved not through external dependencies themselves but through the assistant's ability to manage their unpredictability. Encapsulating third-party integrations in self-contained modules with well-defined contracts preserved overall system stability and ensured that external failures did not compromise core functionality.

4.4.5 Cross-Cutting Qualities and Lessons Learned

Beyond specific modules or functionalities, several cross-cutting qualities defined the assistant's overall behavior and maintainability. These qualities, maintainability, responsiveness, robustness, transparency, and security, guided decisions across all layers of implementation.

Maintainability was prioritized through an interface-first architecture and strict separation of responsibilities. This ensured that new modules could be added or existing ones modified with minimal impact on the rest of the system. Dynamic loading and modular contracts reinforced the principle of "local reasoning," allowing developers to focus on a single component without needing to understand the internals of others.

Responsiveness, as discussed earlier, was a recurring goal, particularly when dealing with long-running operations such as LLM inference or API requests. Asynchronous task execution and input gating mechanisms were essential in maintaining fluidity, ensuring that the user interface acknowledged input immediately even while background processing continued.

Robustness and transparency were equally critical. Rather than silently failing or exposing raw errors, the assistant consistently provided user-readable explanations. This strengthened its role as a cooperative and trustworthy partner, informing users not just when an action succeeded but also why it failed when necessary.

Security and privacy, though limited in scope for this implementation, were nonetheless treated as integral. Sensitive data such as API keys were loaded securely at runtime, and modules were designed with the principle of least privilege, accessing only the data required for their operation. These measures, though modest, reflect good practices that would scale to more complex deployments.

Furthermore, the assistant's emphasis on reliability, feedback, and transparency aligns with established human–AI interaction principles [37], which emphasize making system behavior understandable, predictable, and trustworthy. By surfacing system states and providing contextually clear feedback, the assistant reinforces user confidence and usability.

Finally, deployment considerations helped define the project's practical scope. Targeting Windows as the initial platform simplified packaging and distribution, ensuring stable integration with system APIs and a controlled deployment environment. Although this limited immediate cross-platform compatibility, it provided a solid foundation for future portability.

In summary, the implementation of the digital assistant demonstrates how deliberate architectural, modular, and defensive design choices can collectively enhance reliability, usability, and long-term maintainability. By combining solid software engineering principles with practical trade-offs, the system achieves a level of robustness and extensibility that reflects the qualities expected from modern, intelligent assistant solutions.

Chapter 5

Experimental Evaluation

This chapter presents the evaluation of the developed DA, focusing on verifying that it meets the functional and non-functional requirements defined in Section 3.2. The evaluation is organized into three main parts. First, Section 5.1 details the experimental configuration and testing setup, describing the hardware, software, and methodological parameters used to measure system performance. Next, Section 5.2 presents the quantitative results, focusing on execution times for the functionality modules and the response latency and semantic correctness of the integrated LLMs. Finally, Section 5.3 provides a qualitative assessment of the remaining functional and non-functional requirements, analyzing the graphical interface, speech components, modularity, robustness, privacy, and deployment aspects. Together, these sections ensure a comprehensive evaluation of both the system's technical performance and its practical usability within the intended application context. Detailed information about the test parameters, utterances, and raw results supporting this evaluation can be found in Annex A, while the complete usability questionnaire administered to participants is reproduced in Annex C.

5.1 Experimental Configuration and Settings

The evaluation was conducted on a personal laptop, specifically a Lenovo IdeaPad 3 (6º Generation) 15ALC6-401 15.6" Arctic Grey running Windows 11 (64-bit). The system is equipped with an AMD Ryzen 7 Mobile 5700U processor, 16GB of RAM, an integrated graphics card, and uses Wi-Fi through a stable connection to support API requests and LLM communication. The software environment uses Python 3.10 with all dependencies managed through *pip*.
Two main types of tests were performed:

- one evaluating execution times for the functionality modules;
- and the other assessing execution times and correctness for the LLM.

For the functionality modules, the objective was to measure end-to-end response time, from the assistant triggering the function to its final result value. Each module was tested with 30 parameter instances (except the meteorology module, where 10 different locations were selected and used across the three supported parameter types: latitude/longitude, city name with optional

country/state code, and zip code with country code). For the LLM, the evaluation measured both response latency and correctness. Correctness was defined as the model's ability to (1) correctly distinguish between general conversation and function calls, and (2) extract the correct set of parameters from the user utterance when a function call was required. As with the modules, thirty test utterances were created for each functionality and an additional thirty utterances for general conversation. To make function calling possible, the models were either prompted with the available function descriptions (system prompt method) or provided with JSON schemas when using native tool calls, depending on the experimental setup. By combining these tests, the evaluation captures both the system's responsiveness and its semantic accuracy in interpreting user requests.

5.2 Performance Evaluation

This section presents and interprets the results of the experimental evaluation, focusing on two key takeaways: (1) the execution time of functionality modules and the response time, and (2) semantic correctness of the LLM. Together, these results measure the responsiveness and reliability of the assistant, reflecting its ability to provide timely and accurate answers to user requests, evaluating the non-functional requirement for responsiveness (NFR2).

5.2.1 Execution Time of Functionality Modules

Table 5.1 reports the execution time for each functionality module, measured from the moment the function was invoked until a result was returned. For the meteorology functionality, measurements were taken with and without formatting, where formatting refers to converting the raw JSON response into a human-readable message for display in the conversation interface.

Table 5.1 Execution Time of Functionality Modules (in seconds), thirty tests per module

Functionality	Functions	Format	Mean	Median	Min	Max
Meteorology	`get_current_weather`	No	0.29	0.29	0.27	0.32
		Yes	0.29	0.29	0.28	0.3
	`get_forecast`	No	0.33	0.33	0.32	0.35
		Yes	0.34	0.33	0.31	0.55
	`get_air_pollution`	No	0.29	0.29	0.27	0.32
		Yes	0.29	0.29	0.28	0.31
OS	`launch_application`	N/A	1.31	1.25	1.02	1.87

Across all meteorology functions (`get_current_weather`, `get_forecast`, and `get_air_pollution`), the mean execution times were consistently low, generally around 0.3 seconds, with minimal deviation between formatted and unformatted outputs. This indicates that the additional processing overhead of producing human-readable responses is negligible. Launching applications exhibited higher variance (mean ≈ 1.31 seconds, max 1.87 seconds),

likely because this task involves querying the file system, retrieving potential matches, and performing fuzzy matching before deciding whether to launch or return multiple options. Nevertheless, even the upper-bound latencies remain well within a reasonable range for interactive use.

Overall, these results confirm that the functionality modules are fast enough to support a smooth user experience defined in NFR2 (timely responses).

5.2.2 LLM Response Time and Correctness

The performance of the LLM was assessed along two dimensions: response latency and semantic correctness. Table 5.2 summarizes these results for three models (Llama 3.1 8B Instruct, Gemini 2.0 Flash, and Qwen 2.5 7B Instruct), where one is tested on two different providers (Llama on Awan LLM and Hugging Face) and another is tested with different function calling methods (through function description on system prompt and through tool calls [where function schemas are explicitly defined according to OpenAI's specification]). Latency is reported as mean, minimum and maximum execution time across 150 utterances (30 per functionality plus 30 general conversation utterances). Correctness is reported both per-function and as a total number of correct inferences out of 150.

These results reveal clear differences between the models and the function-calling strategies. Gemini achieved the lowest mean latency across all categories (≈ 1.0 seconds for function calls, 2.39 seconds overall), offering near-real-time responses while maintaining very high correctness ($148/150$). Qwen (system-prompt-based) was the fastest in absolute terms ($\overline{x} = 1.37$ seconds overall), but exhibited more semantic errors, particularly for `launch_application` utterances ($8/30$ correct). Many of these errors stemmed from systematically setting the `is_sure_after_multiple_matches` parameter incorrectly. In this setup, the function specification did not declare a default value for this parameter, leaving the model to infer it, sometimes unnecessarily adding fields or setting the wrong value.

The new Qwen with tool calls configuration addresses this by explicitly providing a default value for the parameter through the tool schema. This change sharply improves the accuracy ($140/150$), including perfect handling of `launch_application` ($30/30$ correct). Although latency increases slightly ($\overline{x} \approx 2.32$ seconds overall), it remains well within interactive thresholds, demonstrating that schema-based function calling removes ambiguity and strengthens robustness.

Both function call strategies are valid and should be selected based on system constraints, specifically context token cost, because both consume tokens in the same way [46]. The function descriptions injected into the system prompt consume fewer tokens when parameter lists are shared between multiple functions, since descriptions can reference other functions (for example, 'same parameters as [function]'). In contrast, schema-based tool calls require the parameters of each function to be fully defined and cannot reference other definitions, which can increase token usage and cost, especially in large toolsets. Future work should include token-level cost analysis to quantify this trade-off.

Table 5.2 Execution Times and Correctness of the LLM. Crct = Correctness, N = Normal conversation, CW = Get current weather, 5DF = Get forecast, AP = Get air pollution, LA = Launch Application, TCrct = Total Correctness.

Models	Metrics	N	CW	5DF	AP	LA	TCrct
Awan LLM's Llama	$\overline{x}$	15.76	3.73	3.93	3.83	4.06	140/150
	min	1.45	2.74	2.82	3.22	2.59	
	max	34.11	14.26	14.31	14.29	21.05	
	Crct	30/30	28/30	28/30	27/30	27/30	
HuggingFace's Llama	$\overline{x}$	1.82	0.79	0.72	0.71	0.61	138/150
	min	0.40	0.62	0.51	0.49	0.42	
	max	3.60	1.20	1.22	1.39	1.17	
	Crct	29/30	29/30	28/30	27/30	25/30	
Gemini	$\overline{x}$	2.39	1.01	0.96	0.99	0.92	148/150
	min	0.62	0.62	0.60	0.73	0.54	
	max	6.66	2.10	2.01	1.95	2.03	
	Crct	30/30	29/30	29/30	30/30	30/30	
Qwen	$\overline{x}$	1.37	1.02	0.93	0.90	0.94	116/150
	min	0.56	0.73	0.63	0.67	0.66	
	max	3.18	2.05	1.77	1.79	2.07	
	Crct	24/30	30/30	28/30	26/30	8/30	
Qwen with tool calls	$\overline{x}$	2.32	0.88	1.25	0.95	0.72	140/150
	min	0.67	0.63	0.71	0.65	0.57	
	max	4.58	1.60	10.76	1.44	1.28	
	Crct	30/30	28/30	28/30	24/30	30/30	

Llama shows a strong dependency on the provider infrastructure. When accessed via the Awan provider, the mean response times were very high (15.76 seconds) with outliers above 30 seconds, making it impractical for interactive use despite solid correctness (140/150).

When tested via Hugging Face, latency improved dramatically ($\bar{x} \approx 1.82$ seconds) while correctness remained comparable (138/150), confirming that the model architecture itself is competitive and that the previous latency was largely provider-induced.

In general, Gemini offers the best balance between speed and correctness out of the box, Qwen with tool calls reaches near-Gemini correctness with competitive latency, and Hugging Face's Llama is a viable alternative when served by a low-latency provider. These findings reinforce the importance of both model choice and deployment configuration when targeting responsive digital assistants.

5.2.3 Implications for Requirements Validation

From these results, several functional and non-functional requirements can be validated, confirming that the proposed system behaves as intended under realistic conditions:

- **FR4 (Interpretation via LLM) -** All evaluated models demonstrated the ability to distinguish between general conversation and function calls. The improvement observed with Qwen when switching from system-prompt function descriptions to schema-based tool calls confirms that accurate parameter extraction can be further optimized by using structured function definitions. Both approaches are viable and interchangeable; however, schema-based calling proves more robust in practice, as it eliminates ambiguity about default values and parameter inclusion.

- **FR5 (Functionalities provided) -** The experiments confirm that all supported functionalities can be invoked naturally through language input and that their parameters are correctly interpreted when defaults are defined. The results further highlight the importance of clear function interface design and well-documented metadata, ensuring that LLMs can consistently map user intent to the correct executable functionality.

- **NFR2 (Responsiveness) -** The use of fast model providers (such as Gemini or Hugging Face's Llama) ensures response latencies within an acceptable range for interactive applications. This satisfies the system's responsiveness requirements, demonstrating that both the assistant's architecture and backend configuration successfully minimize user-perceived delays.

Additionally, these findings suggest that future evaluations should consider measuring token usage and cost implications associated with each function-calling strategy. System-prompt injection is typically more token-efficient when functions share parameter definitions, whereas schema-based tool calls favor robustness and correctness at the expense of potentially higher token consumption. This trade-off becomes particularly significant in large-scale or cost-sensitive deployments, where prompt size and operational cost must be carefully optimized.

5.3 Qualitative Validation of Remaining Requirements

This section provides a user-centered evaluation based on a structured questionnaire, as shown in Annex C, to supplement the quantitative analysis. The objective of such a questionnaire is to understand the perceived usability, clarity, and practical effectiveness of the DA, and to relate those observations to the functional and non-functional requirements defined in Section 3.2. The questionnaire covered four task groups based on the functionalities (current weather, 5-day forecast, air pollution, and launch applications) plus the conversational ability of the DA, and a post-test System Usability Scale (SUS) evaluation.

5.3.1 Participants Characterization

The study was finished by nineteen participants. According to Figure 5.1, the majority of respondents were between the ages of 25 and 34, with additional participation from the 18 to 24, 35 to 44, 45 to 54, and 55+ age groups.

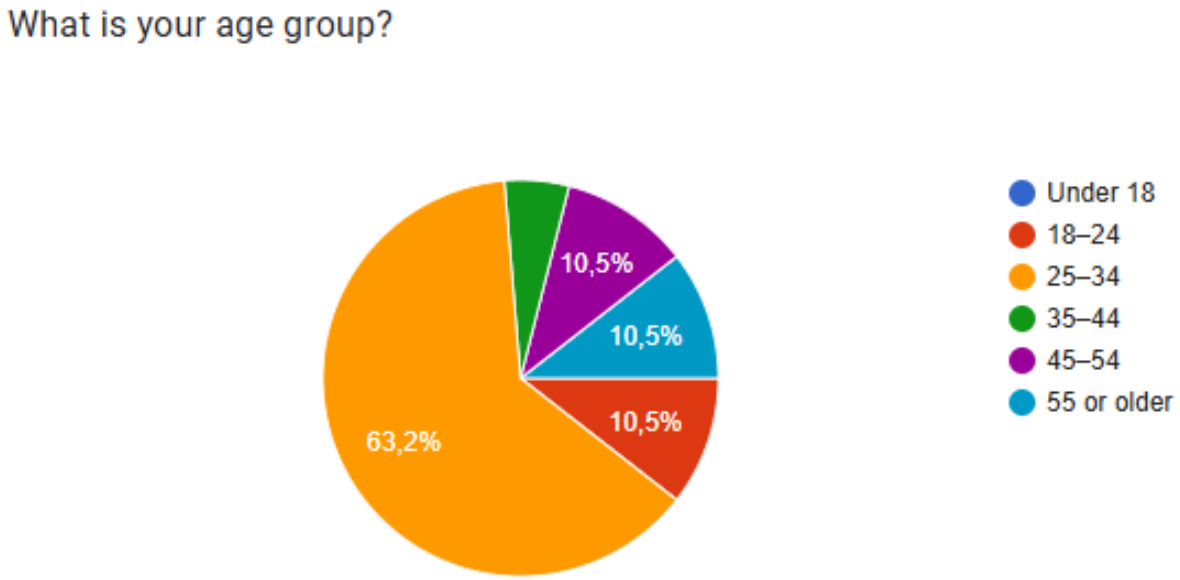

Figure 5.1 Distribution of participants' age groups in the questionnaire

There was an almost equal number of male and female users (10 men and 9 women). As demonstrated in Figure 5.2, self-reported familiarity with AI assistants ranged from largely moderate to high (3–5 on a five-point scale, mean score of 3.37), suggesting that the majority of participants had some prior knowledge or experience with similar systems.

This profile makes sense to the wanted evaluation, compared to a profile where the majority of users are not experienced with similar systems, because the participants already have a baseline of what a DA is and what it has.

5.3.2 Tasks and Questionnaire

Participants completed four representative tasks that were in line with supported functionalities (FR5): (1) get the current weather; (2) get a five-day forecast; (3) ask about air pollution; and

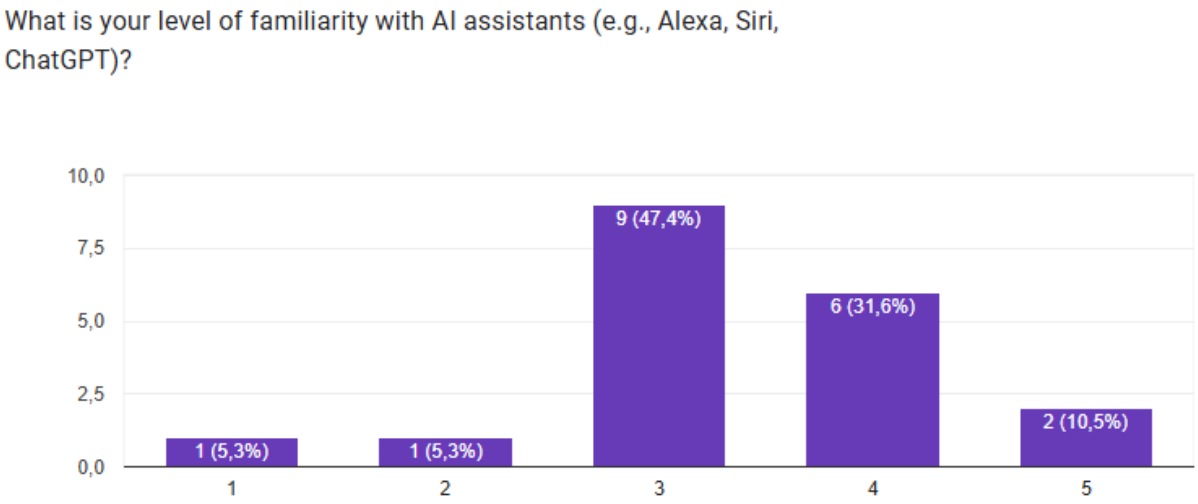

Figure 5.2 Self-reported familiarity with AI assistants

(4) start a local application. For each task, the questionnaire gathered opinions on correctness, clarity, ease of use, and encountered issues. Voice interaction questions evaluated speech output (FR2.2, FR3.1) and microphone recognition and selection (FR2.1, FR3.2). Relevance, naturalness, and follow-up comprehension were examined in a conversation following the aforementioned tasks (FR4). Finally, in order to provide an overall usability rating (NFR1), participants filled out the SUS.

5.3.3 Results and Requirement Mapping

Weather, Forecast, and Air Pollution (FR5.1). Across participants, information retrieval for current weather, 5-day forecast, and air quality was generally successful and easy to follow (an example is shown in Figure 5.3).

Responses were described as clear and well structured, however, a recurring issue concerned the speech output reading measurement units and symbols literally (e.g., “µg/m^3” pronounced as “micro g slash m super script three”), which reduced clarity when text-to-speech was enabled (FR2.2, FR3.1). Overall, the content and presentation of meteorology-related results support FR5, with improvement opportunities in text-to-speech rendering of scientific units.

Speech Input and Output (FR2.1, FR2.2, FR3.1, FR3.2). Many users found voice recognition to be reliable, but several reported difficulty selecting the correct microphone when multiple similarly named devices appeared (FR3.2). Only microphones that are actively recording noise are listed in the current implementation; this is a design that conceals inactive devices but frequently leaves quiet environments with no options. Participants pointed out that this resulted in times when there was no microphone visible, making speech input impossible. Robustness would be enhanced by a hybrid detection strategy that combines fallback listing and persistent indexing.

Speech recognition accuracy was otherwise satisfactory, though non-English city names

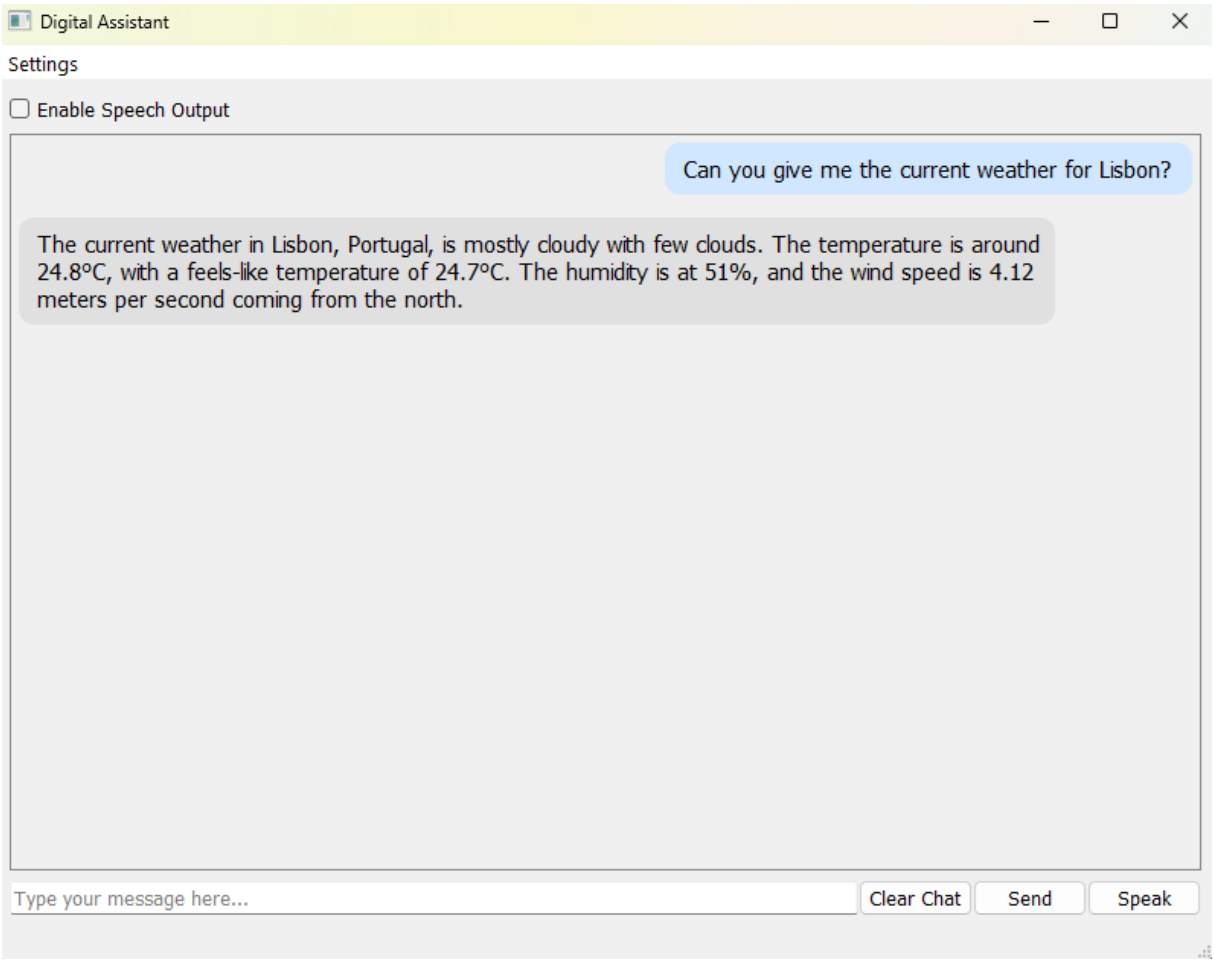

Figure 5.3 Example of the DA giving the current weather for Lisbon

occasionally caused partial misrecognition (FR2.1). Users also pointed out that there was no audio or visual cue that the system was listening (FR3.1). These results show that FR2.1/FR2.2 and FR3.1/FR3.2 are partially satisfied, indicating specific development priorities for subsequent iterations.

Launching Applications (FR5.2.1). This feature had the most success variability. Some people were able to open programs like PowerPoint or Word, but others couldn't even open simple programs like Photos or Calculator. The way Start Menu shortcuts are found explains this difference. The system currently retrieves application entries from the Start Menu folders located at `%APPDATA%\Microsoft\Windows\Start Menu\Programs` and `%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs`. These directories contain shortcuts for traditional desktop applications but exclude Microsoft Store apps, which are installed in a protected subsystem under `C:\Program Files\WindowsApps`. This directory is inaccessible even to users with administrative privileges, which the current Python-based approach didn't consider. Future improvements could integrate a complementary discovery mechanism by invoking PowerShell commands such as `Get-AppxPackage` or by interfacing with the `shell:AppsFolder` namespace through COM from Python, thereby extending coverage to Microsoft Store applications. When launches worked, users said the delay was acceptable, which suggests that the main problem is reliability, not latency. So, FR5 is met for the meteorology module but only partially met for launching applications.

Conversational Ability and Interpretation (FR4). Participants thought responses to factual or command-oriented questions were logical, pertinent, and organic (another example is shown in Figure 5.4).

Although follow-up comprehension performed well in brief interactions, it deteriorated over longer ones, which is somewhat consistent with the design decision to forego persistent memory for privacy (NFR9). For brief to medium-length conversations, FR4 is thus mainly satisfied, though there is opportunity to enhance contextual retention during sessions.

Usability (NFR1) via SUS. According to standard interpretation, the mean score of 75.38 on the SUS falls into the "Good" range. The DA was rated by participants as being well integrated, quick to learn, and easy to use, requiring little technical support. The conclusion that NFR1 is met was supported by qualitative comments that praised the assistant's responsiveness and clarity.

Perceived responsiveness (NFR2). Participants characterized the DA as quick and fluid for information tasks, which was consistent with their subjective perceptions and the measured latencies. Only when microphone detection failed or an application could not be found were delays recorded; these were implementation-related problems rather than system speed problems. These findings validate that NFR2 is satisfied by perceived responsiveness.

Error Handling and Robustness (NFR3). Some failures were gracefully handled by the DA, who provided backup messages or suggestions. Users did, however, also report instances in which the process ended suddenly (e.g., unfound applications or missing microphones). This shows partial fulfillment of NFR3, indicating that more explicit recovery prompts, retry logic, and

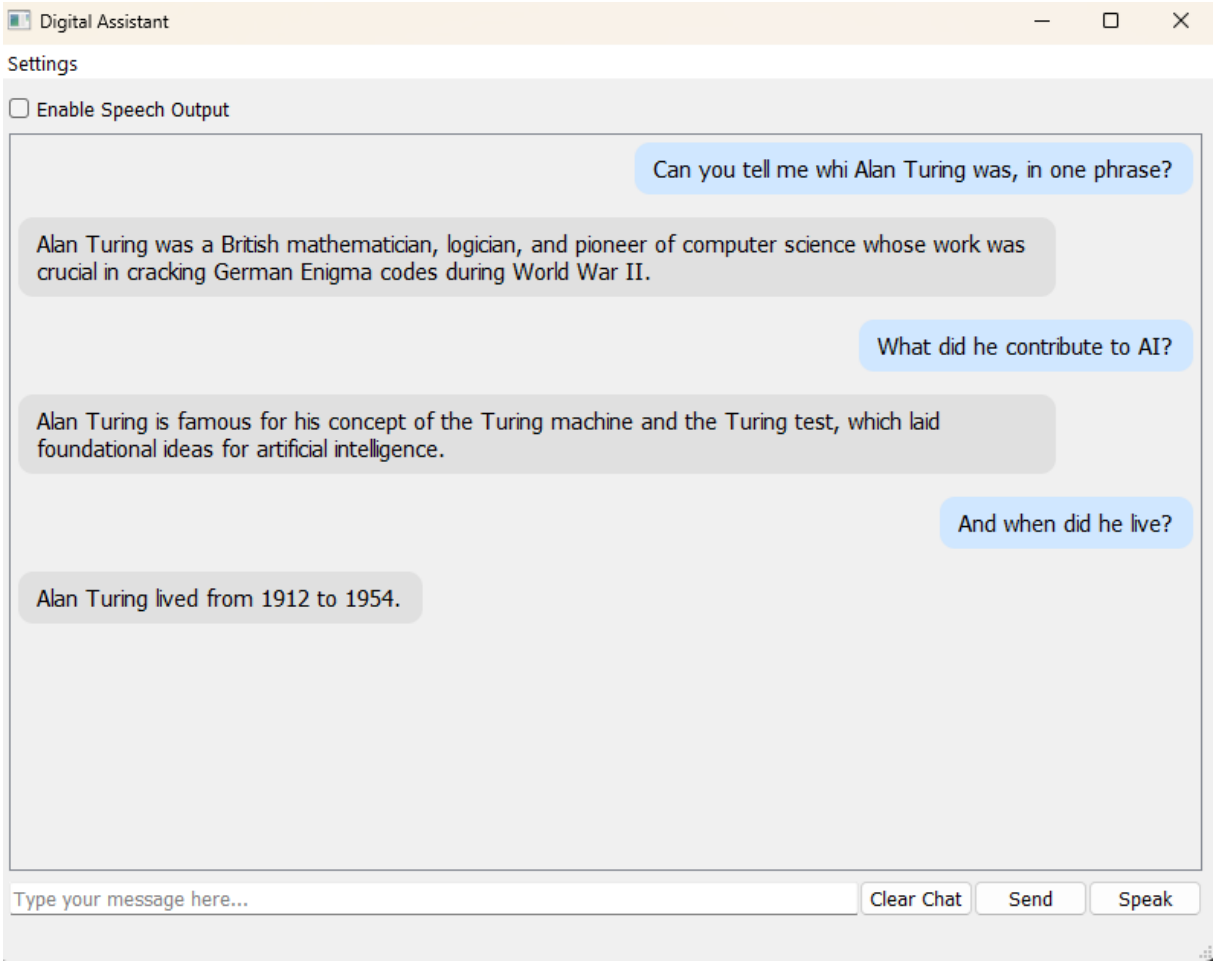

Figure 5.4 Example of the DA contextual awareness during a multi-turn conversation about Alan Turing

more lucid feedback should be included in future work.

Requirements not directly assessed by the questionnaire. Some requirements were not explicitly tested in the user study and remain primarily validated by design and experimental evidence presented elsewhere in this chapter:

- **NFR4 (Modularity) & NFR7 (Separation of Concerns):** Not directly visible to users but confirmed by successful backend interchangeability in Section 5.2.
- **NFR5 (Extensibility):** Supported by the ModuleLoader class with the configuration file plus the FunctionHandler interface, not directly perceivable in the study.
- **NFR6 (Platform/Deployment):** All evaluations were performed on Windows 10 and 11 systems by executing the project directly through the Python environment rather than as a standalone packaged application; portability and distribution to other operating systems remain future work.
- **NFR9 (Privacy/Security):** Not directly surveyed; compliance ensured through non-persistent conversation design and transparent reliance on external LLM providers.
- **FR3.3 (Input Sensitivity Configuration):** Not implemented in this version, and therefore not assessable by users.

Chapter 6

Conclusions

This work presented the design, implementation, and evaluation of a modular DA capable of natural language conversation and function execution powered by LLM. The primary objective was to create an interactive desktop system that combines text and speech input, speech synthesis output, and practical functionalities such as weather retrieval and launching applications. Unlike mobile and cloud-based ecosystems, the desktop environment still lacks versatile, user-friendly digital assistant solutions, this project aims to fill that gap by proposing a scalable and extensible desktop-oriented alternative.

The developed solution achieved these goals by implementing a layered and modular architecture separating the GUI, conversation handling, LLM integration, and functionality execution. The system supports both typed and spoken interaction, produces natural-sounding speech responses, and dynamically decides between conversational replies and function calling based on LLM interpretation. Its modular design was validated through the integration of multiple LLM backends and two distinct function-calling strategies, demonstrating that the core logic can adapted with minimal changes.

The comprehensive evaluation, including both performance testing and a structured user study, demonstrated that most functional and non-functional requirements were successfully met. Participants from diverse age groups evaluated the assistant through guided tasks and a usability questionnaire. The System Usability Scale yielded a mean score of 75.38, placing the DA in the "Good" usability range. Qualitative feedback confirmed that the interface was intuitive, responses were clear, and task success rates were high across weather and pollution related queries. Voice interaction was effective but limited by microphone detection issues and text-to-speech playback that cannot yet be interrupted mid-utterance. These findings validate requirements FR2.1 / FR2.2 / FR3.1 / FR3.2 as partially met, with clear improvement directions. Application launching worked for many desktop applications but not for Microsoft Store apps, due to their restricted installation paths, leaving FR 5 partially satisfied for that module. Conversational ability (FR4) was rated as coherent and natural for short exchanges, aligning with the design goal of non-persistent, privacy-preserving interactions (NFR9).

Performance testing confirmed that functionality modules consistently achieved sub-second latency, while modern LLM backends (Gemini and Qwen with tool calls) delivered near-real-

time responses, satisfying NFR2 (Responsiveness). Error handling (NFR3) was only partially satisfied, as user feedback revealed abrupt terminations when no microphones were detected or when applications were unavailable. The modular architecture fulfills NFR4, NFR5, and NFR7, as demonstrated by backend interchangeability and separation of concerns. Platform validation confirmed stable operation on Windows 11 (NFR6), though cross-platform portability remains to be explored.

Overall, the assistant meets its principal goals and demonstrates that a modular, LLM-driven desktop digital assistant can be practical, responsive, and extensible. The integration of user feedback through structured evaluation provides external validation that the system is usable, efficient, and functionally complete for its intended scope.

6.1 Future Work

While the project achieved its main objectives, several areas for improvement and expansion were identified during the user evaluation and technical analysis.

First, speech recognition remains dependent on clear articulation and close microphone proximity. The use of the `SpeechRecognition` library limits device enumeration reliability. Future work should explore hybrid or persistent microphone indexing, or adopt newer ASR technologies such as Whisper API or other open-source recognizers, to improve robustness and accessibility.

Second, speech synthesis should support playback interruption and optional voice customization. The current TTS backend cannot cancel or replace an ongoing utterance once playback begins, slightly reducing interactivity. Integrating engines with mid-speech cancellation or user-selectable voices would enhance naturalness and control.

Third, application launching requires expanded coverage. The present method scans Start Menu directories under `%APPDATA%\Microsoft\Windows\Start Menu\Programs` and `%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs`, which exclude Microsoft Store applications stored in `C:\Program Files\WindowsApps`. Because this directory is protected, future iterations could use PowerShell commands such as `Get-AppxPackage` or COM-based access to the `shell:AppsFolder` namespace from Python to enable comprehensive detection.

Fourth, error handling and robustness can be reinforced through systematic testing, retry logic, and user-friendly recovery prompts when microphones or applications are unavailable. This will strengthen NFR3 and further align subjective reliability with objective performance.

Finally, extending cross-platform deployment and conducting broader usability studies are important next steps. Packaging the assistant as a standalone executable (e.g., via PyInstaller) and testing on Linux and macOS would validate portability (NFR6). A larger user study could refine perceived usability, conversational continuity, and long-term adoption insights.

Taken together, these improvements would make the assistant more resilient, accessible, and user-centric, while further demonstrating the scalability of its modular architecture for future features and deployment contexts.

Annex A

Performance Evaluation Details

This annex is organized to provide more details on the evaluations present at Section 5.2 of the digital assistant. Section A.1 lists the parameters used to test each functionality module, including meteorology and operating system modules, where these parameters were used to create the results present in Section A.3, which are summarized in Table 5.1. Section A.2 presents representative utterances used to evaluate the response time and correctness of the LLMs across normal conversation and each supported functionality. These phrases are then used to obtain the results, present in Section A.4, of each test case, including execution time, correctness, and notable observations, where these results are summarized in Table 5.2. Regarding the complete logs of the results, these are available in the project repository located at:

https://github.com/pataponjak3/Digital-Assistant-Project

A.1 Functionality Module Test Parameters

A.1.1 Meteorology Module

```
'lat': 40.7128, 'lon': -74.006
'lat': -33.8688, 'lon': 151.2093
'lat': -23.5505, 'lon': -46.6333
'lat': 30.0444, 'lon': 31.2357
'lat': 55.7558, 'lon': 37.6173
'lat': -1.2921, 'lon': 36.8219
'lat': 64.1466, 'lon': -21.9426
'lat': -33.9249, 'lon': 18.4241
'lat': 35.6895, 'lon': 139.6917
'lat': 38.7223, 'lon': -9.1393
'zip': 10001, 'country_code': 'US'
'zip': 90210, 'country_code': 'US'
'zip': 60614, 'country_code': 'US'
```

```
'zip': 10115, 'country_code': 'DE'
'zip': 'SW1A 1AA', 'country_code': 'GB'
'zip': 75001, 'country_code': 'FR'
'zip': '1250-096', 'country_code': 'PT'
'zip': 2000, 'country_code': 'AU'
'zip': '01000-000', 'country_code': 'BR'
'zip': 110001, 'country_code': 'IN'
'city': 'Kyoto', 'country_code': 'JP'
'city': 'Toronto', 'country_code': 'CA'
'city': 'Nairobi'
'city': 'Auckland', 'country_code': 'NZ'
'city': 'Reykjavik', 'country_code': 'IS'
'city': 'Lima', 'country_code': 'PE'
'city': 'Johannesburg', 'country_code': 'ZA'
'city': 'Munich', 'country_code': 'DE'
'city': 'San Francisco', 'state_code': 'CA', 'country_code': 'US'
'city': 'Porto', 'country_code': 'PT'
```

A.1.2 OS Module (Launch Application)

```
'app_name': 'Steam'
'app_name': 'dis cord'
'app_name': 'Eclipse IDE'
'app_name': 'VS Code'
'app_name': 'visual studio code'
'app_name': 'VSC'
'app_name': 'Win RAR'
'app_name': 'WinRAR'
'app_name': 'Google Chorme'
'app_name': 'chrome browser'
'app_name': 'Firefox'
'app_name': 'Notepad'
'app_name': 'calc'
'app_name': 'VLC'
'app_name': 'Spotify'
'app_name': 'Adobe Reader'
'app_name': 'Acrobat'
'app_name': 'Slack'
'app_name': 'MS Teams'
'app_name': 'Outlok'
'app_name': 'Word'
```

```
'app_name':  'Excel'
'app_name':  'Power Point'
'app_name':  'Paint'
'app_name':  'Edge browser'
'app_name':  'Git Extensions'
'app_name':  '7Zip'
'app_name':  'IntelliJ'
'app_name':  'Pychram'
'app_name':  'Android Studio'
```

A.2 LLM Test Phrases

Example utterances used for each functionality category:

Normal Conversation:

"Explain the difference between threads and processes in one paragraph."
"Give me 3 meal prep ideas for a vegetarian who trains for half-marathons."
"Summarize the key idea of 'inversion' from mental models (keep it short)."
"What are common pitfalls when using Python's asyncio with CPU-bound work?"
"Draft a friendly email asking for a project deadline extension."
"What's a good daily warm-up routine for desk workers? Keep it under 8 steps."
"I keep forgetting people's names—share 4 memory techniques."
"Turn this into a bullet list: 'Plan, execute, measure, iterate.' Add one emoji each."
"Write a tiny story (<=120 words) about a lighthouse learning Morse code."
"Explain CAP theorem like I'm new to distributed systems."
"Compare SQLite vs PostgreSQL for a solo desktop app."
"How would you unit test a function that parses CSV lines? Keep it high level."
"What's the minimum I need to know about UX heuristics to not mess up my UI?"
"I'm anxious before presentations—give me 5 quick tips."
"Rewrite 'optimize the pipeline' in plainer language."
"Brainstorm 6 team-building activities for remote developers."
"What's the trade-off between early abstraction and duplication?"
"Explain why floating-point math can be surprising to newcomers."
"Give me a one-liner pep talk for debugging at 2 a.m."
"List 5 git hygiene practices for small teams."
"What's a simple analogy for gradient descent?"
"Suggest 4 interview questions to test problem decomposition skills."
"How do I politely push back on scope creep?"
"Convert this to title case: 'an introduction to concurrency primitives'"
"Name 5 signs a backlog item isn't ready for dev."
"What's a good rubric to decide whether to refactor now or later?"

"Give me a tiny regex cheat sheet (anchors, groups, classes)."

"What's the 80/20 of Docker I should know to ship a Python app?"

"Explain the difference between correlation and causation with one example."

"Turn 'I will try' into 3 stronger alternatives."

Get current weather:

"Current weather at 40.7128, -74.0060?"

"What's it like now near -33.8688, 151.2093 (Sydny)?"

"Weather rn at -23.5505, -46.6333."

"Temp & conditions at 30.0444, 31.2357 please."

"What's the current weather @ 55.7558, 37.6173?"

"Is it raining around -1.2921, 36.8219 right now?"

"How cold is it near 64.1466, -21.9426?"

"Current wind & temp for -33.9249, 18.4241."

"Weather now at 35.6895, 139.6917 (Tokio)?"

"What's the weather like at 38.7223, -9.1393?"

"Current weather for zip 10001, US."

"Weather now at 90210, US?"

"How's Chicago right now—60614, US."

"What's it like at 10115, DE (Berlin)?"

"Weather for SW1A 1AA, GB (Buckingham)?"

"Now in 75001, FR?"

"Current weather for 1250-096, PT (Lisboa)?"

"What's it now at 2000, AU (Sydney CBD)?"

"Weather rn at 01000-000, BR (São Paulo)."

"Current conditions for 110001, IN (New Delhi)."

"What's the weather in Kyoto, JP right now?"

"Current conditions in Toronto, CA?"

"How's Nairobi today?"

"Weather now in Auckland, NZ."

"What's it like in Reykyavik, IS?"

"Current weather for Lima, PE."

"How's Jo'burg (Johannesburg, ZA) right now?"

"Weather in Munich, DE (now)."

"Current conditions San Francisco, US-CA."

"What's the weather in Porto, PT?"

Get forecast

"5-day forecast for 40.7128, -74.0060."

"Next 5 days near -33.8688, 151.2093?"

"Forecast (5d) at -23.5505, -46.6333."

"Show me 5-day outlook for 30.0444, 31.2357."

"What’s the week ahead at 55.7558, 37.6173?"

"Five-day forecast around -1.2921, 36.8219."

"Is it cooling later this week near 64.1466, -21.9426?"

"Give 5-day hi/lo for -33.9249, 18.4241."

"Forecast next days at 35.6895, 139.6917."

"Five-day for 38.7223, -9.1393 (Lisbon)."

"5-day forecast 10001, US."

"Forecast for 90210, US next 5 days."

"Weather outlook 60614, US."

"5-day for 10115, DE."

"Next five days SW1A 1AA, GB."

"Forecast 75001, FR."

"5-day for 1250-096, PT."

"Forecast for 2000, AU."

"5-day outlook 01000-000, BR."

"Next 5 days 110001, IN."

"Five-day forecast Kyoto, JP."

"What’s the 5-day in Toronto, CA?"

"5-day forecast Nairobi."

"Week ahead in Auckland, NZ."

"5-day for Reykjavik, IS (typo earlier)."

"Forecast 5 days Lima, PE."

"Next 5 days Johannesburg, ZA."

"Munich, DE 5-day outlook."

"San Francisco, US-CA 5-day forecast."

"5-day for Porto, PT."

Get air pollution

"Current air quality at 40.7128, -74.0060?"

"AQI near -33.8688, 151.2093 right now."

"What’s PM2.5 at -23.5505, -46.6333?"

"Air pollution for 30.0444, 31.2357."

"AQI around 55.7558, 37.6173 (Moscow)."

"Current AQI for -1.2921, 36.8219."

"Is air clean near 64.1466, -21.9426?"

"Pollution stats at -33.9249, 18.4241."

"AQI now 35.6895, 139.6917."

"Air quality 38.7223, -9.1393."

"Air quality for 10001, US."

"AQI now at 90210, US."

"Pollution level 60614, US."

"AQI 10115, DE."
"Air quality SW1A 1AA, GB."
"AQI 75001, FR."
"Air quality 1250-096, PT."
"AQI 2000, AU."
"Air quality 01000-000, BR."
"AQI 110001, IN."
"Current AQI Kyoto, JP."
"Air quality Toronto, CA now."
"AQI Nairobi?"
"Air quality in Auckland, NZ."
"AQI Reykjavik, IS (sorry for prev typo)."
"Pollution level Lima, PE."
"AQI Johannesburg, ZA."
"Air quality Munich, DE."
"AQI San Francisco, US-CA."
"Air quality Porto, PT."

Launch Application

"Open Steam."
"Launch dis cord."
"Start Eclipse IDE."
"Open VS Code."
"Run visual studio code pls."
"Launch VSC."
"Open Win RAR."
"Start WinRAR."
"Open Google Chorme."
"Launch chrome browser."
"Open Firefox."
"Start Notepad."
"Launch calc."
"Open VLC."
"Start Spotify."
"Open Adobe Reader."
"Launch Acrobat."
"Open Slack."
"Start MS Teams."
"Open Outlok."
"Launch Word."
"Open Excel."

"Start Power Point."

"Open Paint."

"Start Edge browser."

"Launch Git Extensions."

"Open 7Zip."

"Start IntelliJ."

"Open Pychram."

"Launch Android Studio."

A.3 Results for the Execution Time of Functionality Modules

A.3.1 Meteorology Functionality

Table A.1 Execution Time of Get current weather (in seconds)

Parameters	Time	
	Without Formatting	With Formatting
{'lat': 40.7128, 'lon': -74.006}	0,32	0,28
{'lat': -33.8688, 'lon': 151.2093}	0,29	0,28
{'lat': -23.5505, 'lon': -46.6333}	0,27	0,28
{'lat': 30.0444, 'lon': 31.2357}	0,29	0,30
{'lat': 55.7558, 'lon': 37.6173}	0,29	0,28
{'lat': -1.2921, 'lon': 36.8219}	0,29	0,29
{'lat': 64.1466, 'lon': -21.9426}	0,28	0,29
{'lat': -33.9249, 'lon': 18.4241}	0,29	0,28
{'lat': 35.6895, 'lon': 139.6917}	0,29	0,29
{'lat': 38.7223, 'lon': -9.1393}	0,30	0,29
{'zip': 10001, 'country_code': 'US'}	0,30	0,28
{'zip': 90210, 'country_code': 'US'}	0,29	0,28
{'zip': 60614, 'country_code': 'US'}	0,29	0,29
{'zip': 10115, 'country_code': 'DE'}	0,32	0,29
{'zip': 'SW1A 1AA', 'country_code': 'GB'}	0,29	0,29
{'zip': 75001, 'country_code': 'FR'}	0,28	0,29
{'zip': '1250-096', 'country_code': 'PT'}	0,28	0,30
{'zip': 2000, 'country_code': 'AU'}	0,29	0,28
{'zip': '01000-000', 'country_code': 'BR'}	0,29	0,29
{'zip': 110001, 'country_code': 'IN'}	0,29	0,30
{'city': 'Kyoto', 'country_code': 'JP'}	0,28	0,28
{'city': 'Toronto', 'country_code': 'CA'}	0,29	0,28
{'city': 'Nairobi'}	0,29	0,29
{'city': 'Auckland', 'country_code': 'NZ'}	0,29	0,28
{'city': 'Reykjavik', 'country_code': 'IS'}	0,30	0,29
{'city': 'Lima', 'country_code': 'PE'}	0,29	0,29
{'city': 'Johannesburg', 'country_code': 'ZA'}	0,28	0,29
{'city': 'Munich', 'country_code': 'DE'}	0,29	0,28
{'city': 'San Francisco', 'state_code': 'CA', 'country_code': 'US'}	0,30	0,28
{'city': 'Porto', 'country_code': 'PT'}	0,28	0,29

Table A.2 Execution Time of 5-day forecast (in seconds)

Parameters	Time	
	Without Formatting	With Formatting
{'lat': 40.7128, 'lon': -74.006}	0,32	0,32
{'lat': -33.8688, 'lon': 151.2093}	0,32	0,31
{'lat': -23.5505, 'lon': -46.6333}	0,33	0,33
{'lat': 30.0444, 'lon': 31.2357}	0,33	0,33
{'lat': 55.7558, 'lon': 37.6173}	0,33	0,33
{'lat': -1.2921, 'lon': 36.8219}	0,34	0,50
{'lat': 64.1466, 'lon': -21.9426}	0,32	0,35
{'lat': -33.9249, 'lon': 18.4241}	0,35	0,33
{'lat': 35.6895, 'lon': 139.6917}	0,32	0,32
{'lat': 38.7223, 'lon': -9.1393}	0,34	0,55
{'zip': 10001, 'country_code': 'US'}	0,34	0,33
{'zip': 90210, 'country_code': 'US'}	0,33	0,32
{'zip': 60614, 'country_code': 'US'}	0,34	0,33
{'zip': 10115, 'country_code': 'DE'}	0,34	0,34
{'zip': 'SW1A 1AA', 'country_code': 'GB'}	0,33	0,34
{'zip': 75001, 'country_code': 'FR'}	0,33	0,36
{'zip': '1250-096', 'country_code': 'PT'}	0,33	0,33
{'zip': 2000, 'country_code': 'AU'}	0,35	0,33
{'zip': '01000-000', 'country_code': 'BR'}	0,33	0,34
{'zip': 110001, 'country_code': 'IN'}	0,33	0,35
{'city': 'Kyoto', 'country_code': 'JP'}	0,33	0,32
{'city': 'Toronto', 'country_code': 'CA'}	0,33	0,33
{'city': 'Nairobi'}	0,33	0,33
{'city': 'Auckland', 'country_code': 'NZ'}	0,33	0,33
{'city': 'Reykjavik', 'country_code': 'IS'}	0,33	0,33
{'city': 'Lima', 'country_code': 'PE'}	0,34	0,33
{'city': 'Johannesburg', 'country_code': 'ZA'}	0,33	0,33
{'city': 'Munich', 'country_code': 'DE'}	0,35	0,34
{'city': 'San Francisco', 'state_code': 'CA', 'country_code': 'US'}	0,32	0,33
{'city': 'Porto', 'country_code': 'PT'}	0,33	0,33

Table A.3 Execution Time of Air pollution (in seconds)

Parameters	Time	
	Without Formatting	With Formatting
{'lat': 40.7128, 'lon': -74.006}	0,27	0,28
{'lat': -33.8688, 'lon': 151.2093}	0,29	0,28
{'lat': -23.5505, 'lon': -46.6333}	0,28	0,28
{'lat': 30.0444, 'lon': 31.2357}	0,28	0,30
{'lat': 55.7558, 'lon': 37.6173}	0,29	0,29
{'lat': -1.2921, 'lon': 36.8219}	0,28	0,28
{'lat': 64.1466, 'lon': -21.9426}	0,28	0,30
{'lat': -33.9249, 'lon': 18.4241}	0,29	0,28
{'lat': 35.6895, 'lon': 139.6917}	0,29	0,29
{'lat': 38.7223, 'lon': -9.1393}	0,28	0,31
{'zip': 10001, 'country_code': 'US'}	0,29	0,28
{'zip': 90210, 'country_code': 'US'}	0,31	0,28
{'zip': 60614, 'country_code': 'US'}	0,29	0,28
{'zip': 10115, 'country_code': 'DE'}	0,31	0,29
{'zip': 'SW1A 1AA', 'country_code': 'GB'}	0,29	0,29
{'zip': 75001, 'country_code': 'FR'}	0,29	0,29
{'zip': '1250-096', 'country_code': 'PT'}	0,32	0,28
{'zip': 2000, 'country_code': 'AU'}	0,31	0,30
{'zip': '01000-000', 'country_code': 'BR'}	0,30	0,29
{'zip': 110001, 'country_code': 'IN'}	0,30	0,29
{'city': 'Kyoto', 'country_code': 'JP'}	0,29	0,28
{'city': 'Toronto', 'country_code': 'CA'}	0,29	0,28
{'city': 'Nairobi'}	0,30	0,29
{'city': 'Auckland', 'country_code': 'NZ'}	0,29	0,28
{'city': 'Reykjavik', 'country_code': 'IS'}	0,29	0,29
{'city': 'Lima', 'country_code': 'PE'}	0,28	0,28
{'city': 'Johannesburg', 'country_code': 'ZA'}	0,29	0,31
{'city': 'Munich', 'country_code': 'DE'}	0,30	0,29
{'city': 'San Francisco', 'state_code': 'CA', 'country_code': 'US'}	0,28	0,31
{'city': 'Porto', 'country_code': 'PT'}	0,30	0,28

Table A.4 Execution Time of `launch_application` (in seconds)

Parameters	Time
{'app_name': 'Steam'}	1,87
{'app_name': 'dis cord'}	1,18
{'app_name': 'Eclipse IDE'}	1,26
{'app_name': 'VS Code'}	1,20
{'app_name': 'visual studio code'}	1,45
{'app_name': 'VSC'}	1,48
{'app_name': 'Win RAR'}	1,54
{'app_name': 'WinRAR'}	1,46
{'app_name': 'Google Chorme'}	1,85
{'app_name': 'chrome browser'}	1,48
{'app_name': 'Firefox'}	1,22
{'app_name': 'Notepad'}	1,14
{'app_name': 'calc'}	1,21
{'app_name': 'VLC'}	1,26
{'app_name': 'Spotify'}	1,05
{'app_name': 'Adobe Reader'}	1,04
{'app_name': 'Acrobat'}	1,20
{'app_name': 'Slack'}	1,04
{'app_name': 'MS Teams'}	1,02
{'app_name': 'Outlok'}	1,05
{'app_name': 'Word'}	1,24
{'app_name': 'Excel'}	1,32
{'app_name': 'Power Point'}	1,25
{'app_name': 'Paint'}	1,37
{'app_name': 'Edge browser'}	1,55
{'app_name': 'Git Extensions'}	1,25
{'app_name': '7Zip'}	1,20
{'app_name': 'IntelliJ'}	1,22
{'app_name': 'Pychram'}	1,34
{'app_name': 'Android Studio'}	1,70

A.4 Results for LLM Response Time and Correctness

A.4.1 Awan LLM's Llama 3.1 8B Instruct

Table A.5 Normal Conversation

Utterance	Time(s)	Correctness	Notes
"Explain the difference between threads and processes in one paragraph."	20,51	Yes	N/A
"Give me 3 meal prep ideas for a vegetarian who trains for half-marathons."	19,97	Yes	N/A
"Summarize the key idea of 'inversion' from mental models (keep it short)."	6,05	Yes	N/A
"What are common pitfalls when using Python's asyncio with CPU-bound work?"	19,06	Yes	N/A
"Draft a friendly email asking for a project deadline extension."	14,44	Yes	N/A
"What's a good daily warm-up routine for desk workers? Keep it under 8 steps."	18,97	Yes	N/A
"I keep forgetting people's names—share 4 memory techniques."	17,22	Yes	N/A
"Turn this into a bullet list: 'Plan, execute, measure, iterate.' Add one emoji each."	2,98	Yes	N/A
"Write a tiny story (<=120 words) about a lighthouse learning Morse code."	8,40	Yes	N/A
"Explain CAP theorem like I'm new to distributed systems."	23,67	Yes	N/A
"Compare SQLite vs PostgreSQL for a solo desktop app."	26,01	Yes	N/A
"How would you unit test a function that parses CSV lines? Keep it high level."	24,05	Yes	N/A
"What's the minimum I need to know about UX heuristics to not mess up my UI?"	18,17	Yes	N/A
"I'm anxious before presentations—give me 5 quick tips."	16,00	Yes	N/A
"Rewrite 'optimize the pipeline' in plainer language."	2,74	Yes	N/A
"Brainstorm 6 team-building activities for remote developers."	20,50	Yes	N/A
"What's the trade-off between early abstraction and duplication?"	20,69	Yes	N/A
"Explain why floating-point math can be surprising to newcomers."	19,07	Yes	N/A
"Give me a one-liner pep talk for debugging at 2 a.m."	2,57	Yes	N/A
"List 5 git hygiene practices for small teams."	18,24	Yes	N/A
"What's a simple analogy for gradient descent?"	9,83	Yes	N/A
"Suggest 4 interview questions to test problem decomposition skills."	34,11	Yes	N/A
"How do I politely push back on scope creep?"	19,88	Yes	N/A
"Convert this to title case: 'an introduction to concurrency primitives'"	1,45	Yes	N/A
"Name 5 signs a backlog item isn't ready for dev."	16,83	Yes	N/A
"What's a good rubric to decide whether to refactor now or later?"	25,81	Yes	N/A
"Give me a tiny regex cheat sheet (anchors, groups, classes)."	10,86	Yes	N/A
"What's the 80/20 of Docker I should know to ship a Python app?"	16,39	Yes	N/A
"Explain the difference between correlation and causation with one example."	15,06	Yes	N/A
"Turn 'I will try' into 3 stronger alternatives."	3,29	Yes	N/A

Table A.6 Get Current Weather

Utterance	Time(s)	Correctness	Notes
"Current weather at 40.7128, -74.0060?"	14,26	Yes	N/A
"What's it like now near -33.8688, 151.2093 (Sydny)?"	3,43	Yes	N/A
"Weather rn at -23.5505, -46.6333."	3,40	Yes	N/A
"Temp & conditions at 30.0444, 31.2357 please."	3,41	Yes	N/A
"What's the current weather @ 55.7558, 37.6173?"	3,43	Yes	N/A
"Is it raining around -1.2921, 36.8219 right now?"	3,42	Yes	N/A
"How cold is it near 64.1466, -21.9426?"	3,41	Yes	N/A
"Current wind & temp for -33.9249, 18.4241."	3,81	Yes	N/A
"Weather now at 35.6895, 139.6917 (Tokio)?"	3,44	Yes	N/A
"What's the weather like at 38.7223, -9.1393?"	4,42	Yes	N/A
"Current weather for zip 10001, US."	3,24	Yes	N/A
"Weather now at 90210, US?"	3,18	Yes	N/A
"How's Chicago right now—60614, US."	3,18	Yes	N/A
"What's it like at 10115, DE (Berlin)?"	3,58	Yes	N/A
"Weather for SW1A 1AA, GB (Buckingham)?"	3,41	Yes	N/A
"Now in 75001, FR?"	3,20	Yes	N/A
"Current weather for 1250-096, PT (Lisboa)?"	3,31	Yes	N/A
"What's it now at 2000, AU (Sydney CBD)?"	3,41	Yes	Converted location given by postal code to latitude/longitude.
"Weather rn at 01000-000, BR (São Paulo)."	3,42	Yes	Converted location given by postal code to latitude/longitude.
"Current conditions for 110001, IN (New Delhi)."	3,19	Yes	N/A
"What's the weather in Kyoto, JP right now?"	3,21	Yes	N/A
"Current conditions in Toronto, CA?"	3,47	No	Put parameter `state_code` without value.
"How's Nairobi today?"	2,74	Yes	N/A
"Weather now in Auckland, NZ."	3,18	Yes	N/A
"What's it like in Reykyavik, IS?"	3,29	Yes	N/A
"Current weather for Lima, PE."	3,21	Yes	N/A
"How's Jo'burg (Johannesburg, ZA) right now?"	3,29	Yes	N/A
"Weather in Munich, DE (now)."	3,27	Yes	N/A
"Current conditions San Francisco, US-CA."	3,29	No	Put country and state code in parameter `state_code`.
"What's the weather in Porto, PT?"	3,36	Yes	N/A

Table A.7 Get Forecast

Utterance	Time(s)	Correctness	Notes
"5-day forecast for 40.7128, -74.0060."	14,31	Yes	N/A
"Next 5 days near -33.8688, 151.2093?"	3,43	Yes	N/A
"Forecast (5d) at -23.5505, -46.6333."	3,45	Yes	N/A
"Show me 5-day outlook for 30.0444, 31.2357."	3,39	Yes	N/A
"What's the week ahead at 55.7558, 37.6173?"	3,43	Yes	N/A
"Five-day forecast around -1.2921, 36.8219."	3,40	Yes	N/A
"Is it cooling later this week near 64.1466, -21.9426?"	3,39	Yes	N/A
"Give 5-day hi/lo for -33.9249, 18.4241."	3,44	Yes	N/A
"Forecast next days at 35.6895, 139.6917."	5,30	Yes	N/A
"Five-day for 38.7223, -9.1393 (Lisbon)."	4,25	Yes	N/A
"5-day forecast 10001, US."	3,94	Yes	N/A
"Forecast for 90210, US next 5 days."	3,95	Yes	N/A
"Weather outlook 60614, US."	3,29	Yes	N/A
"5-day for 10115, DE."	3,30	Yes	N/A
"Next five days SW1A 1AA, GB."	3,51	Yes	N/A
"Forecast 75001, FR."	3,28	Yes	N/A
"5-day for 1250-096, PT."	3,41	Yes	N/A
"Forecast for 2000, AU."	3,29	Yes	N/A
"5-day outlook 01000-000, BR."	3,40	Yes	N/A
"Next 5 days 110001, IN."	3,31	Yes	N/A
"Five-day forecast Kyoto, JP."	3,30	Yes	N/A
"What's the 5-day in Toronto, CA?"	3,59	No	Put parameter `state_code` without value.
"5-day forecast Nairobi."	2,82	Yes	N/A
"Week ahead in Auckland, NZ."	3,29	Yes	N/A
"5-day for Reykjavik, IS (typo earlier)."	3,38	Yes	N/A
"Forecast 5 days Lima, PE."	3,29	Yes	N/A
"Next 5 days Johannesburg, ZA."	3,41	Yes	N/A
"Munich, DE 5-day outlook."	4,10	Yes	N/A
"San Francisco, US-CA 5-day forecast."	4,23	No	Put country and state code in parameter `state_code`.
"5-day for Porto, PT."	4,06	Yes	N/A

Table A.8 Get Air Pollution

Utterance	Time(s)	Correctness	Notes
"Current air quality at 40.7128, -74.0060?"	14,29	Yes	N/A
"AQI near -33.8688, 151.2093 right now."	3,48	Yes	N/A
"What's PM2.5 at -23.5505, -46.6333?"	3,49	Yes	N/A
"Air pollution for 30.0444, 31.2357."	3,49	Yes	N/A
"AQI around 55.7558, 37.6173 (Moscow)."	3,50	Yes	N/A
"Current AQI for -1.2921, 36.8219."	3,49	Yes	N/A
"Is air clean near 64.1466, -21.9426?"	3,51	Yes	N/A
"Pollution stats at -33.9249, 18.4241."	3,49	Yes	N/A
"AQI now 35.6895, 139.6917."	3,49	Yes	N/A
"Air quality 38.7223, -9.1393."	3,48	Yes	N/A
"Air quality for 10001, US."	3,43	Yes	N/A
"AQI now at 90210, US."	3,28	Yes	N/A
"Pollution level 60614, US."	3,27	Yes	N/A
"AQI 10115, DE."	3,30	Yes	N/A
"Air quality SW1A 1AA, GB."	3,47	Yes	N/A
"AQI 75001, FR."	3,27	Yes	N/A
"Air quality 1250-096, PT."	3,39	Yes	N/A
"AQI 2000, AU."	4,58	No	Put parameters `city, state_code, zip` without value, while `lat, lon` as `None`.
"Air quality 01000-000, BR."	3,38	Yes	N/A
"AQI 110001, IN."	4,57	No	Put parameters `city, state_code, country_code, lat, lon` as `None`.
"Current AQI Kyoto, JP."	3,26	Yes	N/A
"Air quality Toronto, CA now."	3,22	Yes	N/A
"AQI Nairobi?"	3,34	Yes	N/A
"Air quality in Auckland, NZ."	3,25	Yes	N/A
"AQI Reykjavik, IS (sorry for prev typo)."	3,40	Yes	N/A
"Pollution level Lima, PE."	3,25	Yes	N/A
"AQI Johannesburg, ZA."	3,42	Yes	N/A
"Air quality Munich, DE."	3,31	Yes	N/A
"AQI San Francisco, US-CA."	3,40	No	Put country and state code in parameter `state_code`.
"Air quality Porto, PT."	3,30	Yes	N/A

Table A.9 Launch Application

Utterance	Time(s)	Correctness	Notes
"Open Steam."	13,57	Yes	N/A
"Launch dis cord."	21,05	No	Failed to respond, likely due to a problem in the provider's end.
"Start Eclipse IDE."	3,44	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Open VS Code."	2,70	Yes	N/A
"Run visual studio code pls."	2,75	Yes	N/A
"Launch VSC."	2,71	Yes	N/A
"Open Win RAR."	2,77	Yes	N/A
"Start WinRAR."	2,65	Yes	N/A
"Open Google Chorme."	3,32	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Launch chrome browser."	2,63	Yes	N/A
"Open Firefox."	3,28	Yes	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start Notepad."	2,70	Yes	N/A
"Launch calc."	3,29	Yes	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open VLC."	2,73	Yes	N/A
"Start Spotify."	2,83	Yes	N/A
"Open Adobe Reader."	2,70	Yes	N/A
"Launch Acrobat."	2,72	Yes	N/A
"Open Slack."	2,72	Yes	N/A
"Start MS Teams."	3,30	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Open Outlok."	2,76	Yes	N/A
"Launch Word."	2,73	Yes	N/A
"Open Excel."	3,33	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Start Power Point."	2,59	Yes	N/A
"Open Paint."	2,69	Yes	N/A
"Start Edge browser."	3,28	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Launch Git Extensions."	7,34	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Open 7Zip."	4,27	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Start IntelliJ."	2,78	Yes	N/A
"Open Pychram."	3,33	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Launch Android Studio."	2,69	Yes	N/A

A.4.2 HuggingFace's Llama 3.1 8B Instruct

Table A.10 Normal Conversation

Utterance	Time(s)	Correctness	Notes
"Explain the difference between threads and processes in one paragraph."	1.60	Yes	N/A
"Give me 3 meal prep ideas for a vegetarian who trains for half-marathons."	2.58	Yes	N/A
"Summarize the key idea of 'inversion' from mental models (keep it short)."	0.80	Yes	N/A
"What are common pitfalls when using Python's asyncio with CPU-bound work?"	0.60	No	Interpreted conversation as function call.
"Draft a friendly email asking for a project deadline extension."	1.34	Yes	N/A
"What's a good daily warm-up routine for desk workers? Keep it under 8 steps."	2.13	Yes	N/A
"I keep forgetting people's names—share 4 memory techniques."	1.93	Yes	N/A
"Turn this into a bullet list: 'Plan, execute, measure, iterate.' Add one emoji each."	0.54	Yes	N/A
"Write a tiny story (<=120 words) about a lighthouse learning Morse code."	1.29	Yes	N/A
"Explain CAP theorem like I'm new to distributed systems."	2.42	Yes	N/A
"Compare SQLite vs PostgreSQL for a solo desktop app."	2.67	Yes	N/A
"How would you unit test a function that parses CSV lines? Keep it high level."	2.00	Yes	N/A
"What's the minimum I need to know about UX heuristics to not mess up my UI?"	2.35	Yes	N/A
"I'm anxious before presentations—give me 5 quick tips."	2.13	Yes	N/A
"Rewrite 'optimize the pipeline' in plainer language."	1.18	Yes	N/A
"Brainstorm 6 team-building activities for remote developers."	1.98	Yes	N/A
"What's the trade-off between early abstraction and duplication?"	2.46	Yes	N/A
"Explain why floating-point math can be surprising to newcomers."	3.60	Yes	N/A
"Give me a one-liner pep talk for debugging at 2 a.m."	0.74	Yes	N/A
"List 5 git hygiene practices for small teams."	2.05	Yes	N/A
"What's a simple analogy for gradient descent?"	1.10	Yes	N/A
"Suggest 4 interview questions to test problem decomposition skills."	2.27	Yes	N/A
"How do I politely push back on scope creep?"	2.80	Yes	N/A
"Convert this to title case: 'an introduction to concurrency primitives'"	0.40	Yes	N/A
"Name 5 signs a backlog item isn't ready for dev."	2.24	Yes	N/A
"What's a good rubric to decide whether to refactor now or later?"	0.53	Yes	N/A
"Give me a tiny regex cheat sheet (anchors, groups, classes)."	3.30	Yes	N/A
"What's the 80/20 of Docker I should know to ship a Python app?"	2.71	Yes	N/A
"Explain the difference between correlation and causation with one example."	1.79	Yes	N/A
"Turn 'I will try' into 3 stronger alternatives."	1.09	Yes	N/A

Table A.11 Get Current Weather

Utterance	Time(s)	Correctness	Notes
"Current weather at 40.7128, -74.0060?"	1.20	Yes	N/A
"What's it like now near -33.8688, 151.2093 (Sydny)?"	0.77	Yes	N/A
"Weather rn at -23.5505, -46.6333."	0.68	Yes	N/A
"Temp & conditions at 30.0444, 31.2357 please."	0.74	Yes	N/A
"What's the current weather @ 55.7558, 37.6173?"	0.76	Yes	N/A
"Is it raining around -1.2921, 36.8219 right now?"	0.76	Yes	N/A
"How cold is it near 64.1466, -21.9426?"	0.69	Yes	N/A
"Current wind & temp for -33.9249, 18.4241."	0.79	Yes	N/A
"Weather now at 35.6895, 139.6917 (Tokio)?"	0.73	Yes	N/A
"What's the weather like at 38.7223, -9.1393?"	0.71	Yes	N/A
"Current weather for zip 10001, US."	0.66	Yes	N/A
"Weather now at 90210, US?"	0.76	Yes	N/A
"How's Chicago right now—60614, US."	1.04	Yes	Inferred parameters `units, lang` with values `standard, en`.
"What's it like at 10115, DE (Berlin)?"	0.85	Yes	N/A
"Weather for SW1A 1AA, GB (Buckingham)?"	0.84	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Now in 75001, FR?"	0.73	Yes	N/A
"Current weather for 1250-096, PT (Lisboa)?"	1.00	Yes	Put city name obtained from text.
"What's it now at 2000, AU (Sydney CBD)?"	0.86	Yes	Converted Location given by postal code to latitude/longitude, while maintaining country code and adding city name.
"Weather rn at 01000-000, BR (São Paulo)."	1.05	Yes	Put city name obtained from text and inferred parameters `units, lang` with values `metric, pt`.
"Current conditions for 110001, IN (New Delhi)."	0.79	Yes	Put city name obtained from text.
"What's the weather in Kyoto, JP right now?"	0.62	Yes	N/A
"Current conditions in Toronto, CA?"	0.66	Yes	N/A
"How's Nairobi today?"	0.65	Yes	Inferred parameter `country_code` correctly.
"Weather now in Auckland, NZ."	0.78	Yes	N/A
"What's it like in Reykyavik, IS?"	0.71	Yes	N/A
"Current weather for Lima, PE."	0.70	Yes	N/A
"How's Jo'burg (Johannesburg, ZA) right now?"	0.66	Yes	N/A
"Weather in Munich, DE (now)."	0.82	Yes	N/A
"Current conditions San Francisco, US-CA."	0.88	No	Put country and state code in parameter `state_code`, also put the parameter `country_code` correctly.
"What's the weather in Porto, PT?"	0.79	Yes	N/A

Table A.12 Get Forecast

Utterance	Time(s)	Correctness	Notes
"5-day forecast for 40.7128, -74.0060."	1.22	Yes	N/A
"Next 5 days near -33.8688, 151.2093?"	0.77	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Forecast (5d) at -23.5505, -46.6333."	0.67	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Show me 5-day outlook for 30.0444, 31.2357."	0.61	Yes	N/A
"What's the week ahead at 55.7558, 37.6173?"	0.79	Yes	N/A
"Five-day forecast around -1.2921, 36.8219."	0.64	Yes	N/A
"Is it cooling later this week near 64.1466, -21.9426?"	0.61	Yes	N/A
"Give 5-day hi/lo for -33.9249, 18.4241."	0.51	Yes	N/A
"Forecast next days at 35.6895, 139.6917."	0.71	Yes	N/A
"Five-day for 38.7223, -9.1393 (Lisbon)."	1.06	Yes	Inferred parameter `units` with value `standard`.
"5-day forecast 10001, US."	0.71	Yes	N/A
"Forecast for 90210, US next 5 days."	0.89	Yes	N/A
"Weather outlook 60614, US."	0.60	Yes	N/A
"5-day for 10115, DE."	0.55	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Next five days SW1A 1AA, GB."	0.62	Yes	N/A
"Forecast 75001, FR."	0.90	Yes	N/A
"5-day for 1250-096, PT."	0.71	Yes	Inferred parameters `units, lang` with values `standard, pt`.
"Forecast for 2000, AU."	0.52	Yes	N/A
"5-day outlook 01000-000, BR."	0.81	Yes	Inferred parameter `city` correctly.
"Next 5 days 110001, IN."	0.66	No	Put correct country code in parameter `state_code` and inferred parameter `country_code` incorrectly.
"Five-day forecast Kyoto, JP."	0.61	Yes	N/A
"What's the 5-day in Toronto, CA?"	0.76	Yes	N/A
"5-day forecast Nairobi."	0.80	Yes	N/A
"Week ahead in Auckland, NZ."	0.61	Yes	N/A
"5-day for Reykjavik, IS (typo earlier)."	0.60	Yes	Inferred parameter `units` with value `standard`.
"Forecast 5 days Lima, PE."	0.85	Yes	Inferred parameter `units` with value `standard`.
"Next 5 days Johannesburg, ZA."	0.70	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Munich, DE 5-day outlook."	0.84	Yes	N/A
"San Francisco, US-CA 5-day forecast."	0.65	No	Put country and state code in parameter `state_code` and inferred parameter `units` with value `standard`.
"5-day for Porto, PT."	0.54	Yes	N/A

Table A.13 Get Air Pollution

Utterance	Time(s)	Correctness	Notes
"Current air quality at 40.7128, -74.0060?"	1.17	Yes	N/A
"AQI near -33.8688, 151.2093 right now."	0.60	Yes	N/A
"What's PM2.5 at -23.5505, -46.6333?"	0.72	Yes	N/A
"Air pollution for 30.0444, 31.2357."	0.59	Yes	N/A
"AQI around 55.7558, 37.6173 (Moscow)."	0.71	Yes	N/A
"Current AQI for -1.2921, 36.8219."	0.69	Yes	N/A
"Is air clean near 64.1466, -21.9426?"	0.60	Yes	N/A
"Pollution stats at -33.9249, 18.4241."	0.53	Yes	N/A
"AQI now 35.6895, 139.6917."	0.75	Yes	N/A
"Air quality 38.7223, -9.1393."	0.69	Yes	N/A
"Air quality for 10001, US."	0.97	No	Put correct parameters but also `city`, `lat`, `lon` without value.
"AQI now at 90210, US."	0.63	Yes	N/A
"Pollution level 60614, US."	0.74	Yes	N/A
"AQI 10115, DE."	0.65	Yes	N/A
"Air quality SW1A 1AA, GB."	0.58	Yes	N/A
"AQI 75001, FR."	0.90	Yes	N/A
"Air quality 1250-096, PT."	0.59	Yes	N/A
"AQI 2000, AU."	0.65	No	Didn't put postal code, also put a city name that is correct with the country but not the location.
"Air quality 01000-000, BR."	0.65	Yes	N/A
"AQI 110001, IN."	0.57	No	Put correct country code in parameter `state_code` and inferred parameter `country_code` incorrectly.
"Current AQI Kyoto, JP."	0.50	Yes	N/A
"Air quality Toronto, CA now."	0.60	Yes	N/A
"AQI Nairobi?"	0.55	Yes	N/A
"Air quality in Auckland, NZ."	0.79	Yes	N/A
"AQI Reykjavik, IS (sorry for prev typo)."	0.82	Yes	N/A
"Pollution level Lima, PE."	0.64	Yes	N/A
"AQI Johannesburg, ZA."	1.39	Yes	N/A
"Air quality Munich, DE."	0.49	Yes	N/A
"AQI San Francisco, US-CA."	0.71	Yes	N/A
"Air quality Porto, PT."	0.82	Yes	N/A

Table A.14 Launch Application

Utterance	Time(s)	Correctness	Notes
"Open Steam."	1.17	Yes	N/A
"Launch dis cord."	0.67	Yes	Corrected the name of the application.
"Start Eclipse IDE."	0.63	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open VS Code."	0.59	No	Put parameter `is_sure_after_multiple_matches` incorrectly. Extended the name of the application correctly.
"Run visual studio code pls."	0.65	Yes	N/A
"Launch VSC."	0.56	Yes	Extended the name of the application correctly.
"Open Win RAR."	0.48	Yes	N/A
"Start WinRAR."	0.56	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Google Chorme."	0.55	Yes	N/A
"Launch chrome browser."	0.76	Yes	N/A
"Open Firefox."	0.51	Yes	N/A
"Start Notepad."	0.59	Yes	N/A
"Launch calc."	0.65	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open VLC."	0.66	Yes	N/A
"Start Spotify."	0.47	Yes	N/A
"Open Adobe Reader."	0.61	Yes	N/A
"Launch Acrobat."	0.50	Yes	N/A
"Open Slack."	0.84	No	Interpreted function call as conversation.
"Start MS Teams."	0.55	Yes	N/A
"Open Outlok."	0.48	Yes	Corrected the name of the application.
"Launch Word."	0.60	Yes	N/A
"Open Excel."	0.56	Yes	N/A
"Start Power Point."	0.76	Yes	N/A
"Open Paint."	0.51	Yes	N/A
"Start Edge browser."	0.64	Yes	N/A
"Launch Git Extensions."	0.55	Yes	N/A
"Open 7Zip."	0.51	Yes	N/A
"Start IntelliJ."	0.42	Yes	N/A
"Open Pychram."	0.81	Yes	N/A
"Launch Android Studio."	0.53	Yes	N/A

A.4.3 Gemini 2.0 Flash

Table A.15 Normal Conversation

Utterance	Time(s)	Correctness	Notes
"Explain the difference between threads and processes in one paragraph."	2,64	Yes	N/A
"Give me 3 meal prep ideas for a vegetarian who trains for half-marathons."	3,30	Yes	N/A
"Summarize the key idea of 'inversion' from mental models (keep it short)."	0,71	Yes	N/A
"What are common pitfalls when using Python's asyncio with CPU-bound work?"	3,95	Yes	N/A
"Draft a friendly email asking for a project deadline extension."	1,99	Yes	N/A
"What's a good daily warm-up routine for desk workers? Keep it under 8 steps."	2,35	Yes	N/A
"I keep forgetting people's names—share 4 memory techniques."	2,53	Yes	N/A
"Turn this into a bullet list: 'Plan, execute, measure, iterate.' Add one emoji each."	0,76	Yes	N/A
"Write a tiny story (<=120 words) about a lighthouse learning Morse code."	2,25	Yes	N/A
"Explain CAP theorem like I'm new to distributed systems."	3,42	Yes	N/A
"Compare SQLite vs PostgreSQL for a solo desktop app."	1,19	Yes	N/A
"How would you unit test a function that parses CSV lines? Keep it high level."	5,17	Yes	N/A
"What's the minimum I need to know about UX heuristics to not mess up my UI?"	4,00	Yes	N/A
"I'm anxious before presentations—give me 5 quick tips."	2,11	Yes	N/A
"Rewrite 'optimize the pipeline' in plainer language."	0,93	Yes	N/A
"Brainstorm 6 team-building activities for remote developers."	2,62	Yes	N/A
"What's the trade-off between early abstraction and duplication?"	1,32	Yes	N/A
"Explain why floating-point math can be surprising to newcomers."	2,58	Yes	N/A
"Give me a one-liner pep talk for debugging at 2 a.m."	0,62	Yes	N/A
"List 5 git hygiene practices for small teams."	2,26	Yes	N/A
"What's a simple analogy for gradient descent?"	1,43	Yes	N/A
"Suggest 4 interview questions to test problem decomposition skills."	1,99	Yes	N/A
"How do I politely push back on scope creep?"	2,68	Yes	N/A
"Convert this to title case: 'an introduction to concurrency primitives'"	1,04	Yes	N/A
"Name 5 signs a backlog item isn't ready for dev."	2,16	Yes	N/A
"What's a good rubric to decide whether to refactor now or later?"	6,66	Yes	N/A
"Give me a tiny regex cheat sheet (anchors, groups, classes)."	3,07	Yes	N/A
"What's the 80/20 of Docker I should know to ship a Python app?"	4,39	Yes	N/A
"Explain the difference between correlation and causation with one example."	1,07	Yes	N/A
"Turn 'I will try' into 3 stronger alternatives."	0,63	Yes	N/A

Table A.16 Get Current Weather

Utterance	Time(s)	Correctness	Notes
"What's the weather in Lisbon right now?"	2,10	Yes	N/A
"Current weather for 90210, US."	1,13	Yes	N/A
"Weather at -23.5505, -46.6333 please."	0,68	Yes	N/A
"Temperature and humidity in London."	0,97	Yes	N/A
"Show current weather in Cairo."	0,94	Yes	N/A
"Tell me if it's raining in Berlin right now."	0,92	Yes	N/A
"Weather near 48.8566, 2.3522."	0,71	Yes	N/A
"Current temperature in Sydney, AU."	0,96	Yes	N/A
"Weather in New York City."	1,13	Yes	N/A
"Is it sunny in Madrid right now?"	1,05	Yes	N/A
"Show me the weather in Nairobi, KE."	1,40	Yes	N/A
"Temperature in Toronto, CA."	0,80	Yes	N/A
"Weather near 34.0522, -118.2437."	1,67	Yes	N/A
"Current conditions in Tokyo, JP."	0,96	Yes	N/A
"Weather report for 1250-096, PT."	0,92	Yes	N/A
"Tell me if it's windy in Cape Town."	0,74	Yes	N/A
"Current temperature for 10001, US."	0,62	Yes	N/A
"What's the weather like in Mexico City?"	0,83	Yes	N/A
"Is it cloudy in Helsinki?"	0,97	Yes	N/A
"Weather now in Rome, IT."	0,69	Yes	N/A
"Temperature in Warsaw, PL."	0,84	Yes	N/A
"Is it snowing in Oslo?"	1,02	Yes	N/A
"Show weather for 10115, DE."	0,93	Yes	N/A
"Weather update for Seoul."	0,94	Yes	N/A
"Current weather for 2000, AU."	1,75	Yes	N/A
"Is it hot in Bangkok?"	0,73	Yes	N/A
"Show me the weather at 60614, US."	1,00	Yes	N/A
"Weather for Porto, PT."	1,06	Yes	N/A
"What's the weather in São Paulo, BR-SP?"	0,84	No	Put country and state code together with city name in parameter `city`.
"Weather in Reykjavik, IS."	1,02	Yes	N/A

Table A.17 Get Forecast

Utterance	Time(s)	Correctness	Notes
"5-day forecast for 40.7128, -74.0060."	2,01	Yes	N/A
"Next 5 days near -33.8688, 151.2093?"	1,03	Yes	N/A
"Forecast (5d) at -23.5505, -46.6333."	0,76	Yes	N/A
"Show me 5-day outlook for 30.0444, 31.2357."	0,77	Yes	N/A
"What's the week ahead at 55.7558, 37.6173?"	0,99	Yes	N/A
"Five-day forecast around -1.2921, 36.8219."	1,43	Yes	N/A
"Is it cooling later this week near 64.1466, -21.9426?"	0,76	Yes	N/A
"Give 5-day hi/lo for -33.9249, 18.4241."	0,84	Yes	N/A
"Forecast next days at 35.6895, 139.6917."	1,09	Yes	N/A
"Five-day for 38.7223, -9.1393 (Lisbon)."	0,98	Yes	N/A
"5-day forecast 10001, US."	0,93	Yes	N/A
"Forecast for 90210, US next 5 days."	0,64	Yes	N/A
"Weather outlook 60614, US."	0,91	Yes	N/A
"5-day for 10115, DE."	0,83	Yes	N/A
"Next five days SW1A 1AA, GB."	1,54	Yes	N/A
"Forecast 75001, FR."	0,80	Yes	N/A
"5-day for 1250-096, PT."	1,06	Yes	N/A
"Forecast for 2000, AU."	1,04	Yes	N/A
"5-day outlook 01000-000, BR."	1,23	Yes	N/A
"Next 5 days 110001, IN."	1,04	Yes	N/A
"Five-day forecast Kyoto, JP."	0,84	Yes	N/A
"What's the 5-day in Toronto, CA?"	0,79	Yes	N/A
"5-day forecast Nairobi."	0,96	Yes	N/A
"Week ahead in Auckland, NZ."	0,94	Yes	N/A
"5-day for Reykjavik, IS (typo earlier)."	0,65	Yes	N/A
"Forecast 5 days Lima, PE."	1,18	Yes	N/A
"Next 5 days Johannesburg, ZA."	0,88	Yes	N/A
"Munich, DE 5-day outlook."	0,60	Yes	N/A
"San Francisco, US-CA 5-day forecast."	0,71	No	Put country and state code together with city name in parameter `city`.
"5-day for Porto, PT."	0,63	Yes	N/A

Table A.18 Get Air Pollution

Utterance	Time(s)	Correctness	Notes
"Current air quality at 40.7128, -74.0060?"	1,95	Yes	N/A
"AQI near -33.8688, 151.2093 right now."	1,01	Yes	N/A
"What's PM2.5 at -23.5505, -46.6333?"	0,73	Yes	N/A
"Air pollution for 30.0444, 31.2357."	1,13	Yes	N/A
"AQI around 55.7558, 37.6173 (Moscow)."	1,17	Yes	N/A
"Current AQI for -1.2921, 36.8219."	0,89	Yes	N/A
"Is air clean near 64.1466, -21.9426?"	0,84	Yes	N/A
"Pollution stats at -33.9249, 18.4241."	1,14	Yes	N/A
"AQI now 35.6895, 139.6917."	1,04	Yes	N/A
"Air quality 38.7223, -9.1393."	0,82	Yes	N/A
"Air quality for 10001, US."	0,83	Yes	N/A
"AQI now at 90210, US."	0,94	Yes	N/A
"Pollution level 60614, US."	0,84	Yes	N/A
"AQI 10115, DE."	0,89	Yes	N/A
"Air quality SW1A 1AA, GB."	1,18	Yes	N/A
"AQI 75001, FR."	0,74	Yes	N/A
"Air quality 1250-096, PT."	0,94	Yes	N/A
"AQI 2000, AU."	1,12	Yes	N/A
"Air quality 01000-000, BR."	1,29	Yes	N/A
"AQI 110001, IN."	0,89	Yes	N/A
"Current AQI Kyoto, JP."	1,02	Yes	N/A
"Air quality Toronto, CA now."	1,03	Yes	N/A
"AQI Nairobi?"	0,88	Yes	Inferred parameter `country_code` correctly.
"Air quality in Auckland, NZ."	0,79	Yes	N/A
"AQI Reykjavik, IS (sorry for prev typo)."	0,81	Yes	N/A
"Pollution level Lima, PE."	0,95	Yes	N/A
"AQI Johannesburg, ZA."	1,12	Yes	N/A
"Air quality Munich, DE."	0,93	Yes	N/A
"AQI San Francisco, US-CA."	0,84	Yes	N/A
"Air quality Porto, PT."	0,83	Yes	N/A

Table A.19 Launch Application

Utterance	Time(s)	Correctness	Notes
"Open Steam."	2,03	Yes	N/A
"Launch dis cord."	0,95	Yes	Corrected the name of the application.
"Start Eclipse IDE."	0,71	Yes	N/A
"Open VS Code."	1,34	Yes	N/A
"Run visual studio code pls."	0,68	Yes	N/A
"Launch VSC."	1,27	Yes	N/A
"Open Win RAR."	0,74	Yes	Corrected the name of the application.
"Start WinRAR."	0,73	Yes	N/A
"Open Google Chorme."	1,11	Yes	N/A
"Launch chrome browser."	0,74	Yes	N/A
"Open Firefox."	1,17	Yes	N/A
"Start Notepad."	0,69	Yes	N/A
"Launch calc."	0,74	Yes	N/A
"Open VLC."	0,98	Yes	N/A
"Start Spotify."	0,68	Yes	N/A
"Open Adobe Reader."	0,54	Yes	N/A
"Launch Acrobat."	1,32	Yes	N/A
"Open Slack."	0,92	Yes	N/A
"Start MS Teams."	0,82	Yes	N/A
"Open Outlok."	0,75	Yes	Corrected the name of the application.
"Launch Word."	0,65	Yes	N/A
"Open Excel."	1,00	Yes	N/A
"Start Power Point."	0,84	Yes	N/A
"Open Paint."	0,76	Yes	N/A
"Start Edge browser."	0,90	Yes	N/A
"Launch Git Extensions."	0,82	Yes	N/A
"Open 7Zip."	1,08	Yes	N/A
"Start IntelliJ."	0,79	Yes	Put parameter `is_sure_after_multiple_matches` correctly, but in other cases didn't.
"Open Pychram."	0,73	Yes	N/A
"Launch Android Studio."	1,04	Yes	N/A

A.4.4 Qwen 2.5 7B Instruct

Table A.20 Normal Conversation

Utterance	Time(s)	Correctness	Notes
"Explain the difference between threads and processes in one paragraph."	2,30	Yes	N/A
"Give me 3 meal prep ideas for a vegetarian who trains for half-marathons."	1,19	Yes	N/A
"Summarize the key idea of 'inversion' from mental models (keep it short)."	0,93	Yes	N/A
"What are common pitfalls when using Python's asyncio with CPU-bound work?"	1,30	Yes	N/A
"Draft a friendly email asking for a project deadline extension."	1,49	No	Interpreted conversation as function call.
"What's a good daily warm-up routine for desk workers? Keep it under 8 steps."	3,18	Yes	N/A
"I keep forgetting people's names—share 4 memory techniques."	1,67	Yes	N/A
"Turn this into a bullet list: 'Plan, execute, measure, iterate.' Add one emoji each."	0,89	No	Interpreted conversation as function call.
"Write a tiny story (<=120 words) about a lighthouse learning Morse code."	1,19	No	Interpreted conversation as function call.
"Explain CAP theorem like I'm new to distributed systems."	1,81	Yes	N/A
"Compare SQLite vs PostgreSQL for a solo desktop app."	1,66	Yes	N/A
"How would you unit test a function that parses CSV lines? Keep it high level."	1,52	Yes	N/A
"What's the minimum I need to know about UX heuristics to not mess up my UI?"	0,89	Yes	N/A
"I'm anxious before presentations—give me 5 quick tips."	1,26	Yes	N/A
"Rewrite 'optimize the pipeline' in plainer language."	0,73	Yes	N/A
"Brainstorm 6 team-building activities for remote developers."	0,84	No	Interpreted conversation as function call.
"What's the trade-off between early abstraction and duplication?"	1,02	Yes	N/A
"Explain why floating-point math can be surprising to newcomers."	1,26	Yes	N/A
"Give me a one-liner pep talk for debugging at 2 a.m."	0,56	Yes	N/A
"List 5 git hygiene practices for small teams."	1,25	Yes	N/A
"What's a simple analogy for gradient descent?"	2,04	Yes	N/A
"Suggest 4 interview questions to test problem decomposition skills."	0,75	No	Interpreted conversation as function call.
"How do I politely push back on scope creep?"	0,79	Yes	N/A
"Convert this to title case: 'an introduction to concurrency primitives'"	0,78	No	Interpreted conversation as function call.
"Name 5 signs a backlog item isn't ready for dev."	0,74	Yes	N/A
"What's a good rubric to decide whether to refactor now or later?"	2,54	Yes	N/A
"Give me a tiny regex cheat sheet (anchors, groups, classes)."	1,52	Yes	N/A
"What's the 80/20 of Docker I should know to ship a Python app?"	2,98	Yes	N/A
"Explain the difference between correlation and causation with one example."	1,44	Yes	N/A
"Turn 'I will try' into 3 stronger alternatives."	0,66	Yes	N/A

Table A.21 Get Current Weather

Utterance	Time(s)	Correctness	Notes
"Current weather at 40.7128, -74.0060?"	2,05	Yes	N/A
"What's it like now near -33.8688, 151.2093 (Sydny)?"	1,53	Yes	Inferred parameter `units` with value `standard`.
"Weather rn at -23.5505, -46.6333."	0,90	Yes	N/A
"Temp & conditions at 30.0444, 31.2357 please."	0,96	Yes	Inferred parameter `units` with value `standard`.
"What's the current weather @ 55.7558, 37.6173?"	0,74	Yes	N/A
"Is it raining around -1.2921, 36.8219 right now?"	0,79	Yes	N/A
"How cold is it near 64.1466, -21.9426?"	1,17	Yes	Inferred parameter `units` with value `metric`.
"Current wind & temp for -33.9249, 18.4241."	1,04	Yes	N/A
"Weather now at 35.6895, 139.6917 (Tokio)?"	0,92	Yes	N/A
"What's the weather like at 38.7223, -9.1393?"	1,35	Yes	Inferred parameter `units` with value `metric`.
"Current weather for zip 10001, US."	1,10	Yes	N/A
"Weather now at 90210, US?"	1,17	Yes	N/A
"How's Chicago right now—60614, US."	1,37	Yes	N/A
"What's it like at 10115, DE (Berlin)?"	1,00	Yes	N/A
"Weather for SW1A 1AA, GB (Buckingham)?"	0,95	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Now in 75001, FR?"	0,93	Yes	N/A
"Current weather for 1250-096, PT (Lisboa)?"	0,82	Yes	Inferred parameter `units` with value `metric`.
"What's it now at 2000, AU (Sydney CBD)?"	0,93	Yes	Inferred parameters `units, lang` with values `standard, en`.
"Weather rn at 01000-000, BR (São Paulo)."	0,94	Yes	Inferred parameter `units` with value `metric`.
"Current conditions for 110001, IN (New Delhi)."	0,93	Yes	Inferred parameter `units` with value `metric`.
"What's the weather in Kyoto, JP right now?"	0,99	Yes	N/A
"Current conditions in Toronto, CA?"	1,18	Yes	Inferred parameter `units` with value `imperial`.
"How's Nairobi today?"	1,28	Yes	N/A
"Weather now in Auckland, NZ."	0,80	Yes	N/A
"What's it like in Reykyavik, IS?"	0,82	Yes	N/A
"Current weather for Lima, PE."	0,77	Yes	N/A
"How's Jo'burg (Johannesburg, ZA) right now?"	0,79	Yes	N/A
"Weather in Munich, DE (now)."	0,76	Yes	N/A
"Current conditions San Francisco, US-CA."	0,79	Yes	N/A
"What's the weather in Porto, PT?"	0,73	Yes	N/A

Table A.22 Get Forecast

Utterance	Time(s)	Correctness	Notes
"Weather forecast for New York for the next 5 days."	1,77	Yes	N/A
"Show me the 3-day forecast for Sydney."	0,92	Yes	N/A
"What's the forecast for São Paulo tomorrow?"	1,24	Yes	N/A
"Can I get the 7-day forecast for Cairo?"	1,28	Yes	N/A
"Forecast for Moscow, please."	0,78	Yes	N/A
"Give me forecast at coordinates 48.8566, 2.3522 (Paris)."	0,84	No	Inverted parameters `lan, lon` values.
"Next 5 days weather at Nairobi."	1,03	Yes	N/A
"Forecast for Cape Town this week."	0,75	Yes	N/A
"What's expected weather in Tokyo this weekend?"	0,91	Yes	N/A
"Can you show the next 3 days in Lisbon?"	0,95	Yes	N/A
"Forecast for zip 10001, US."	0,82	Yes	N/A
"5-day forecast for 90210, US."	0,82	Yes	N/A
"Tomorrow in Chicago, 60614."	0,75	Yes	N/A
"Weather forecast for Berlin, DE (10115)."	1,33	Yes	N/A
"Forecast for London, UK."	0,84	Yes	N/A
"Give me 5-day forecast for Lisbon, PT."	0,69	Yes	N/A
"Forecast in Sydney (2000, AU)."	0,73	Yes	N/A
"Show 3-day forecast for São Paulo (01000-000, BR)."	0,92	Yes	Converted Location given by postal code to latitude/longitude, kept parameter `country_code`.
"Forecast for New Delhi, IN (110001)."	1,03	Yes	N/A
"Next 7 days in Kyoto, JP."	0,81	Yes	N/A
"Forecast for Toronto, CA."	0,82	Yes	N/A
"Weather for Johannesburg, ZA (5 days)."	0,92	No	Put country code in parameter `state_code`, but inferred parameter `units` with value `standard`.
"5-day forecast for Lima, PE."	0,63	Yes	N/A
"Forecast in Reykjavik, IS (metric units)."	0,93	Yes	Inferred parameter `units` with value `metric`.
"Forecast for Auckland, NZ."	0,93	Yes	N/A
"Next days weather for Munich, DE."	0,83	Yes	N/A
"Forecast for Porto, PT."	0,82	Yes	N/A
"Weather in San Francisco, US for next 5 days."	0,83	Yes	N/A
"Forecast for Paris, FR."	0,94	Yes	N/A
"Next 7 days in Rome, IT."	1,02	Yes	N/A

Table A.23 Get Air Pollution

Utterance	Time(s)	Correctness	Notes
"Current air quality at 40.7128, -74.0060?"	1,79	Yes	N/A
"AQI near -33.8688, 151.2093 right now."	1,24	Yes	N/A
"What's PM2.5 at -23.5505, -46.6333?"	0,93	Yes	N/A
"Air pollution for 30.0444, 31.2357."	0,82	Yes	N/A
"AQI around 55.7558, 37.6173 (Moscow)."	0,96	Yes	N/A
"Current AQI for -1.2921, 36.8219."	0,83	Yes	N/A
"Is air clean near 64.1466, -21.9426?"	0,81	Yes	N/A
"Pollution stats at -33.9249, 18.4241."	0,67	Yes	N/A
"AQI now 35.6895, 139.6917."	0,70	No	Inverted parameters `lan, lon` values.
"Air quality 38.7223, -9.1393."	0,82	Yes	N/A
"Air quality for 10001, US."	0,82	Yes	N/A
"AQI now at 90210, US."	1,08	Yes	N/A
"Pollution level 60614, US."	0,89	Yes	N/A
"AQI 10115, DE."	0,82	Yes	N/A
"Air quality SW1A 1AA, GB."	0,82	Yes	N/A
"AQI 75001, FR."	0,86	Yes	N/A
"Air quality 1250-096, PT."	0,91	Yes	N/A
"AQI 2000, AU."	0,69	Yes	N/A
"Air quality 01000-000, BR."	0,88	Yes	N/A
"AQI 110001, IN."	0,82	Yes	N/A
"Current AQI Kyoto, JP."	0,86	Yes	N/A
"Air quality Toronto, CA now."	1,41	Yes	N/A
"AQI Nairobi?"	0,87	Yes	Inferred parameter `country_code` correctly
"Air quality in Auckland, NZ."	0,72	Yes	N/A
"AQI Reykjavik, IS (sorry for prev typo)."	1,09	Yes	N/A
"Pollution level Lima, PE."	0,79	Yes	N/A
"AQI Johannesburg, ZA."	0,77	No	Put parameter `zip` without value.
"Air quality Munich, DE."	0,85	Yes	N/A
"AQI San Francisco, US-CA."	0,79	Yes	N/A
"Air quality Porto, PT."	0,80	Yes	N/A

Table A.24 Launch Application

Utterance	Time(s)	Correctness	Notes
"Open Steam."	2,00	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Launch dis cord."	0,73	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start Eclipse IDE."	0,66	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open VS Code."	0,78	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Run visual studio code pls."	0,94	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Launch VSC."	0,86	Yes	N/A
"Open Win RAR."	0,89	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start WinRAR."	0,80	Yes	N/A
"Open Google Chorme."	0,87	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Launch chrome browser."	0,70	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Firefox."	1,07	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start Notepad."	2,07	Yes	N/A
"Launch calc."	1,12	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open VLC."	0,83	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start Spotify."	1,13	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Adobe Reader."	0,72	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Launch Acrobat."	0,83	Yes	N/A
"Open Slack."	1,03	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start MS Teams."	1,04	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Outlok."	0,71	Yes	N/A
"Launch Word."	0,79	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Excel."	0,93	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start Power Point."	0,92	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Paint."	0,95	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Start Edge browser."	0,77	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Launch Git Extensions."	0,76	Yes	N/A
"Open 7Zip."	0,83	Yes	N/A
"Start IntelliJ."	0,83	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Open Pychram."	0,80	No	Put parameter `is_sure_after_multiple_matches` incorrectly.
"Launch Android Studio."	0,75	Yes	N/A

A.4.5 Qwen 2.5 7B Instruct with function calling

Table A.25 Normal Conversation

Utterance	Time(s)	Correctness	Notes
"Explain the difference between threads and processes in one paragraph."	1.89	Yes	N/A
"Give me 3 meal prep ideas for a vegetarian who trains for half-marathons."	2.66	Yes	N/A
"Summarize the key idea of 'inversion' from mental models (keep it short)."	0.85	Yes	N/A
"What are common pitfalls when using Python's asyncio with CPU-bound work?"	3.32	Yes	N/A
"Draft a friendly email asking for a project deadline extension."	2.40	Yes	N/A
"What's a good daily warm-up routine for desk workers? Keep it under 8 steps."	4.13	Yes	N/A
"I keep forgetting people's names—share 4 memory techniques."	2.41	Yes	N/A
"Turn this into a bullet list: 'Plan, execute, measure, iterate.' Add one emoji each."	0.67	Yes	N/A
"Write a tiny story (<=120 words) about a lighthouse learning Morse code."	1.47	Yes	N/A
"Explain CAP theorem like I'm new to distributed systems."	3.32	Yes	N/A
"Compare SQLite vs PostgreSQL for a solo desktop app."	3.75	Yes	N/A
"How would you unit test a function that parses CSV lines? Keep it high level."	4.58	Yes	N/A
"What's the minimum I need to know about UX heuristics to not mess up my UI?"	3.87	Yes	N/A
"I'm anxious before presentations—give me 5 quick tips."	1.86	Yes	N/A
"Rewrite 'optimize the pipeline' in plainer language."	0.69	Yes	N/A
"Brainstorm 6 team-building activities for remote developers."	2.36	Yes	N/A
"What's the trade-off between early abstraction and duplication?"	2.51	Yes	N/A
"Explain why floating-point math can be surprising to newcomers."	3.16	Yes	N/A
"Give me a one-liner pep talk for debugging at 2 a.m."	0.87	Yes	N/A
"List 5 git hygiene practices for small teams."	1.50	Yes	N/A
"What's a simple analogy for gradient descent?"	0.97	Yes	N/A
"Suggest 4 interview questions to test problem decomposition skills."	1.90	Yes	N/A
"How do I politely push back on scope creep?"	2.30	Yes	N/A
"Convert this to title case: 'an introduction to concurrency primitives'"	0.86	Yes	N/A
"Name 5 signs a backlog item isn't ready for dev."	1.53	Yes	N/A
"What's a good rubric to decide whether to refactor now or later?"	3.30	Yes	N/A
"Give me a tiny regex cheat sheet (anchors, groups, classes)."	3.17	Yes	N/A
"What's the 80/20 of Docker I should know to ship a Python app?"	3.84	Yes	N/A
"Explain the difference between correlation and causation with one example."	2.63	Yes	N/A
"Turn 'I will try' into 3 stronger alternatives."	0.81	Yes	N/A

Table A.26 Get Current Weather

Utterance	Time(s)	Correctness	Notes
"Current weather at 40.7128, -74.0060?"	1.60	Yes	Inferred parameters `units, lang` with values `standard, en`.
"What's it like now near -33.8688, 151.2093 (Sydny)?"	1.05	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Weather rn at -23.5505, -46.6333."	0.94	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Temp & conditions at 30.0444, 31.2357 please."	0.98	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's the current weather @ 55.7558, 37.6173?"	0.90	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Is it raining around -1.2921, 36.8219 right now?"	0.82	Yes	Inferred parameters `units, lang` with values `metric, en`.
"How cold is it near 64.1466, -21.9426?"	0.87	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Current wind & temp for -33.9249, 18.4241."	0.94	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Weather now at 35.6895, 139.6917 (Tokio)?"	1.05	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's the weather like at 38.7223, -9.1393?"	0.83	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Current weather for zip 10001, US."	0.66	Yes	N/A
"Weather now at 90210, US?"	0.99	Yes	Inferred parameters `units, lang` with values `imperial, en`.
"How's Chicago right now—60614, US."	0.89	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's it like at 10115, DE (Berlin)?"	1.06	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Weather for SW1A 1AA, GB (Buckingham)?"	0.81	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Now in 75001, FR?"	0.75	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Current weather for 1250-096, PT (Lisboa)?"	0.79	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's it now at 2000, AU (Sydney CBD)?"	1.03	No	Inferred parameter `state_code` incorrectly and in forbidden situation, but inferred parameters `units, lang` with values `metric, en`.
"Weather rn at 01000-000, BR (São Paulo)."	0.73	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Current conditions for 110001, IN (New Delhi)."	0.87	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's the weather in Kyoto, JP right now?"	0.77	Yes	Inferred parameter `lang` with value `en`.
"Current conditions in Toronto, CA?"	0.85	No	Inferred parameter `state_code` incorrectly and in forbidden situation, but inferred parameters `units, lang` with values `metric, en`.
"How's Nairobi today?"	0.87	Yes	Inferred parameter `country_code` correctly and inferred parameters `units, lang` with values `metric, en`.
"Weather now in Auckland, NZ."	0.84	Yes	Inferred parameter `lang` with value `en`.
"What's it like in Reykyavik, IS?"	0.72	Yes	N/A
"Current weather for Lima, PE."	0.74	Yes	N/A
"How's Jo'burg (Johannesburg, ZA) right now?"	0.72	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Weather in Munich, DE (now)."	0.83	Yes	Inferred parameter `lang` with value `en`.
"Current conditions San Francisco, US-CA."	0.87	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's the weather in Porto, PT?"	0.63	Yes	N/A

Table A.27 Get Forecast

Utterance	Time(s)	Correctness	Notes
"5-day forecast for 40.7128, -74.0060."	10.76	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Next 5 days near -33.8688, 151.2093?"	1.06	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Forecast (5d) at -23.5505, -46.6333."	0.83	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Show me 5-day outlook for 30.0444, 31.2357."	1.05	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's the week ahead at 55.7558, 37.6173?"	0.89	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Five-day forecast around -1.2921, 36.8219."	0.78	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Is it cooling later this week near 64.1466, -21.9426?"	0.99	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Give 5-day hi/lo for -33.9249, 18.4241."	1.03	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Forecast next days at 35.6895, 139.6917."	1.08	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Five-day for 38.7223, -9.1393 (Lisbon)."	0.90	Yes	Inferred parameters `units, lang` with values `metric, en`.
"5-day forecast 10001, US."	0.92	Yes	Converted location given by postal code to city, also inferred parameters `units, lang` with values `imperial, en`.
"Forecast for 90210, US next 5 days."	0.96	Yes	Inferred parameters `units, lang` with values `imperial, en`.
"Weather outlook 60614, US."	1.10	Yes	Inferred parameter `lang` with value `en`.
"5-day for 10115, DE."	0.82	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Next five days SW1A 1AA, GB."	0.83	No	Put incomplete postal code as city name, but inferred parameter `lang` with value `en`.
"Forecast 75001, FR."	0.91	Yes	Inferred parameters `units, lang` with values `metric, fr`.
"5-day for 1250-096, PT."	0.82	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Forecast for 2000, AU."	1.04	Yes	Inferred parameters `units, lang` with values `metric, en`.
"5-day outlook 01000-000, BR."	0.82	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Next 5 days 110001, IN."	1.08	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Five-day forecast Kyoto, JP."	0.80	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's the 5-day in Toronto, CA?"	0.74	No	Put correct country code in parameter `state_code` and inferred parameter `country_code` incorrectly, but inferred parameters `units, lang` with values `metric, en`.
"5-day forecast Nairobi."	0.84	Yes	Inferred parameters `units, lang` with values `metric, en` and inferred correct country code.
"Week ahead in Auckland, NZ."	1.06	Yes	Inferred parameters `units, lang` with values `metric, en`.
"5-day for Reykjavik, IS (typo earlier)."	1.04	Yes	Inferred parameter `lang` with value `en`.
"Forecast 5 days Lima, PE."	0.92	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Next 5 days Johannesburg, ZA."	0.96	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Munich, DE 5-day outlook."	0.86	Yes	Inferred parameter `lang` with value `en`.
"San Francisco, US-CA 5-day forecast."	0.71	Yes	Inferred parameters `units, lang` with values `metric, en`.
"5-day for Porto, PT."	0.79	Yes	Inferred parameters `units, lang` with values `metric, en`.

Table A.28 Get Air Pollution

Utterance	Time(s)	Correctness	Notes
"Current air quality at 40.7128, -74.0060?"	1.44	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI near -33.8688, 151.2093 right now."	1.40	Yes	Inferred parameters `units, lang` with values `metric, en`.
"What's PM2.5 at -23.5505, -46.6333?"	0.89	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Air pollution for 30.0444, 31.2357."	0.89	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI around 55.7558, 37.6173 (Moscow)."	0.98	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Current AQI for -1.2921, 36.8219."	0.94	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Is air clean near 64.1466, -21.9426?"	0.96	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Pollution stats at -33.9249, 18.4241."	0.95	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI now 35.6895, 139.6917."	0.75	No	Interpreted function call as conversation.
"Air quality 38.7223, -9.1393."	1.18	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Air quality for 10001, US."	1.08	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI now at 90210, US."	1.34	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Pollution level 60614, US."	0.73	No	Put postal code as city name, but inferred parameters `units, lang` with values `metric, en`.
"AQI 10115, DE."	1.18	No	Interpreted function call as conversation.
"Air quality SW1A 1AA, GB."	0.92	No	Put postal code as city name.
"AQI 75001, FR."	1.08	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Air quality 1250-096, PT."	0.83	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI 2000, AU."	1.34	No	Interpreted function call as conversation.
"Air quality 01000-000, BR."	0.71	Yes	N/A
"AQI 110001, IN."	0.84	No	Put postal code as city name.
"Current AQI Kyoto, JP."	0.82	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Air quality Toronto, CA now."	0.76	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI Nairobi?"	0.83	Yes	Inferred correct country code and inferred parameters `units, lang` with values `metric, en`.
"Air quality in Auckland, NZ."	0.80	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI Reykjavik, IS (sorry for prev typo)."	0.67	Yes	N/A
"Pollution level Lima, PE."	0.83	Yes	Inferred parameters `units, lang` with values `metric, en`.
"AQI Johannesburg, ZA."	0.96	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Air quality Munich, DE."	0.65	Yes	N/A
"AQI San Francisco, US-CA."	0.81	Yes	Inferred parameters `units, lang` with values `metric, en`.
"Air quality Porto, PT."	1.03	Yes	Inferred parameters `units, lang` with values `metric, en`.

Table A.29 Launch Application

Utterance	Time(s)	Correctness	Notes
"Open Steam."	1.28	Yes	N/A
"Launch dis cord."	0.72	Yes	Corrected the name of the application.
"Start Eclipse IDE."	0.63	Yes	N/A
"Open VS Code."	0.84	Yes	N/A
"Run visual studio code pls."	0.65	Yes	N/A
"Launch VSC."	0.66	Yes	N/A
"Open Win RAR."	0.74	Yes	Corrected the name of the application.
"Start WinRAR."	0.67	Yes	N/A
"Open Google Chorme."	1.06	Yes	N/A
"Launch chrome browser."	0.67	Yes	N/A
"Open Firefox."	0.77	Yes	N/A
"Start Notepad."	0.70	Yes	N/A
"Launch calc."	0.78	Yes	N/A
"Open VLC."	0.62	Yes	N/A
"Start Spotify."	0.69	Yes	N/A
"Open Adobe Reader."	0.78	Yes	N/A
"Launch Acrobat."	0.69	Yes	N/A
"Open Slack."	0.68	Yes	N/A
"Start MS Teams."	0.58	Yes	N/A
"Open Outlok."	0.70	Yes	Corrected the name of the application.
"Launch Word."	0.63	Yes	N/A
"Open Excel."	0.68	Yes	N/A
"Start Power Point."	0.65	Yes	N/A
"Open Paint."	0.64	Yes	N/A
"Start Edge browser."	0.71	Yes	N/A
"Launch Git Extensions."	0.65	Yes	N/A
"Open 7Zip."	0.67	Yes	N/A
"Start IntelliJ."	0.74	Yes	N/A
"Open Pychram."	0.69	Yes	N/A
"Launch Android Studio."	0.57	Yes	N/A

Annex B

API Endpoints and Integration Details

This annex lists the external APIs and corresponding endpoints used by the functionality modules of the developed digital assistant. Each section includes a brief description of what the API does, the type of request made, and the main parameters and response data. All requests were performed via HTTP `GET` requests using Python's requests library, and keys were securely managed through environment variables.

B.1 OpenWeatherMap API

The OpenWeatherMap API was used to provide meteorological and environmental data to the digital assistant. It enables both location resolution through geocoding and access to live weather, forecasts, and air pollution information.
The API can be accessed at `https://openweathermap.org/api`
The base URL for all endpoints is:

```
https://api.openweathermap.org
```

B.1.1 Geocoding API

The geocoding API was used to translate user-provided location references into standardized geographic coordinates and to retrieve location details (city and country) when only coordinates were available. This ensured that every meteorological query handled by the assistant consistently included `latitude`, `longitude`, the location name (parameter `city`), and `country`.
Three geocoding endpoints were used: two for direct geocoding (based on location name or zip code), and one for reverse geocoding (from coordinates to location name).

- **Coordinates by location name:**

  ```
  /geo/1.0/direct
  ```

  API call: https://api.openweathermap.org/geo/1.0/direct?q={city name},{state code},{country code}&limit={limit}&appid={API key}
  Parameters:

- `q` required City name, state code (only for the US) and country code divided by comma. It uses ISO 3166 country codes.
- `appid` required API key.
- `limit` optional Number of the locations in the API response (up to 5 results can be returned in the API response).

Description: Converts a textual location name into geographical coordinates. It is used when the user specifies a location in natural language, e.g., "weather in Porto."
Field in API response:

- `name`: Name of the found location.
- `local_names`:
  - `local_names.[language code]`: Name of the found location in different languages. The list of names can be different for different locations.
  - `local_names.ascii`: Internal field.
  - `local_names.feature_name`: Internal field.
- `lat`: Geographical coordinates of the found location (latitude).
- `lon`: Geographical coordinates of the found location (longitude).
- `country`: Country of the found location.
- `state`: (where available) State of the found location.

- **Coordinates by zip/post code**

  `/geo/1.0/zip`

API call: https://api.openweathermap.org/geo/1.0/zip?zip={zip code},{country code}
&appid={API key}
Parameters:

- `zip code` required Zip/post code and country code divided by comma. It uses ISO 3166 country codes.
- `appid` required API key.

Description: Resolves geographic coordinates using postal codes, supporting users who specify a ZIP or postal code instead of a city name.
Field in API response:

- `zip`: Specified zip/post code in the API request.
- `name`: Name of the found area.
- `lat`: Geographical coordinates of the centroid of found zip/post code (latitude).
- `lon`: Geographical coordinates of the centroid of found zip/post code (longitude).
- `country`: Country of the found zip/post code.

- **Reverse Geocoding Endpoint:**

  `/geo/1.0/reverse`

  API call: https://api.openweathermap.org/geo/1.0/reverse?lat={lat}&lon={lon}&limit={limit}&appid={API key}
  Parameters:

  - `lat, lon` required Geographical coordinates (latitude, longitude).
  - `appid` required API key.
  - `limit` optional Number of the location names in the API response (several results can be returned in the API response).

  Description: Converts geographical coordinates into corresponding city and country names. This is used to complement weather responses with human-readable location identifiers.
  Field in API response:

  - `name`: Name of the found location.
  - `local_names`:
    - `local_names.[language code]`: Name of the found location in different languages. The list of names can be different for different locations.
    - `local_names.ascii`: Internal field.
    - `local_names.feature_name`: Internal field.
  - `lat`: Geographical coordinates of the found location (latitude).
  - `lon`: Geographical coordinates of the found location (longitude).
  - `country`: Country of the found location.
  - `state`: (where available) State of the found location.

B.1.2 OpenWeatherMap Functionality Endpoints

The functionalities of the digital assistant that rely on meteorological information use the OpenWeatherMap API's data endpoints, all under the base endpoint:

`/data/2.5`

These functionalities use the parameters that were obtained from the geocoding API, in this case, `latitude` and `longitude` are used for the requests, while `city` and `country` are included in the response. For each functionality, a specific endpoint is used based on the purpose:

1. **Current weather:**

   `/weather`

API call: https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}

Parameters:

- `lat` required Latitude.
- `lon` required Longitude.
- `appid` required API key.
- `mode` optional Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default.
- `units` optional Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default.
- `lang` optional You can use this parameter to get the output in your language.

Description: Provides current weather data for a specific location.

API responses:

(a) **JSON format API response fields**:

- `coord`:
  - `coord.lon`: Longitude of the location.
  - `coord.lat`: Latitude of the location.
- `weather`:
  - `weather.id`: Weather condition id.
  - `weather.main`: Group of weather parameters (Rain, Snow, Clouds etc.).
  - `weather.description`: Weather condition within the group.
  - `weather.icon`: Weather icon id.
- `base`: Internal parameter.
- `main`:
  - `main.temp`: Temperature. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.feels_like`: Temperature. This temperature parameter accounts for the human perception of weather. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.pressure`: Atmospheric pressure on the sea level, hPa.
  - `main.humidity`: Humidity, %.
  - `main.temp_min`: Minimum temperature at the moment. This is minimal currently observed temperature (within large megalopolises and urban areas). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.

  - `main.temp_max`: Maximum temperature at the moment. This is maximal currently observed temperature (within large megalopolises and urban areas). Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
  - `main.sea_level`: Atmospheric pressure on the sea level, hPa.
  - `main.grnd_level`: Atmospheric pressure on the ground level, hPa.
- `visibility`: Visibility, meter. The maximum value of the visibility is 10 km.
- `wind`:
  - `wind.speed`: Wind speed. Unit Default: meter/sec, Metric: meter/sec, Imperial: miles/hour.
  - `wind.deg`: Wind direction, degrees (meteorological).
  - `wing.gust`: Wind gust. Unit Default: meter/sec, Metric: meter/sec, Imperial: miles/hour.
- `clouds`:
  - `clouds.all`: Cloudiness, %.
- `rain`:
  - `1h`: (where available) Precipitation, mm/h. Please note that only mm/h as units of measurement are available for this parameter.
- `snow`:
  - `1h`: (where available) Precipitation, mm/h. Please note that only mm/h as units of measurement are available for this parameter.
- `dt`: Time of data calculation, unix, UTC.
- `sys`:
  - `sys.type`: Internal parameter
  - `sys.id`: Internal parameter
  - `sys.message`: Internal parameter
  - `sys.country`: Country code (GB, JP etc.).
  - `sys.sunrise`: Sunrise time, unix, UTC.
  - `sys.sunset`: Sunrise time, unix, UTC.
- `timezone`: Shift in seconds from UTC.
- `id`: City ID. Please note that built-in geocoder functionality has been deprecated.
- `name`: City name. Please note that built-in geocoder functionality has been deprecated.
- `cod`: Internal parameter.

`XML format API response fields`:

- `city`:
  - `city.id`: City ID. Please note that built-in geocoder functionality has been deprecated.

  - `city.name`: City name. Please note that built-in geocoder functionality has been deprecated.
  - `city.coord`:
    * `city.coord.lon`: Geo location, longitude
    * `city.coord.lan`: Geo location, latitude
  - `city.country`: Country code (GB, JP etc.). Please note that built-in geocoder functionality has been deprecated.
  - `timezone`: Shift in seconds from UTC.
  - `city.sun`:
    * `city.sun.rise`: Sunrise time.
    * `city.sun.set`: Sunset time.
- `temperature`:
  - `temperature.value`: Temperature.
  - `temperature.min`: Minimum temperature at the moment of calculation. This is minimal currently observed temperature (within large megalopolises and urban areas), use this parameter optionally.
  - `temperature.max`: Maximum temperature at the moment of calculation. This is maximal currently observed temperature (within large megalopolises and urban areas), use this parameter optionally.
  - `temperature.unit`: Unit of measurements. Possible value is Celsius, Kelvin, Fahrenheit.
- `feels_like`:
  - `feels_like.value`: Temperature. This temperature parameter accounts for the human perception of weather.
  - `feels_like.unit`: Unit of measurements. Possible value is Celsius, Kelvin, Fahrenheit. Unit Default: Kelvin.
- `humidity`:
  - `humidity.value`: Humidity value.
  - `humidity.unit`: Humidity units, %.
- `pressure`:
  - `pressure.value`: Pressure value.
  - `pressure.unit`: Pressure units, hPa.
- `wind`:
  - `wind.speed`:
    * `wind.speed.value`: Wind speed.
    * `wind.speed.unit`: Wind speed units, m/s.
    * `wind.speed.name`: Type of the wind.
  - `wind.direction`:

        * `wind.direction.value`: Wind direction, degrees (meteorological).
        * `wind.direction.code`: Code of the wind direction. Possible value is WSW, N, S etc.
        * `wind.direction.name`: Full name of the wind direction.
    - `clouds`:
        - `clouds.value`: Cloudiness.
        - `clouds.name`: Name of the cloudiness.
    - `visibility`:
        - `visibility.value`: Visibility, meter. The maximum value of the visibility is 10 km.
    - `precipitation`:
        - `precipitation.value`: Precipitation, mm. Please note that only mm as units of measurement are available for this parameter.
        - `precipitation.mode`: Possible values are 'no", name of weather phenomena as 'rain', 'snow'.
    - `weather`:
        - `weather.number`: Weather condition id.
        - `weather.value`: Weather condition name.
        - `weather.icon`: Weather icon id.
    - `lastupdate`:
        - `lastupdate.value`: Last time when data was updated.

2. **5 day weather forcast:**

    `/forecast`

    API call: https://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={API key}

    Parameters:

    - `lat` required Latitude.
    - `lon` required Longitude.
    - `appid` required API key.
    - `mode` optional Response format. JSON format is used by default. To get data in XML format use `mode=xml`.
    - `cnt` optional A number of timestamps, which will be returned in the API response.
    - `units` optional Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default.

- `lang` optional You can use this parameter to get the output in your language.

Description: Provides 5-day weather forecasts with 3-hour intervals.
API responses:

(a) **JSON format API response fields**

- `cod`: Internal parameter.
- `message`: Internal parameter.
- `cnt`: A number of timestamps returned in the API response.
- `list`:
  - `list.dt`: Time of data forecasted, unix, UTC.
  - `list.main`:
    - `list.main.temp`: Temperature. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.feels_like`: This temperature parameter accounts for the human perception of weather. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.temp_min`: Minimum temperature at the moment of calculation. This is minimal forecasted temperature (within large megalopolises and urban areas), use this parameter optionally. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.temp_max`: Maximum temperature at the moment of calculation. This is maximal forecasted temperature (within large megalopolises and urban areas), use this parameter optionally. Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit.
    - `list.main.pressure`: Atmospheric pressure on the sea level by default, hPa.
    - `list.main.sea_level`: Atmospheric pressure on the sea level, hPa.
    - `list.main.grnd_level`: Atmospheric pressure on the ground level, hPa.
    - `list.main.humidity`: Humidity, %.
    - `list.main.temp_kf`: Internal parameter
  - `list.weather`:
    - `list.weather.id`: Weather condition id.
    - `list.weather.main`: Group of weather parameters (Rain, Snow, Clouds etc.).
    - `list.weather.description`: Weather condition within the group.
    - `list.weather.icon`: Weather icon id.
  - `list.clouds`:
    - `list.clouds.all`: Cloudiness, %.

- `list.wind`:
  - `list.wind.speed`: Wind speed. Unit Default: meter/sec, Metric: meter/sec, Imperial: miles/hour.
  - `list.wind.deg`: Wind direction, degrees (meteorological).
  - `list.wing.gust`: Wind gust. Unit Default: meter/sec, Metric: meter/sec, Imperial: miles/hour.
- `list.visibility`: Average visibility, metres. The maximum value of the visibility is 10km.
- `list.pop`: Probability of precipitation. The values of the parameter vary between 0 and 1, where 0 is equal to 0%, 1 is equal to 100%.
- `list.rain`:
  - `list.rain.3h`: Rain volume for last 3 hours, mm. Please note that only mm as units of measurement are available for this parameter.
- `list.snow`:
  - `list.snow.3h`: Snow volume for last 3 hours. Please note that only mm as units of measurement are available for this parameter.
- `list.sys`:
  - `list.sys.pod`: Part of the day (n - night, d - day).
- `list.dt_txt`: Time of data forecasted, ISO, UTC.

- `city`:
  - `city.id`: City ID. Please note that built-in geocoder functionality has been deprecated.
  - `city.name`: City name. Please note that built-in geocoder functionality has been deprecated.
  - `city.coord`:
    - `city.coord.lat`: Geo location, latitude.
    - `city.coord.lon`: Geo location, longitude.
  - `city.country`: Country code (GB, JP etc.). Please note that built-in geocoder functionality has been deprecated.
  - `city.population`: City population.
  - `city.timezone`: Shift in seconds from UTC.
  - `city.sunrise`: Sunrise time, Unix, UTC.
  - `city.sunset`: Sunset time, Unix, UTC.

(b) **XML format API response fields**

- `location`:
  - `location.name`: City name. Please note that built-in geocoder functionality has been deprecated.
  - `location.type`: Internal parameter.

  - location.country: Country code (GB, JP etc.). Please note that built-in geocoder functionality has been deprecated.
  - location.timezone: Shift in seconds from UTC.
  - location.location:
    * location.location.altitude: Geo location, altitude above the sea level.
    * location.location.latitude: Geo location, latitude.
    * location.location.longitude: Geo location, longitude.
    * location.location.geobase: Internal parameter.
    * location.location.geobaseid: Internal parameter
- meta:
  - meta.lastupdate: Prototype parameter.
  - meta.calctime: Speed of data calculation.
  - meta.nextupdate: Prototype parameter.
- sun:
  - sun.rise: Sunrise time.
  - sun.set: Sunset time.
- forecast:
  - forecast.time:
    * forecast.time.from: Beginning of the period of data forecasted
    * forecast.time.to: End of the period of data forecasted
  - forecast.symbol:
    * forecast.symbol.number: Weather condition id.
    * forecast.symbol.name: Weather condition.
    * forecast.symbol.var: Weather icon id.
  - forecast.precipitation:
    * forecast.precipitation.probability: Probability of precipitation. The values of the parameter vary between 0 and 1, where 0 is equal to 0%, 1 is equal to 100%.
    * forecast.precipitation.unit: Period of measurements. Possible value is 1 hour, 3 hours.
    * forecast.precipitation.value: Precipitation volume for the last 3 hours, mm. Please note that only mm as units of measurement are available for this parameter.
    * forecast.precipitation.type: Type of precipitation. Possible value is rain, snow.
  - forecast.windDirection:
    * forecast.windDirection.deg: Wind direction, degrees (meteorological).

    - `forecast.windDirection.code`: Code of the wind direction. Possible value is WSW, N, S etc.
    - `forecast.windDirection.name`: Full name of the wind direction.
- `forecast.windSpeed`:
    - `forecast.windSpeed.mps`: Wind speed, meters per second.
    - `forecast.windSpeed.unit`: Wind speed units, m/s.
    - `forecast.windSpeed.name`: Type of wind.
- `forecast.windGust`:
    - `forecast.windGust.gust`: Wind gust, meters per second.
    - `forecast.windGust.unit`: Wind gust units, m/s.
- `forecast.temperature`:
    - `forecast.temperature.unit`: Unit of measurements. Possible value is Celsius, Kelvin, Fahrenheit.
    - `forecast.temperature.value`: Temperature.
    - `forecast.temperature.min`: Minimum temperature at the moment of calculation. This is minimal forecasted temperature (within large megalopolises and urban areas), use this parameter optionally.
    - `forecast.temperature.max`: Maximum temperature at the moment of calculation. This is maximal forecasted temperature (within large megalopolises and urban areas), use this parameter optionally.
- `forecast.feels_like`:
    - `forecast.feels_like.unit`: Unit of measurements. Possible value is Celsius, Kelvin, Fahrenheit. Unit Default: Kelvin.
    - `forecast.feels_like.value`: Temperature. This temperature parameter accounts for the human perception of weather.
- `forecast.pressure`
    - `forecast.pressure.unit`: hPa
    - `forecast.pressure.value`: Pressure value
- `forecast.humidity`:
    - `forecast.humidity.unit`: %.
    - `forecast.humidity.value`: Humidity value.
- `forecast.clouds`:
    - `forecast.pressure.value`: Name of the cloudiness
    - `forecast.pressure.all`: Cloudiness
    - `forecast.pressure.unit`: %.
- `forecast.visibility`
    - `forecast.visibility.value`: Average visibility, metres. The maximum value of the visibility is 10km.

3. **Current air pollution:**

   `/air_pollution`

   API call: https://api.openweathermap.org/data/2.5/air_pollution?lat={lat}&lon={lon}&appid={API key}

   Parameters:

   - `lat` required Latitude.
   - `lon` required Longitude.
   - `appid` required API key.

   Description: Provides air quality data based on geographic coordinates.
   Fields in API response:

   - `coord`: Coordinates from the specified location (latitude, longitude).
   - `list`:
     - `dt`: Date and time, Unix, UTC.
     - `main`:
       * `main.aqi`: Air Quality Index. Possible values: 1, 2, 3, 4, 5. Where 1 = Good, 2 = Fair, 3 = Moderate, 4 = Poor, 5 = Very Poor. If you want to recalculate Air Quality indexes according UK, Europe, USA and Mainland China scales.
     - `components`:
       * `components.co`: Concentration of CO (Carbon monoxide), $\mu g/m^3$.
       * `components.no`: Concentration of NO (Nitrogen moxoxide), $\mu g/m^3$.
       * `components.no2`: Concentration of NO_2 (Nitrogen dioxide), $\mu g/m^3$.
       * `components.o3`: Concentration of O_3 (Ozone), $\mu g/m^3$.
       * `components.so2`: Concentration of SO_2 (Sulphur dioxide), $\mu g/m^3$.
       * `components.pm2_5`: Concentration of $PM_{2.5}$ (Fine particulate matter), $\mu g/m^3$.
       * `components.pm10`: Concentration of PM_{10} (Coarse particulate matter), $\mu g/m^3$.
       * `components.nh3`: Concentration of NH_3 (Ammonia), $\mu g/m^3$.

B.2 Language Model APIs

The DA's LLM adapters are dependent on RESTful or OpenAI-compatible API. Each adapter communicates with its respective provider through standardized `POST` requests carrying structured JSON payloads that contain conversation history, model parameters, and generation settings. All authentication keys were managed through the `APIKeyManager` class and stored in a local `.env` file, keeping credentials separate from the source code and allowing flexible reconfiguration without code changes.

Awan Llama API

The Awan Llama model was accessed through the official AwanLLM REST API.

- **Base URL:** `https://api.awanllm.com`
- **Endpoint:** `/v1/chat/completions`
- **Method:** `POST`
- **Authentication:** Bearer token in the `Authorization` header
- **Request body:** Model name, conversation history (`messages`), and parameters such as `temperature`, `top_p`, and `max_tokens`.

Further documentation: `https://www.awanllm.com/docs`

Gemini API (Google)

The Gemini adapter integrates with Google's OpenAI-compatible Gemini endpoint using the same schema as OpenAI's `chat.completions` API.

- **Base URL:** `https://generativelanguage.googleapis.com/v1beta/openai/`
- **Endpoint:** `/chat/completions`
- **Method:** `POST`
- **Authentication:** Bearer token in the `Authorization` header
- **Request body:** Chat messages, temperature, and optional function/tool schemas.

Further documentation: `https://ai.google.dev/gemini-api/docs/openai`

Hugging Face Llama API

The Hugging Face adapter connects to Hugging Face's OpenAI-compatible Inference Router, which allows using models hosted on the platform with the same interface as OpenAI.

- **Base URL:** `https://router.huggingface.co/v1`
- **Endpoint:** `/chat/completions`
- **Method:** `POST`
- **Authentication:** Bearer token in the `Authorization` header
- **Request body:** Chat messages and generation parameters such as `temperature` and `top_p`.

Further documentation: `https://huggingface.co/docs/huggingface_hub/main/en/guides/inference#openai-compatibility`

Qwen API

The Qwen models were also accessed through Hugging Face's Inference Router, using the same OpenAI-compatible interface as the Hugging Face Llama configuration.

- **Base URL:** `https://router.huggingface.co/v1`
- **Endpoint:** `/chat/completions`
- **Method:** `POST`
- **Authentication:** Bearer token in the `Authorization` header
- **Request body:** Equivalent to Hugging Face Llama requests, supporting function-calling where available.

Further documentation: `https://huggingface.co/docs/huggingface_hub/main/en/guides/inference#openai-compatibility`

OpenAI-Compatible Specification

Each of the aforementioned API emulates or complies with the OpenAI `chat.completions` endpoint schema. The `LLMAdapter` interface, which defines a common abstraction, made it easier to implement the various adapters thanks to this shared request format. Although each adapter interacts with its provider on its own, the DA can handle messages and parse responses consistently thanks to the standardized API structure. Reference documentation: `https://platform.openai.com/docs/api-reference`

Annex C

Questionnaire

This annex presents the complete usability questionnaire administered to participants during the qualitative validation of the DA. The form was implemented using Google Forms and organized into sections corresponding to each evaluated functionality: weather retrieval, forecast, air pollution, application launching, and conversational ability. The final section included the System Usability Scale (SUS). The following figures reproduce the visual layout of the questionnaire as presented to users.

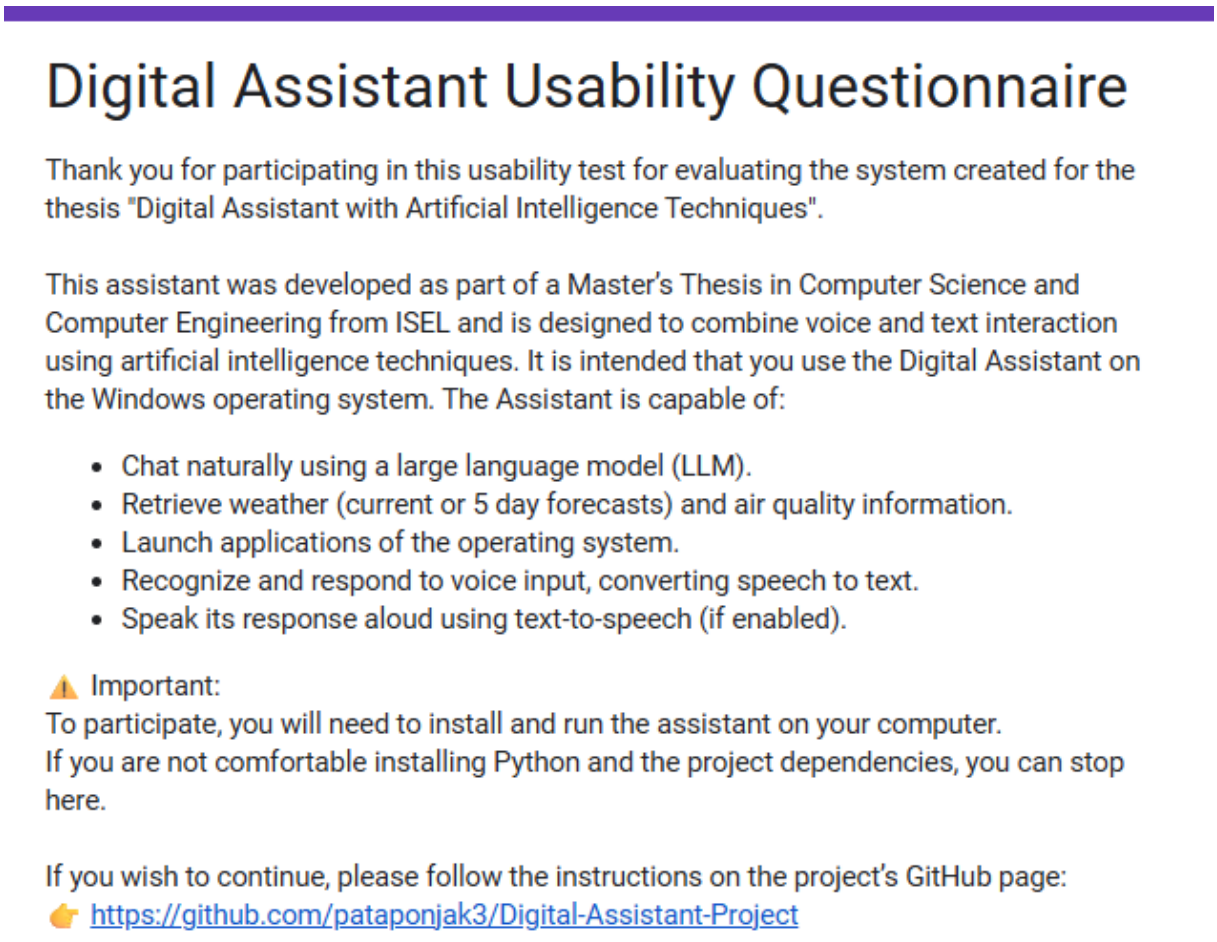

Figure C.1 Questionnaire Section 1

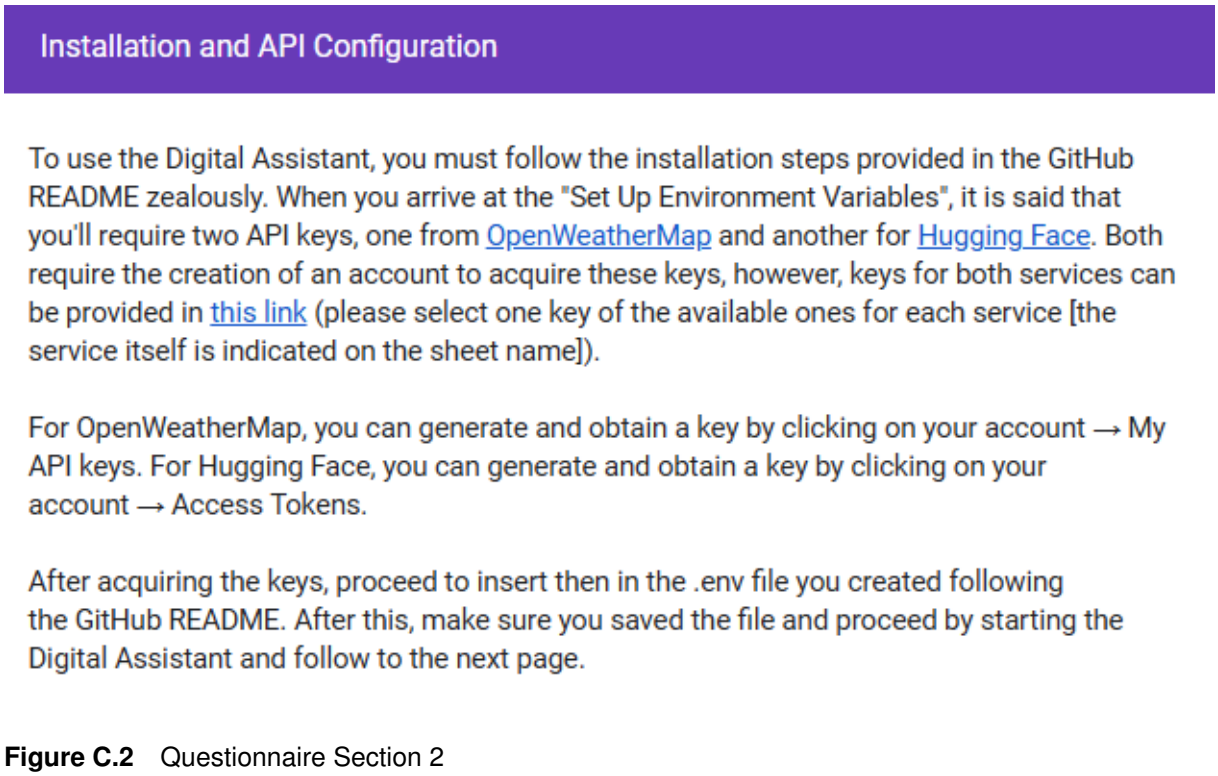

Figure C.2 Questionnaire Section 2

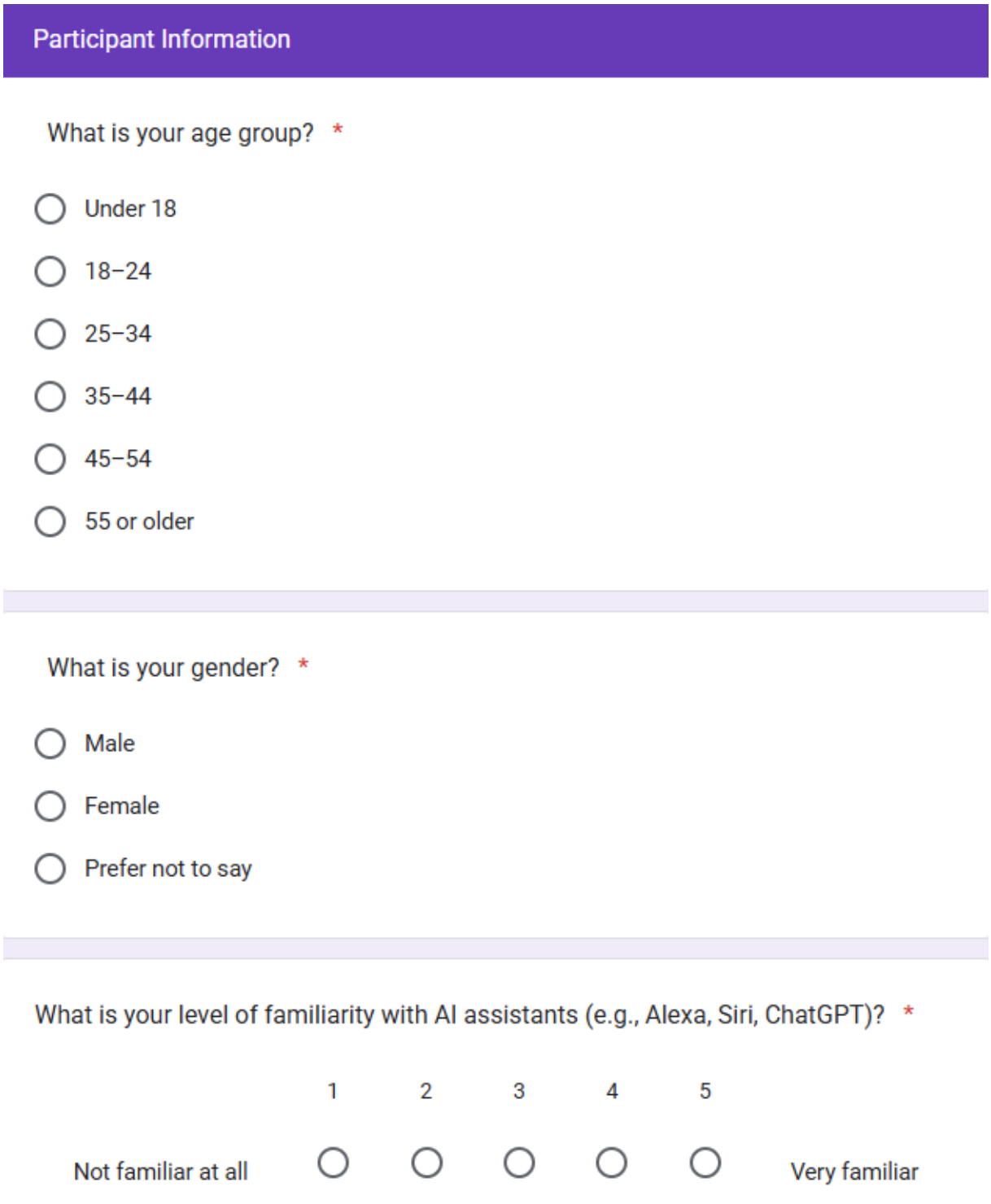

Figure C.3 Questionnaire Section 3

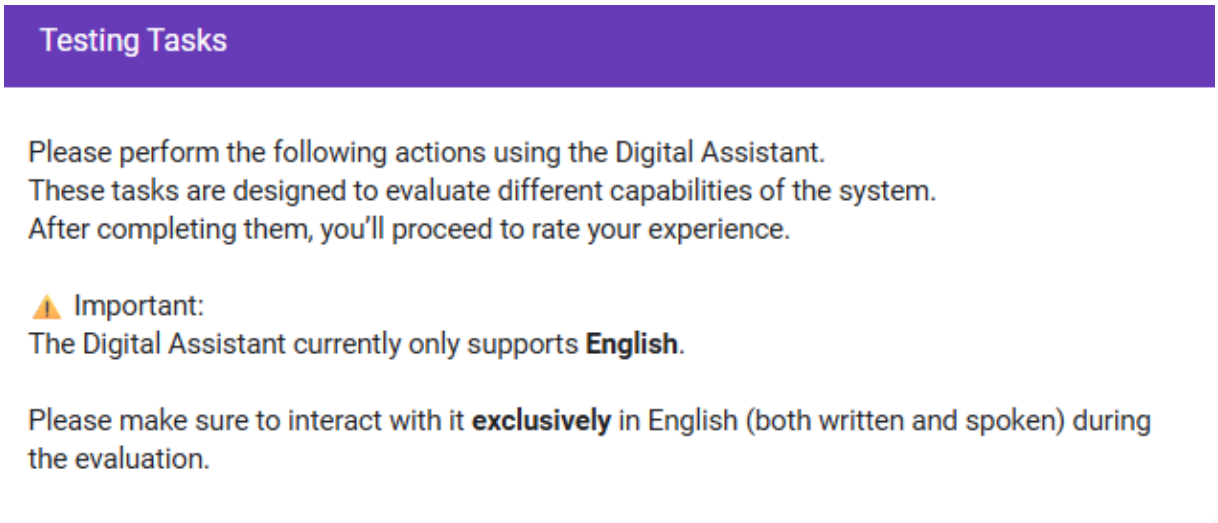

Figure C.4 Questionnaire Section 4

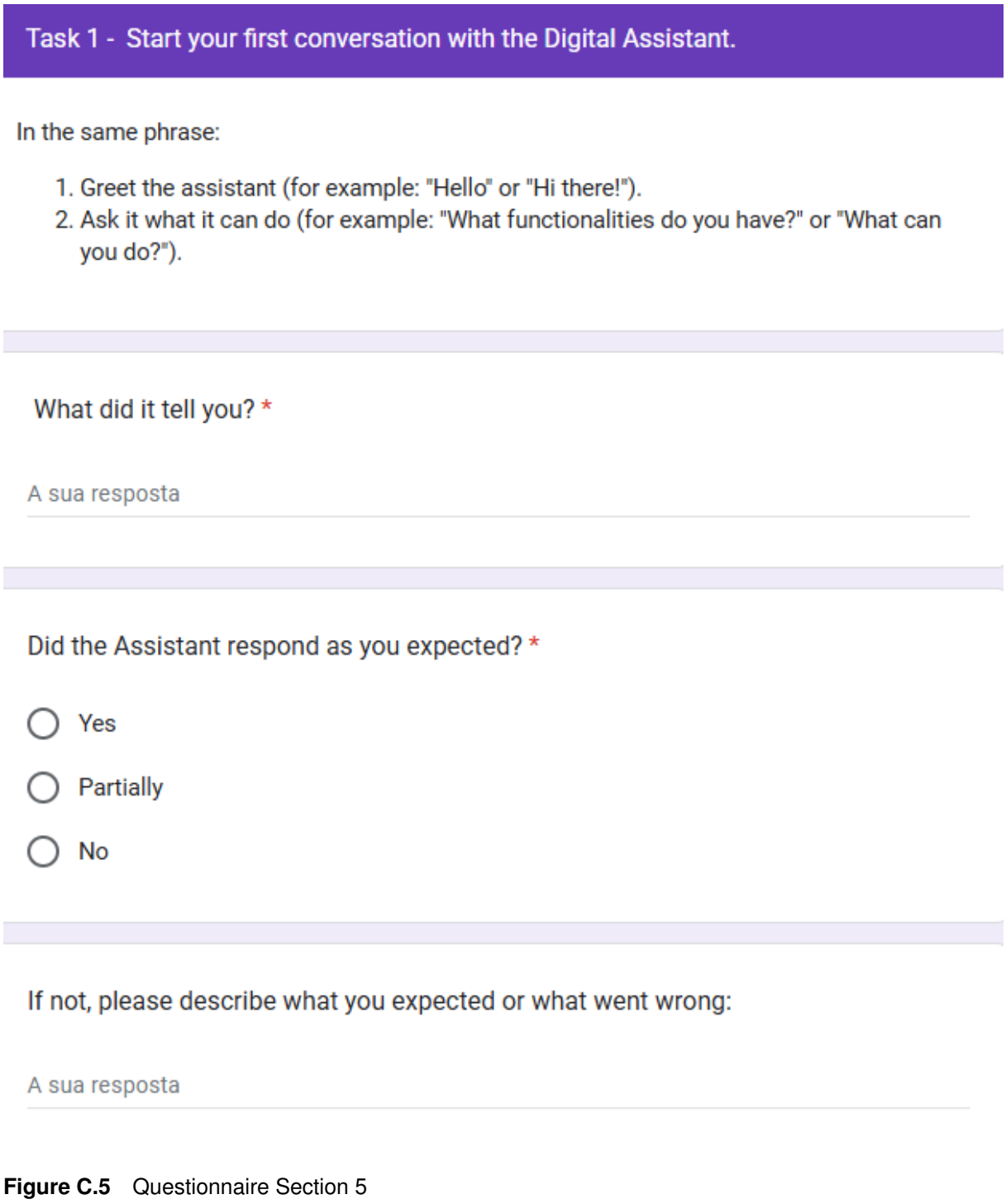

Figure C.5 Questionnaire Section 5

Task 2 - Get current weather

This task will test the Assistant's current weather functionality. Ask it for the current weather in a specific location of your choice.
For example, you can say: "What's the weather like in Lisbon right now?" or "Tell me the current weather in London.".

Did the Assistant correctly provide the weather information? *

- ◯ Yes
- ◯ Partially
- ◯ No

How would you rate the clarity of the response (how easy it was to understand)? *

	1	2	3	4	5	
Very unclear	◯	◯	◯	◯	◯	Very clear

How easy was it to get the weather information using the Assistant? *

	1	2	3	4	5	
Very difficult	◯	◯	◯	◯	◯	Very easy

Please describe any difficulties or unexpected behavior you experienced.

A sua resposta

Figure C.6 Questionnaire Section 6

Task 3 - Get the 5-Day Forecast (Using Voice Recognition, if possible)

This task will test the Assistant's speech recognition (if possible) and 5-day weather forecast functionalities.

Step 1 - Configure the microphone

Before starting, please ensure that your microphone is correctly configured:

1. Open the menu "Settings → Microphone Settings" in the Digital Assistant.
2. Select your available microphone from the list.
3. Save your configuration and close the window.

If the assistant does not detect a microphone or you do not have one, you can still complete the task by typing your request instead.

Were you able to select your microphone? *

- ◯ Yes
- ◯ No, because it was not showing as an option
- ◯ No, because I do not have a microphone
- ◯ Outra: _____

Step 2 - Ask for the forecast

If you were able to select your microphone, press the "Speak" button and say something like: "What's the 5-day forecast for Lisbon?" or "Give me the weather forecast for the next 5 days in Porto.".

If you were **not** able to select your microphone, type the same question in the text box and press "Send" or press the Enter key.

Figure C.7 Questionnaire Section 7 Part 1

Were you able to use the voice recognition feature? *

- ◯ Yes
- ◯ No (I had to type the request)

Did the Assistant understand your request correctly? *

- ◯ Yes, perfectly
- ◯ Partially (needed repetition or correction)
- ◯ No

Did the Assistant provide a 5-day forecast as expected? *

- ◯ Yes
- ◯ Partially
- ◯ No

How easy was it to use the voice recognition (if used)?

	1	2	3	4	5	
Very difficult	◯	◯	◯	◯	◯	Very easy

Please describe any difficulties or comments you had while performing this task:

A sua resposta

Figure C.8 Questionnaire Section 7 Part 2

Task 4 - Check Air Pollution (Using Text-to-Speech)

In this task, you will use the air pollution feature of the Digital Assistant and verify the text-to-speech output (spoken response).

Step 1 - Enable Voice Output

Before sending your message, make sure that speech output is enabled:

- At the top-left of the window, check the box "Enable Speech Output".
- If it's already checked, leave it as is.

This will allow the assistant to speak its answer aloud after responding in text.

Step 2 - Ask about air pollution

Type the following question in the text box and send it: "What's the air pollution in Lisbon?" You can replace Lisbon with any city you want.

The assistant should respond with information about air quality, such as an index value and/or a qualitative description (e.g., "moderate," "good," "unhealthy").

Step 3 - Listen to the response

After receiving the answer in the chat, the assistant should read it aloud using its text-to-speech feature.

If you do not hear any sound, please make sure your speakers or headphones are connected and the volume is up and retry again.

Did the assistant respond with information about air pollution? *

- ◯ Yes
- ◯ Partially (incomplete or unclear)
- ◯ No

Figure C.9 Questionnaire Section 8 Part 1

Did the assistant speak the response aloud? *

- ◯ Yes
- ◯ No

How would you rate the clarity of the voice output? *

	1	2	3	4	5	
Very unclear	◯	◯	◯	◯	◯	Very clear

Did the spoken response match the text response on screen? *

- ◯ Yes
- ◯ Mostly
- ◯ No

Please describe any issues you experienced with the voice output or air pollution response.

A sua resposta

Figure C.10 Questionnaire Section 8 Part 2

Task 5 – Launch an application

In this task, you will evaluate the launch applications feature of the Digital Assistant. It can only launch applications that appear in your Start Menu.

Before asking the assistant to open an application, please check that the application is present in your Start menu.

You can check the applications that are present in the Start Menu by looking through the list of applications in the Start button (the button with the Windows logo), or by searching in the search bar in the Start Menu.

The assistant might ask you to select from a list if there are multiple applications with similar names.

Step 1 - Choose an application and send the command

Type something like: "Can you launch the Word application?" or "Open Word.".
You can replace "Word" with any application that is present in the Start Menu.

Step 2 - Handle name conflicts and/or wait for the launch

If the assistant detects multiple possible matches, it will present a list of options. In that case, type the number or exact name of the application you want to open, like: "Select the first option." or "Launch option 3.".

Please note that launching the application might take a few seconds, depending on your system's speed and configuration.

If the program does not open after a short wait, proceed to the follow-up questions below.

Did the assistant successfully launch the application you requested? *

- ◯ Yes, it opened the correct one.
- ◯ It asked me to choose, and then it opened correctly.
- ◯ It asked me to choose, but none of the options worked.
- ◯ No, it did not open the application.
- ◯ Outra: _____

Figure C.11 Questionnaire Section 9 Part 1

How would you rate the ease of use of this functionality?

	1	2	3	4	5	
Very difficult	○	○	○	○	○	Very easy

How clear were the instructions or options the assistant provided when there were multiple matches?

	1	2	3	4	5	
Very unclear	○	○	○	○	○	Very clear

Did the delay between the command and the application opening feel acceptable?

- ○ Yes
- ○ Somewhat acceptable
- ○ No

Please describe any difficulties you experienced during this task:

A sua resposta

Figure C.12 Questionnaire Section 9 Part 2

Task 6 – Conversational Interaction (LLM Chat)

In this task, you will explore the Digital Assistant's conversational abilities, how it responds to general questions or engages in simple dialogue.

The goal is to assess how natural, coherent, and helpful its replies feel.

Step 1 - Start a conversation

Type something (e.g. "Tell me an interesting fact.")
Observe how the assistant responds.

Does it provide a coherent and relevant answer?

Step 2 - Continue the chat

Try asking follow-up or general knowledge questions, such as:

- "Who created Python?"
- "What's the capital of Japan?"
- "Can you tell me a joke?"
- "Explain what artificial intelligence is in simple terms."

You may also test contextual understanding by referencing earlier questions, for example: "And what about JavaScript?" (after asking about Python).

Were the assistant's responses relevant and understandable? *

	1	2	3	4	5	
Never	○	○	○	○	○	Always

Figure C.13 Questionnaire Section 10 Part 1

How natural did the assistant's replies feel (tone, wording, flow)? *

	1	2	3	4	5	
Very robotic	○	○	○	○	○	Very natural

Did the assistant seem to understand follow-up questions based on the previous context? *

- ○ Yes, consistently
- ○ Sometimes
- ○ Rarely
- ○ Not at all

Please describe your overall impression of the assistant's conversational ability:

A sua resposta

Figure C.14 Questionnaire Section 10 Part 2

Overall Usability Evaluation

Please rate each of the following statements based on your experience with the Digital Assistant. In this questionnaire, the term "functions" or "functionalities" refers to the features and capacities of the Digital Assistant, for example, its ability to interpret your messages, perform actions like checking the weather or opening applications, and respond to your inputs.

I think that I would like to use this system frequently. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I found the system unnecessarily complex. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I thought the system was easy to use. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Figure C.15 Questionnaire Section 11 Part 1

I think that I would need the support of a technical person to be able to use this system. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I found the various functions in this system were well integrated. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I thought there was too much inconsistency in this system. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I would imagine that most people would learn to use this system very quickly. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

Figure C.16 Questionnaire Section 11 Part 2

I found the system very cumbersome to use. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I felt very confident using the system. *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

I needed to learn a lot of things before I could get going with this system (this is related to its usage, not the installation part). *

	1	2	3	4	5	
Strongly Disagree	○	○	○	○	○	Strongly Agree

How would you rate the overall visual design of the Digital Assistant?

	1	2	3	4	5	
Very unappealing	○	○	○	○	○	Very appealing

Figure C.17 Questionnaire Section 11 Part 3

Additional Feedback

Here you can provide additional feedback regarding the Digital Assistant. It is important to note that the focus of such system is to evaluate its performance and functionalities and not necessarily the visual aspect (in this case specifically, not on how good it looks, but how good it feels), however, you can still indicate such things here.

What did you like most about the assistant?

A sua resposta

What did you find confusing or frustrating?

A sua resposta

Do you have any suggestions for improvement?

A sua resposta

Figure C.18 Questionnaire Section 12

Bibliography

[1] Ryen W. White, Adam Fourney, Allen Herring, Paul N. Bennett, Nirupama Chandrasekaran, Robert Sim, Elnaz Nouri, and Mark J. Encarnación. Multi-device digital assistance. *Communications of the ACM*, 62(10):28–31, 2019. doi: 10.1145/3357159.

[2] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966. doi: 10.1145/365153.365168.

[3] Shamala Gallagher, Anna Rafferty, and Amy Wu. Natural language processing: History, 2004. URL https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/nlp/overview_history.html. Accessed: 2025-10-29.

[4] Kenneth Mark Colby. *Artificial Paranoia: A Computer Simulation of Paranoid Processes*. Pergamon, 1975. ISBN 978-0-08-018162-2.

[5] Potential Staff. The evolution of chatbots: From eliza to ai assistants, 2024. URL https://www.potential.com/articles/the-evolution-of-chatbots-from-eliza-to-ai-assistants/. Accessed: 2025-10-29.

[6] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. Draft under preparation, 3rd edition, 2025. URL https://web.stanford.edu/~jurafsky/slp3/. Online manuscript released August 24, 2025.

[7] Richard H. Wiggins. Personal digital assistants. *Journal of Digital Imaging*, 17(1):5–17, 2004. doi: 10.1007/s10278-003-1665-8.

[8] Khalid Mohammad Jaber, Mohammed Laf, Ahmad Aa Alkhatib, Amani Khamis AbedAlghafer, Mohammad Abdul Jawad, and Amal Qassed Ahmad. A comparative study for virtual personal assistants (vpa) and state-of-the-art speech recognition technology. *International Journal of Computational and Experimental Science and Engineering*, 10(3):427–433, 2024. doi: 10.22399/ijcesen.383.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

[11] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pretraining. Technical report, OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[14] Harry McCracken. The bob chronicles, 2010. URL https://technologizer.com/2010/03/29/microsoft-bob/index.html. Accessed: 2025-10-29.

[15] TMAFE. What is microsoft agent?, 2018. URL https://tmafe.com/. Accessed: 2025-10-29.

[16] Jason Dookeran. 7 of the most unpopular windows features of all time, 2024. URL https://www.howtogeek.com/the-most-unpopular-windows-features-of-all-time/. Accessed: 2025-10-29.

[17] Janakiram MSV. Inside microsoft copilot: A look at the technology stack, 2023. URL https://www.forbes.com/sites/janakirammsv/2023/05/26/inside-microsoft-copilot-a-look-at-the-technology-stack/. Accessed: 2025-10-29.

[18] Vivek Pradeep. Introducing mu language model and how it enabled the agent in windows settings. Technical report, Microsoft, 2025. URL https://blogs.windows.com/windowsexperience/2025/06/23/introducing-mu-language-model-and-how-it-enabled-the-agent-in-windows-settings/.

[19] Dave Bergmann. What is machine learning?, 2025. URL https://www.ibm.com/think/topics/machine-learning. Accessed: 2025-10-29.

[20] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. ISBN 978-1-4939-3843-8.

[21] Dave Bergmann. What is deep learning?, 2025. URL https://www.ibm.com/think/topics/deep-learning. Accessed: 2025-10-29.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[23] Ivan Belcic and Cole Styker. What is natural language understanding (nlu)?, 2025. URL https://www.ibm.com/think/topics/natural-language-understanding. Accessed: 2025-10-29.

[24] IBM. What is speech recognition?, 2025. URL https://www.ibm.com/think/topics/speech-recognition. Accessed: 2025-10-29.

[25] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[26] Charlotte Hu and Amanda Downie. What is text to speech?, 2025. URL https://www.ibm.com/think/topics/text-to-speech. Accessed: 2025-10-29.

[27] Cole Stryker and Jim Holdsworth. What is nlp (natural language processing)?, 2025. URL https://www.ibm.com/think/topics/natural-language-processing. Accessed: 2025-10-29.

[28] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf.

[29] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783, 2018. doi: 10.1109/ICASSP.2018.8461368.

[30] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative

model for raw audio. In *9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, page 125, 2016.

[31] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management, 2017. URL https://arxiv.org/abs/1712.05181.

[32] Amazon. Alexa skills kit overview, 2023. URL https://developer.amazon.com/en-US/alexa. Accessed: 2025-10-29.

[33] IBM Corporation. Ibm watson assistant, 2024. URL https://www.ibm.com/watson/assistant/. Accessed: 2025-10-29.

[34] OpenAI. Function calling, 2023. URL https://platform.openai.com/docs/guides/function-calling. Accessed: 2025-10-29.

[35] Matthew B Hoy. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, 37(1):81–88, 2018. doi: 10.1080/02763869.2018.1404391.

[36] Cole Stryker and Eda Kavlakoglu. What is artificial intelligence (ai)?, 2025. URL https://www.ibm.com/think/topics/artificial-intelligence. Accessed: 2025-10-29.

[37] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. Guidelines for human-ai interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300233. URL https://doi.org/10.1145/3290605.3300233.

[38] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 4th edition, 2021. ISBN 9780136885979.

[39] Travis E. Oliphant. Python for scientific computing. *Computing in Science Engineering*, 9 (3):10–20, 2007. doi: 10.1109/MCSE.2007.58.

[40] Mark Summerfield. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Prentice Hall Press, USA, 1st edition, 2015. ISBN 0134393333.

[41] RapidFuzz. Rapidfuzz description in pypi, 2021. URL https://pypi.org/project/RapidFuzz/#description. Accessed: 2025-10-29.

[42] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 2002. ISBN 0201745720.

[43] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994. ISBN 9780080520292.

[44] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996. ISBN 0471958697.

[45] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. 01 2004. ISBN 9780735619678.

[46] OpenAI. Function calling - token usage, 2023. URL `https://platform.openai.com/docs/guides/function-calling#token-usage`. Accessed: 2025-10-29.