Assistente digital baseado em Inteligência Artificial para PC

ANTÓNIO DUARTE GOULÃO
(Licenciado)

Trabalho de Projeto para obtenção do grau de mestre em Engenharia Informática e de Computadores

Orientador(es):
Doutor Nuno Miguel da Costa de Sousa Leite
Doutor Artur Jorge Ferreira

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

Novembro de 2025

Assistente digital baseado em Inteligência Artificial para PC

ANTÓNIO DUARTE GOULÃO
(Licenciado)

Trabalho de Projeto para obtenção do grau de mestre em Engenharia Informática e de Computadores

Orientador(es):
Doutor Nuno Miguel da Costa de Sousa Leite, IPL/ISEL
Doutor Artur Jorge Ferreira, IPL/ISEL

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

Novembro de 2025

Agradecimentos

A realização desta tese só foi possível graças ao apoio, orientação e incentivo de várias pessoas, a quem gostaria de expressar a minha sincera gratidão.

Em primeiro lugar, agradeço aos meus orientadores, Professor Artur Ferreira e ao Professor Nuno Leite, pela disponibilidade, acompanhamento constante e pelas valiosas sugestões que contribuíram de forma decisiva para a concretização deste trabalho.

Agradeço também aos meus colegas Gonçalo Fernandes, José Jorge e Pedro Apolinário, pela partilha de ideias e apoio ao longo deste percurso.

Por fim, um agradecimento especial à minha família, pelo incentivo incondicional, compreensão e apoio contínuo em todas as etapas da minha formação académica. Sem eles, este trabalho não teria sido possível.

Declaração de integridade

Declaro que este trabalho de projeto é o resultado da minha investigação pessoal e independente. O seu conteúdo é original e todas as fontes listadas nas referências bibliográficas foram consultadas e estão devidamente mencionadas no texto. Mais declaro que todas as referências científicas e técnicas relevantes para o desenvolvimento do trabalho estão devidamente citadas e constam das referências bibliográficas.

O autor

Assinado por: **António Duarte Goulão**
Num. de Identificação: 15134828
Data: 2025.11.24 14:08:08+00'00'

Lisboa, 24 de Novembro de 2025

Resumo

Nos últimos anos, os assistentes digitais tornaram-se cada vez mais populares como meio de interação entre utilizadores e sistemas computacionais. No entanto, a maioria das soluções existentes é proprietária e fortemente integrada em ecossistemas fechados, limitando a flexibilidade e transparência. Esta tese propõe o desenvolvimento de um Assistente Digital (AD) para ambiente *desktop* Windows, modular, extensível e baseado em tecnologias de acesso aberto.

O sistema integra reconhecimento automático de fala (ASR), síntese de fala (TTS), processamento de linguagem natural (PLN) e uma interface gráfica interativa. A arquitetura modular permite substituir ou expandir funcionalidades sem comprometer o núcleo do sistema. Um modelo de linguagem em larga escala (LLM) é utilizado para interpretar comandos em linguagem natural, garantindo flexibilidade na compreensão de instruções.

Foram implementadas funcionalidades como execução de comandos locais e integração com serviços externos (Google Calendar e Gmail). Todos os comandos foram avaliados com LLM de diferentes dimensões. Os resultados mostraram que o desempenho está diretamente ligado ao modelo utilizado: modelos menores apresentaram falhas ocasionais, enquanto os de maior escala garantiram elevada precisão e consistência. Em todos os casos, os tempos médios de resposta mantiveram-se baixos, na ordem dos décimos de segundo.

Para avaliar a usabilidade, foi aplicado um questionário baseado na métrica SUS a 15 utilizadores, com resultados muito positivos (pontuação média de 90.33 em 100). Os participantes mostraram facilidade na execução das tarefas e sugeriram melhorias relevantes. A solução confirma a viabilidade de um AD modular, expansível e *open source* para *desktop*. O trabalho constitui uma base sólida para futuras evoluções, permitindo a integração de novos módulos e adoção de diferentes LLM, representando um passo relevante no desenvolvimento de assistentes digitais mais abertos e adaptáveis.

Palavras-chave

Assistente Digital Pessoal; Interação do Utilizador; Modelos de Linguagem de grande escala; Processamento de Linguagem Natural; Reconhecimento de Fala; Síntese de Fala

Abstract

In recent years, digital assistants have become increasingly popular as a means of interaction between users and computer systems. However, most existing solutions are proprietary and tightly integrated into closed ecosystems, limiting both flexibility and transparency. This thesis proposes the development of a Personal Digital Assistant (PDA) for the Windows desktop environment, designed to be modular, extensible, and based on open technologies.

The developed system integrates Automatic Speech Recognition (ASR), Text-to-Speech (TTS), Natural Language Processing (NLP), and an interactive graphical interface. Its modular architecture allows for the replacement or expansion of features without compromising the system core. A large language model (LLM) is used to interpret natural language commands, providing flexibility in understanding user instructions.

Features such as execution of local commands and integration with external services (Google Calendar and Gmail) were implemented. All commands were evaluated using LLMs of different sizes. The results showed that system performance is closely tied to the model used: smaller models exhibited occasional errors, while larger models provided high precision and consistency. In all scenarios, the assistant maintained low average response times, typically under one second.

To assess usability, a SUS-based questionnaire was conducted with 15 participants, yielding highly positive results (average score of 90.33 out of 100). Participants successfully completed the proposed tasks and provided relevant suggestions for future improvements. The proposed solution confirms the feasibility of a modular, extensible, and open-source PDA for desktop environments. The work lays a solid foundation for future evolution, enabling the integration of new modules and adoption of different LLMs according to the application context, marking a relevant step towards more open and adaptable digital assistants.

Keywords

Large Language Models; Natural Language Processing; Personal Digital Assistant; Speech Recognition; User Interaction; Text to Speech

Índice

Índice de figuras

Índice de tabelas

Acrónimos

AD	Assistente Digital
API	Application Programing Interface (Interface de Programação de Aplicação)
ASR	Automatic Speech Recognition (Reconhecimento Automático de Fala)
AV	Assistente Virtual
CNN	Convolutional Neural Network (Rede Neuronal Convolucional)
DTW	Dynamic Time Warping
GMM	Gaussian Mixture Model (Modelo de Mistura de Gaussianas)
GUI	Graphical User Interface (Interface gráfica do utilizador)
HMM	Hidden Markov Model (Modelo não observável de Markov)
IA	Inteligência Artificial
LLM	Large Language Model (Modelo de linguagem de grande escala)
LM	Language Model (Modelo de Linguagem)
MFCC	Mel-Frequency Cepstral Coefficients (coeficientes mel-cepstrais)
NER	Named Entity Recognition (Reconhecimento de Entidades Nomeadas)
PC	Personal Computer (Computador Pessoal)
PDA	Personal Digital Assistant (Assistente Digital Pessoal)
PLN	Processamento de Linguagem Natural
POS	Parts of Speech (Partes do Discurso)
RNN	Recurrent Neural Network (Rede Neuronal Recorrente)
SUS	System Usability Scale
WSD	Word Sense Disambiguation (Desambiguação do Sentido das Palavras)

Capítulo 1

Introdução

Nos últimos anos, a tecnologia tem evoluído significativamente no sentido da interação mais natural e intuitiva entre seres humanos e dispositivos digitais. O desenvolvimento de Assistentes Digitais (AD) ou Assistentes Virtuais (AV) reflete esta evolução. Estes agentes de software conseguem realizar uma variedade de tarefas com base em comandos e instruções fornecidos pelos utilizadores, de forma verbal ou escrita. O objetivo principal dos AD é simplificar e tornar mais eficiente a interação com a tecnologia, permitindo que determinadas tarefas sejam executadas de forma ágil, sem necessidade de conhecimento técnico por parte do utilizador.

Os AD são ferramentas versáteis, capazes de realizar uma variedade de tarefas, facilitando a vida dos utilizadores ao automatizar tarefas diárias e simplificar processos. Além de funções de gestão de agenda, como gerir calendários, criar lembretes, listas de tarefas e agendar reuniões, os AD também podem oferecer sugestões personalizadas com base nas preferências e no estilo de vida do utilizador. Por exemplo, podem sugerir o melhor momento para marcar uma reunião, com base na disponibilidade do utilizador, ou escrever um *e-mail* com base na escrita anterior do utilizador. No que diz respeito à comunicação, os AD podem realizar chamadas telefónicas e enviar ou escrever *e-mails*. Além disso, os AD podem interagir com sistemas de arquivos e aplicações no computador, criar, mover ou apagar pastas, bem como abrir páginas web e lançar aplicações com simples comandos de voz ou texto.

Atualmente, os AD estão disponíveis numa grande variedade de aplicações e plataformas, desde computadores pessoais e telemóveis até dispositivos inteligentes em residências e aparelhos conectados. Esta diversidade permite que as interações ocorram de múltiplas maneiras, seja por voz, texto, toque ou interfaces gráficas, oferecendo uma experiência cada vez mais natural e intuitiva.

Alguns dos assistentes digitais mais conhecidos e amplamente utilizados, como a Siri (Apple), Alexa (Amazon), Google Assistant, Bixby (Samsung) e o Microsoft Copilot (anteriormente Cortana), não só respondem a comandos básicos, mas conseguem interagir de forma conversacional, simulando diálogo humano. Estas funcionalidades são possibilitadas por inovações em Inteligência Artificial (IA), Processamento de Linguagem Natural (PLN) e Aprendizagem Profunda (Deep Learning), que permitem ao AD interpretar comandos complexos e adaptar as respostas de acordo com o contexto do utilizador. Assim, os AD têm sido integrados em

diversos cenários do quotidiano, tanto em ambientes domésticos [52] quanto profissionais [55], estabelecendo-se como uma interface tecnológica promissora.

Apesar do uso extenso de assistentes pessoais em dispositivos móveis e na nuvem, ainda há poucas alternativas *open source* específicas para *desktop Windows*. Ferramentas como o Microsoft Copilot e o mais recente Microsoft Mu oferecem automação avançada, mas operam em ambientes fechados e com integrações próprias.

1.1 Objetivos

Este trabalho foca-se no desenvolvimento de um AD nativo para Windows que seja modular, extensível via *plugins* e inteiramente *open source*, permitindo que qualquer programador possa trocar o motor de IA, adicionar novas funcionalidades e auditar o código sem dependências proprietárias.

Nesse sentido, a presente tese propõe o desenvolvimento de um Assistente Digital modular para PC, com interação por voz e por texto. A arquitetura adotada baseia-se em módulos independentes, o que permite integrar facilmente novos serviços ou substituir componentes sem alterar o núcleo do sistema. O protótipo foi desenvolvido de forma a explorar o potencial dos Modelos de Linguagem de Grande Escala (LLM), responsáveis por interpretar comandos em linguagem natural e convertê-los em ações estruturadas. A arquitetura geral do sistema é apresentada na Figura 1.1.

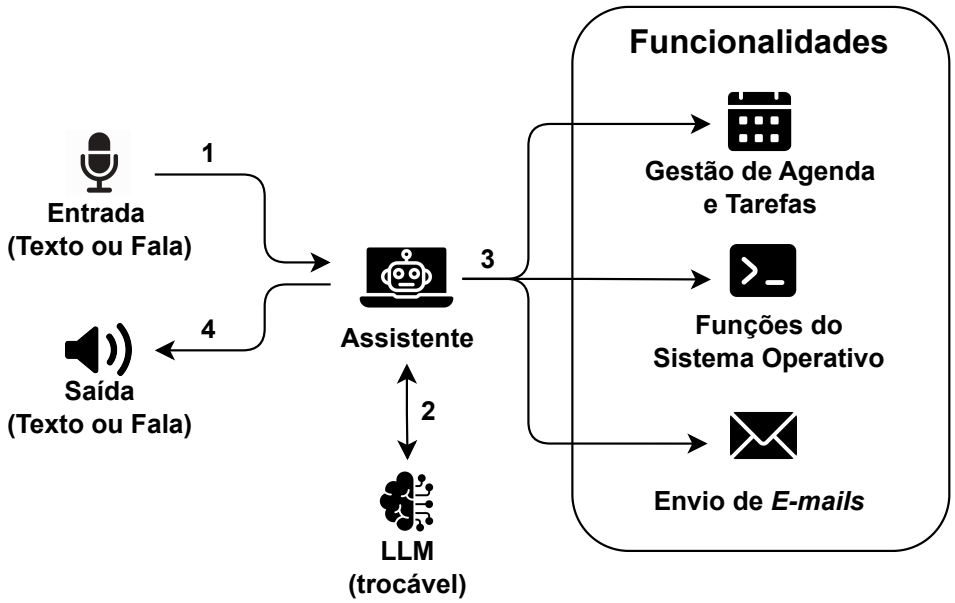

Figura 1.1 Diagrama da arquitetura geral do assistente digital 1) Utilizador envia um pedido em texto ou fala ao Assistente; 2) Assistente encaminha o pedido para LLM, responsável por interpretar a ação pretendida; 3) A ação interpretada é encaminhada para o módulo de Funcionalidades correspondente, responsável pela sua execução; 4) Assistente apresenta a resposta ao utilizador, em texto ou fala

O objetivo principal deste trabalho é disponibilizar um assistente aberto, flexível e extensível, que permita aos utilizadores interagir com o sistema operativo e com aplicações externas

de forma prática e eficiente. A modularidade garante que novas funcionalidades possam ser adicionadas incrementalmente, enquanto a integração com LLM assegura a interpretação de comandos complexos e a adaptação a diferentes estilos de comunicação.

Para concretizar este objetivo, o sistema foi desenvolvido de acordo com os seguintes requisitos.

Requisitos Funcionais

- Incluir funcionalidades de gestão de agenda e tarefas, como criação de eventos e lembretes;
- Executar comandos locais no sistema operativo, como manipulação de ficheiros e lançamento de aplicações;
- Integrar serviços de *e-mail*, permitindo ler e enviar mensagens;
- Disponibilizar uma interface visual que facilite a interação e a consulta de informação.

Requisitos não funcionais

- Permitir interação com o utilizador através de comandos de voz ou texto;
- Adotar uma arquitetura modular, em que cada funcionalidade esteja isolada num módulo independente;
- Permitir a adição de novas funcionalidades ou integrações;
- Permitir a utilização de diferentes LLM de forma intercambiável.

1.2 Contribuições

Deste trabalho, resultaram os seguintes artigos científicos:

- António Goulão, Dinis Dias, Artur Ferreira, Nuno Leite. "*A Personal Digital Assistant Enhanced with Artificial Intelligence Techniques*" [26], INForum – Simpósio em Informática, Setembro 2025, Évora, Portugal
- António Goulão, Dinis Dias, Artur Ferreira, Nuno Leite. "*A Modular Digital Assistant for Windows with Large Language Models*" [25], RECPAD – Portuguese Conference on Pattern Recognition, Outubro 2025, Aveiro, Portugal

O código desenvolvido ao longo deste documento encontra-se disponível no repositório `https://github.com/Agoulao/Personal-Digital-Assistant`. Um contributo relevante desta tese é a disponibilização de uma aplicação de assistente digital pessoal, como nova ferramenta de uso genérico para *Windows desktop*, *open source*.

1.3 Organização do Documento

O restante documento encontra-se estruturado da seguinte forma.

O Capítulo 2 apresenta o estado da arte, abordando a evolução dos assistentes digitais, as principais tecnologias que os suportam e uma análise comparativa das soluções existentes.

O Capítulo 3 descreve as decisões de projeto e os requisitos definidos para o desenvolvimento do assistente digital, abordando a arquitetura proposta, os componentes do sistema, as ferramentas utilizadas e ainda aspetos relacionados com privacidade, segurança e persistência de dados.

O Capítulo 4 apresenta o desenvolvimento do sistema proposto, descrevendo todo o processo de implementação, desde as fases iniciais até à concretização final, incluindo a arquitetura modular, os principais módulos desenvolvidos e a forma como estes interagem entre si e com os modelos de linguagem.

O Capítulo 5 apresenta a avaliação experimental do sistema, com destaque para os testes realizados aos módulos, a comparação de desempenho entre diferentes LLM, a análise da modularidade e extensibilidade da solução e um questionário de usabilidade.

O Capítulo 6 reúne as conclusões do trabalho realizado e apresenta possíveis direções para trabalho futuro, identificando melhorias e novas funcionalidades a explorar.

O Anexo A apresenta o formulário de usabilidade aplicado aos utilizadores, incluindo as instruções, tarefas propostas e questões utilizadas na avaliação do sistema.

Capítulo 2

Estado da arte

Este capítulo tem como objetivo proporcionar uma compreensão abrangente do contexto e dos objetivos desta tese, apresentando uma visão geral do estado atual do conhecimento sobre os assistentes digitais e as tecnologias que os suportam.

Na Secção 2.1, é apresentada a história e evolução dos assistentes digitais, desde os primeiros sistemas de reconhecimento de voz até aos modernos assistentes conversacionais, destacando os principais marcos e avanços tecnológicos que moldaram o seu desenvolvimento.

A Secção 2.2 aborda o funcionamento geral de um assistente digital, detalhando a sua arquitetura e os componentes essenciais que permitem a sua operação.

Na Secção 2.3, são explorados os fundamentos do reconhecimento de fala, incluindo as técnicas de processamento de áudio e os modelos de classificação utilizados.

A Secção 2.4 apresenta o processamento de linguagem natural, descrevendo os seus conceitos fundamentais e as principais fases de análise que permitem a compreensão e geração de linguagem natural por sistemas computacionais.

A Secção 2.5 realiza uma análise comparativa dos principais assistentes virtuais atuais, destacando as suas capacidades, vantagens e limitações.

A Secção 2.6 explora os Modelos de Linguagem de Grande Escala (LLM), apresentando as suas características, arquiteturas e modos de funcionamento, bem como os principais desafios associados à sua utilização e o panorama atual de modelos disponíveis.

2.1 História e evolução dos Assistentes Digitais

Antes de avanços mais sofisticados, ferramentas como Audrey (1952) e IBM Shoebox (1961) [38] já demonstravam capacidades básicas de reconhecimento de fala. No entanto, foi com ELIZA, criado por Joseph Weizenbaum em 1966, que surgiu o primeiro programa de processamento de linguagem natural, marcando o início das interfaces conversacionais e dos *chatbots*. ELIZA simulava conversações utilizando métodos de correspondência de padrões e substituição. Embora não fosse capaz de compreender o significado ou o contexto das interações, foi pioneiro em demonstrar como a linguagem natural poderia mediar o diálogo entre humanos e máquinas. Curiosamente, apesar de o objetivo do programa ser 'demonstrar que a comunicação entre

homem e máquina era superficial' [39], Weizenbaum ficou surpreendido ao descobrir que as pessoas gostavam de interagir com ELIZA e frequentemente atribuíam sentimentos humanos ao programa. Este fenómeno ficou conhecido como "Efeito ELIZA", que descreve a tendência humana de atribuir características humanas a sistemas computacionais aparentemente inteligentes, mesmo quando estes operam de maneira superficial e pré-programada [19].

Nas décadas de 1980 e 1990, surgiram os assistentes digitais pessoais (*Personal Digital Assistant* (PDA)), como o Apple Newton e o PalmPilot. Embora não fossem movidos por Inteligência Artificial (IA), introduziram a ideia de assistência portátil, com ênfase em tarefas organizacionais, como a gestão de notas e agendas. Em paralelo, avanços no reconhecimento de fala, como o ViaVoice da IBM, indicavam o potencial para assistência ativada por voz [55]. A introdução de interfaces gráficas amigáveis e a miniaturização do hardware também contribuíram para o aumento do interesse em assistentes digitais como ferramentas do quotidiano.

Com os anos 2000, as capacidades de Inteligência Artificial (IA) começaram a transformar os assistentes digitais. O Clippy da Microsoft, embora frequentemente criticado [20], foi uma tentativa inicial de oferecer assistência contextual em softwares populares. Mais significativamente, a Siri da Apple, introduzida em 2011, marcou o início da era moderna dos assistentes digitais baseados em IA, ao combinar reconhecimento de fala, conectividade à Internet e Processamento de Linguagem Natural (PLN) [36]. Pouco depois, o Google Now e a Alexa da Amazon introduziram assistência proativa e integração com dispositivos domésticos inteligentes. Estes sistemas utilizavam computação em nuvem para analisar o comportamento dos utilizadores e fornecer recomendações personalizadas, avançando significativamente as capacidades dos assistentes digitais.

Nos últimos anos, assistentes digitais multimodais capazes de integrar entradas de voz, texto e visuais emergiram. Exemplos incluem o Google Assistant e o Microsoft Cortana, que utilizam técnicas avançadas de IA, como aprendizagem profunda e compreensão contextual, para melhorar a precisão e a experiência do utilizador. Além disso, o ChatGPT da OpenAI, introduzido em 2023, demonstrou o potencial de modelos de IA generativa para realizar tarefas conversacionais, redigir textos e até auxiliar na geração de código.

Aplicações específicas para indústrias expandiram o âmbito dos assistentes digitais. Em logística e produção, por exemplo, assistentes digitais são agora utilizados em tarefas que vão desde a gestão de inventários até à manutenção preditiva, demonstrando a sua versatilidade e adaptabilidade [55].

2.2 Funcionamento de um Assistente Digital

Um Assistente Digital (AD) ou Assistente Virtual (AV) é um agente de software capaz de executar uma série de tarefas ou serviços a partir de informações de um utilizador, tais como comandos ou perguntas, através de texto, interface gráfica ou por fala.

O principal objetivo de um assistente digital é simplificar tarefas do quotidiano, permitindo que os utilizadores executem ações como gerir agendas, controlar dispositivos domésticos,

realizar pesquisas ou até fazer compras, através de comandos de voz ou texto. Diferentemente das tecnologias de comando de voz mais antigas, que funcionavam com um conjunto limitado de ordens pré-programadas, os assistentes digitais modernos utilizam PLN para interpretar e responder a uma ampla variedade de instruções [27, 56].

O seu funcionamento baseia-se numa série de etapas, que se encontram representadas na Figura 2.1. De forma simplificada, o software inicialmente encontra-se num estado contínuo de escuta ativa até detetar as palavras-chave. Ao detetar estas palavras-chave, o comando de voz é gravado e enviado para um servidor especializado, onde é processado por sistemas de Reconhecimento de Fala e PLN. O comando é transformado em texto, analisado e, finalmente, é realizada a ação apropriada. Finalmente, o assistente gera uma resposta, converte de texto para fala e apresenta-a ao utilizador.

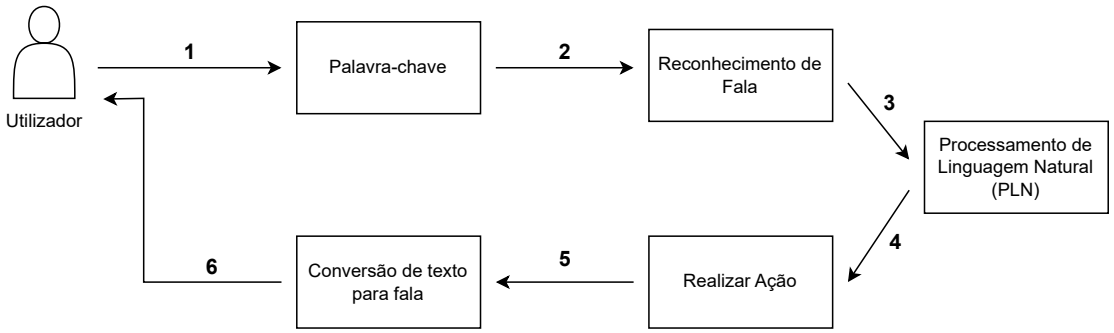

Figura 2.1 Etapas principais do funcionamento de um Assistente Digital baseado em fala 1) Utilizador emite um comando por voz; 2) Sistema deteta a palavra-chave e ativa o Reconhecimento de Fala; 3) Áudio é convertido em texto; 4) Texto é analisado por um sistema de PLN que determina a ação contida no comando; 5) Ação apropriada é determinada e executada; 6) Resposta é gerada, convertida em fala e apresentada ao utilizador

2.3 Reconhecimento de Fala

O reconhecimento de fala em assistentes digitais começa com um processo denominado Reconhecimento Automático de Fala (*Automatic Speech Recognition* (ASR)) [21, 47, 56]. Este processo consiste na utilização de aprendizagem automática para transformar o discurso humano em texto legível. A Figura 2.2 detalha as etapas deste processo.

Em seguida, descrevem-se estas etapas.

1. **Aquisição e Pré-processamento do Áudio**

   O processo de ASR começa com a captação do áudio através do microfone do dispositivo. Nesta primeira etapa, o sinal de fala pode passar por várias transformações para melhorar a sua qualidade e consistência para análise posterior. Isto pode incluir redução de ruídos de fundo, normalização do volume, segmentação em blocos temporais, e potencialmente outras técnicas de processamento de sinal como pré-ênfase. A qualidade deste pré-processamento influencia significativamente as etapas subsequentes [56].

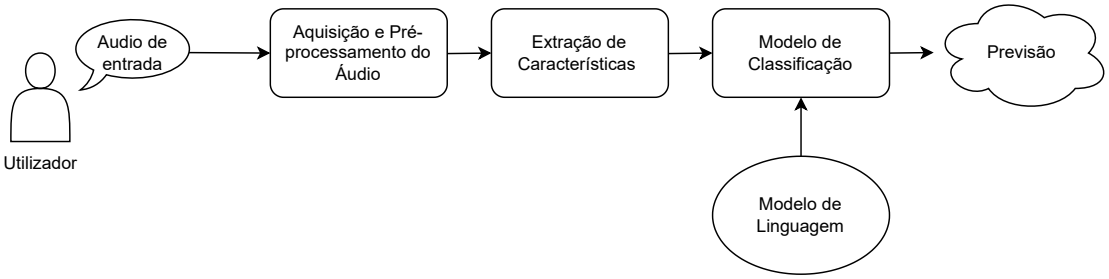

Figura 2.2 Etapas principais de um sistema de Reconhecimento de fala

2. **Extração de Características**

   A extração de características converte os segmentos de áudio pré-processados em representações matemáticas adequadas para análise computacional. Este processo envolve a extração de diferentes propriedades acústicas do sinal, como os coeficientes mel-cepstrais (MFCC) e espetrogramas, que capturam informações sobre frequência, energia e outras características do sinal. O foco desta fase está em transformar o sinal em representações numéricas que destacam as propriedades acústicas relevantes para o reconhecimento de fala.

3. **Modelo de Classificação**

   O modelo de classificação é responsável pela interpretação das características acústicas extraídas e a sua conversão em texto. Dependendo da arquitetura utilizada, este processo pode seguir diferentes abordagens. Sistemas tradicionais como *Hidden Markov Models* (HMM), *Gaussian Mixture Models* (GMM) e *Dynamic Time Warping* (DTW) tipicamente seguem um processo em etapas, primeiro identificando fonemas e depois combinando-os em palavras. Já os sistemas modernos de aprendizagem profunda, como Redes Neuronais Recorrentes (RNN) e Redes Neuronais Convolucionais (CNN), podem realizar um mapeamento mais direto das características acústicas para texto, sem necessariamente passar por uma etapa explícita de identificação de fonemas [21].

4. **Modelo de Linguagem**

   O Modelo de Linguagem (LM) atua como componente complementar que refina e contextualiza as previsões do modelo de classificação. É um sistema de IA treinado em grandes volumes de texto que aprende a prever sequências prováveis de palavras com base no contexto. O LM analisa a probabilidade e coerência das sequências de palavras identificadas, ajudando a resolver ambiguidades e a corrigir erros de reconhecimento. Por exemplo, em casos onde diferentes palavras podem corresponder a padrões acústicos similares, o LM pode ajudar a determinar qual a possibilidade que faz mais sentido no contexto da frase. Nalguns casos, pode também incorporar conhecimento específico do domínio, como vocabulário técnico ou comandos comuns do sistema.

2.4 Processamento de Linguagem Natural

Processamento de Linguagem Natural (PLN) é uma subárea da Inteligência Artificial (IA) que estuda a interação entre humanos e máquinas por meio da linguagem natural. O objetivo das técnicas de PLN é capacitar os sistemas computacionais a entender, interpretar e gerar textos de maneira que pareça natural para os seres humanos [12]. Esta técnica é fundamental para os sistemas realizarem tarefas de acordo com as instruções dos utilizadores, estando presente em múltiplas aplicações quotidianas como assistentes de voz, *chatbots*, categorização de *e-mails*, filtros de SPAM, motores de busca, corretores gramaticais e ferramentas de tradução [48].

PLN envolve técnicas computacionais que permitem às máquinas compreenderem e manipularem a linguagem humana. Este campo integra conceitos de linguística, estatística e aprendizagem automática para processar textos e voz. As técnicas de PLN procuram extrair significado e contexto, permitindo que os sistemas compreendam o significado real de uma frase em diferentes situações.

A importância de PLN no mundo contemporâneo estende-se por diversos setores industriais e aplicações quotidianas. Na área de atendimento ao cliente, *chatbots* utilizam PLN para compreender e responder a consultas [15], enquanto no setor da saúde auxilia na gestão de registos médicos [5]. Assistentes virtuais como Alexa, Siri e Google Assistant utilizam técnicas avançadas de PLN para interpretar comandos e fornecer respostas contextualizadas [11]. Modelos mais sofisticados, como o GPT-3, expandem estas capacidades ao permitir conversas mais naturais e geração de texto em diversos tópicos [10].

PLN é um processo complexo que envolve várias etapas para transformar o texto de entrada em informações estruturadas e compreensíveis para as máquinas [33]. Este processo é organizado em cinco fases, representadas na Figura 2.3. Estas cinco fases de PLN trabalham em conjunto para permitir que os sistemas computacionais compreendam e processem a linguagem natural de forma eficaz.

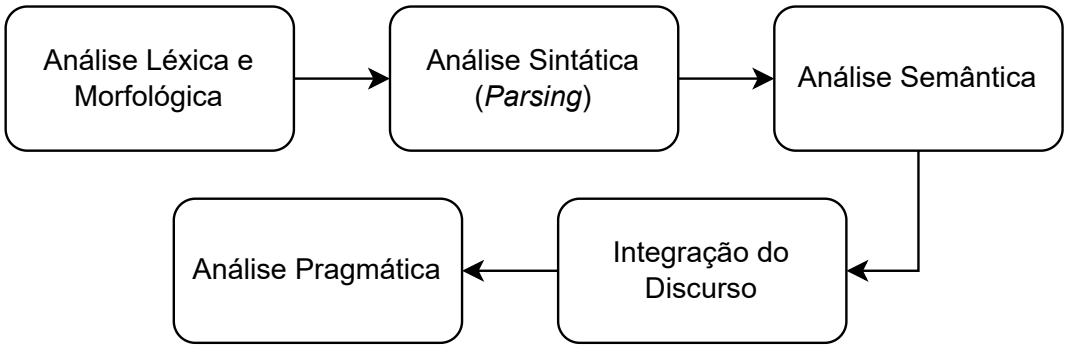

Figura 2.3 Fases principais de PLN

- **Análise Léxica e Morfológica** – Divide o texto em unidades menores (*tokens*) e identifica a estrutura interna das palavras. Inclui tarefas como lematização (reduzir palavras à sua forma base) e remoção de termos irrelevantes (*stopwords*).

- **Análise Sintática** – Examina a estrutura gramatical da frase, atribuindo etiquetas às palavras (substantivo, verbo, artigo, etc.) e verificando as relações sintáticas entre elas, de modo a garantir que a frase está bem formada.
- **Análise Semântica** – Atribui significado ao texto, identificando entidades (pessoas, locais, organizações), desambiguando palavras com múltiplos sentidos e estabelecendo relações de significado entre conceitos.
- **Integração do Discurso** – Relaciona frases consecutivas para formar um discurso coerente, resolvendo referências a elementos já mencionados e mantendo a continuidade temática numa conversa ou texto.
- **Análise Pragmática** – Considera o contexto social, cultural e situacional da comunicação para interpretar intenções implícitas. Permite compreender significados além do literal, como pedidos indiretos ou sugestões subtis.

2.5 Análise comparativa dos assistentes virtuais existentes

De acordo com Pangarkar [45], atualmente, os assistentes virtuais mais utilizados são o Google Assistant, a Siri e a Alexa, cada um desenvolvido para diferentes ecossistemas e dispositivos. A Siri, introduzida pela Apple em 2011, é exclusiva do ecossistema iOS e macOS, permitindo aos utilizadores interagir com aplicações nativas e controlar dispositivos Apple. A Alexa, da Amazon, lançada em 2014, é amplamente utilizada em dispositivos Echo e Fire, sendo otimizada para automação residencial e compras online através da Amazon. O Google Assistant, lançado em 2016, está integrado em dispositivos Android, Google Nest e outros produtos, destacando-se pela sua forte integração com os serviços da Google [27].

A Cortana, lançada em 2013 pela Microsoft, também foi muito relevante no mercado nos últimos anos. No entanto, em 2023, a Microsoft encerrou o suporte à Cortana para concentrar os seus esforços no desenvolvimento e integração de soluções mais avançadas, como o Copilot, o Bing AI e o ChatGPT [37]. Esta mudança evidencia a rápida evolução da tecnologia dos assistentes virtuais e o panorama competitivo em que operam, bem como os desafios que estes sistemas enfrentam para se manterem relevantes face às capacidades crescentes das tecnologias de IA emergentes.

Estudos recentes analisaram o desempenho destes assistentes virtuais. Estas pesquisas realizaram análises quantitativas com base em métricas como taxa de acerto, precisão na compreensão de comandos, tempo de resposta e consistência das respostas em cenários controlados. Em geral, as experiências consistiram em submeter os dispositivos a um conjunto de comandos e questões complexas (por exemplo, reconhecimento de nomes de medicamentos e execução de tarefas contextuais) e calcular a percentagem de respostas corretas ou a redução dos erros na interpretação dos comandos.

No trabalho de Palanica e Fossat [44], foram comparadas as taxas de compreensão de nomes de medicamentos entre 2019 e 2021. O Google Assistant registou os melhores

resultados, com uma precisão entre 84% e 86%, seguido pela Siri, que obteve percentagens entre 75% e 78% e a Alexa, que apresentou o pior desempenho, com taxas entre 64% e 67%. Além disso, o estudo evidenciou um aumento de desempenho na Alexa e na Siri, com melhorias que variaram entre 10% e 24% ao longo desses dois anos, demonstrando a evolução dos algoritmos de reconhecimento.

Adicionalmente, Berdasco et al. [8], (2019) investigaram a experiência do utilizador e concluíram que a naturalidade e a correção das respostas da Alexa e do Google Assistant foram significativamente superiores às da Siri e da Cortana.

Em Alagha e Helbing [3], em 2019, avaliou-se a qualidade das respostas dos assistentes a questões de saúde, especificamente sobre vacinas. Os resultados indicaram que tanto o Google Assistant como a Siri forneceram informações mais precisas e fundamentadas, enquanto a Alexa apresentou limitações na fiabilidade das fontes e na consistência das respostas em contextos de saúde.

Mais recentemente, Jaybhaye et al. [29] (2023) realizaram uma análise comparativa detalhada dos assistentes virtuais, avaliando a experiência do utilizador com base em parâmetros-chave, como personalidade, reconhecimento de fala, compreensão contextual, entre outros. Na análise, o Google Assistant obteve os melhores resultados, enquanto a Alexa e a Siri apresentaram desempenhos inferiores, embora semelhantes entre si.

A análise dos estudos avaliados demonstra que o Google Assistant obteve, de forma geral, os melhores resultados em métricas como reconhecimento de fala, precisão das respostas e experiência do utilizador. No entanto, o desempenho relativo da Siri e da Alexa varia consoante o critério analisado. Enquanto a Siri apresenta melhores resultados em determinados contextos, como a qualidade das respostas em questões de saúde, a Alexa tem melhor desempenho em aspetos como experiência do utilizador e compreensão contextual.

2.6 Modelos de Linguagem em Grande Escala (LLM)

Os LLM são modelos de IA treinados em grandes volumes de texto, concebidos para compreender e gerar linguagem natural de forma contextualizada. O seu desenvolvimento recente foi impulsionado pela introdução da arquitetura *Transformer* [51], pelo aumento significativo da capacidade computacional disponível e pela abundância de dados textuais em larga escala. Ao contrário de abordagens anteriores, como os modelos estatísticos baseados em n-gramas ou as redes neuronais recorrentes que processavam o texto palavra a palavra sequencialmente, os *Transformers* recorrem a mecanismos de atenção capazes de relacionar palavras em qualquer ponto da sequência, superando as limitações anteriores [40].

Na prática, os LLM atuais integram centenas de milhões ou mesmo milhares de milhões de parâmetros, o que lhes permite alcançar desempenhos próximos do humano em diferentes tarefas linguísticas [1]. O treino é geralmente conduzido de forma autossupervisionada, através da previsão da próxima palavra ou do preenchimento de palavras em falta a partir de grandes coleções de texto, que incluem livros, artigos, páginas web ou até código-fonte. Graças a este

processo, um único modelo consegue executar múltiplas tarefas de PLN sem necessidade de treino específico, como tradução automática, resumo de texto, resposta a perguntas, análise de sentimentos ou geração de código.

2.6.1 Arquiteturas e Funcionamento dos LLM

Embora haja diversas variações, a maioria dos LLM modernos assenta na arquitetura *Transformer*. A arquitetura *Transformer*, apresentada por Vaswani et al. em 2017 [51], é composta por duas partes principais: *encoder* e *decoder*, conforme ilustrado na Figura 2.4. O *encoder* (lado esquerdo) recebe o texto de entrada, converte as palavras em vetores numéricos e aplica mecanismos de atenção múltipla (*multi-head attention*) que permitem relacionar cada palavra com todas as outras da frase. O *decoder* (lado direito), por sua vez, utiliza essa informação para gerar a saída passo a passo, recorrendo a um mecanismo de atenção mascarada que garante que cada palavra apenas considera o contexto anterior durante a geração.

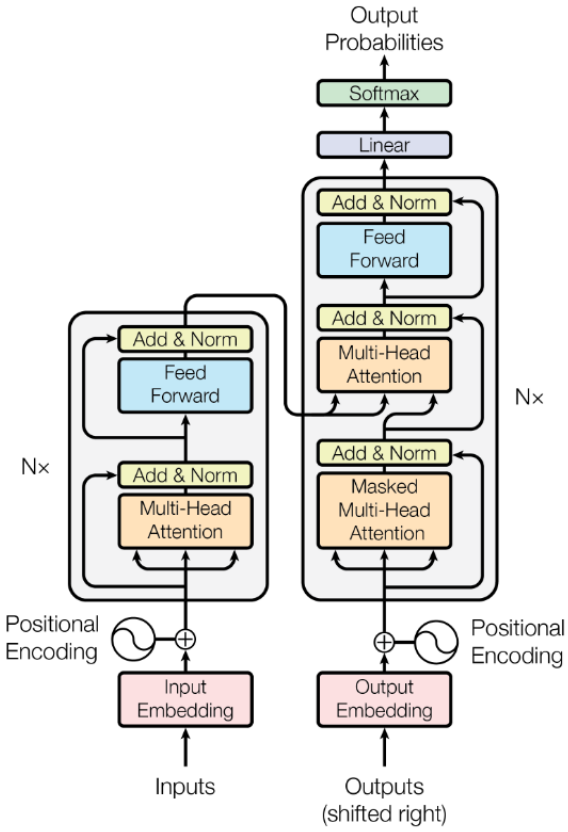

Figura 2.4 Arquitetura *Transformer* [51].

Os mecanismos de atenção funcionam atribuindo diferentes pesos às palavras de acordo com a sua relevância no contexto. Na frase *"O gato subiu à árvore porque estava assustado"*, por exemplo, o modelo aprende que o verbo "estava" se refere ao gato e não à árvore. A atenção múltipla (*multi-head attention*) expande este conceito, permitindo que várias "cabeças" analisem simultaneamente diferentes perspetivas do texto, algumas captam relações gramaticais, outras semânticas ou dependências de longo prazo. A combinação destas visões complementares

dá ao modelo uma compreensão mais rica e completa. Estas características tornaram os *Transformers* a base de praticamente todos os LLM modernos, embora possam assumir diferentes variantes consoante o objetivo de treino.

Entre estas variantes, encontram-se os modelos auto-regressivos, como a família GPT, especializados na geração fluente de texto ao prever o próximo *token* com base no contexto anterior; os modelos bidirecionais, como o BERT, que capturam relações mais profundas entre palavras considerando simultaneamente o contexto anterior e posterior; e as arquiteturas *encoder–decoder*, como o T5, que oferecem flexibilidade em tarefas como tradução ou resumo, uma vez que processam toda a entrada antes de gerar a saída.

Em termos de objetivo de treino, um LLM auto-regressivo procura minimizar o erro na previsão do próximo token em sequências reais de linguagem, enquanto um modelo bidirecional como o BERT aprende a preencher lacunas (*tokens* mascarados) no texto. Ambos os métodos são auto-supervisionados, explorando padrões estatísticos dos dados textuais sem necessidade de rótulos humanos explícitos.

Após o pré-treinamento em larga escala, muitos LLM passam por ajustes adicionais para se tornarem mais úteis e seguros. Entre as técnicas mais comuns encontra-se o *instruction tuning*, onde o modelo é refinado com exemplos de perguntas e respostas ou instruções humanas, e o RLHF (*Reinforcement Learning from Human Feedback*), em que o modelo é ajustado com base em preferências humanas. O objetivo destes métodos é alinhar o comportamento do modelo com critérios como ser útil, honesto e inofensivo, reduzindo respostas inadequadas ou sem sentido que poderiam surgir apenas do pré-treinamento.

Em paralelo, avanços em *prompting* mostraram que grandes modelos conseguem aprender novas tarefas a partir de poucos exemplos (*few-shot learning*) ou até mesmo sem exemplos prévios (*zero-shot reasoning*). Estratégias como o *chain-of-thought* induzem o modelo a explicar passo a passo o seu raciocínio, aumentando a precisão em tarefas que exigem maior capacidade lógica.

2.6.2 Alucinações em LLM

Apesar das suas capacidades, um dos principais desafios dos LLM é o fenómeno das alucinações. Neste contexto, alucinar significa gerar respostas que parecem plausíveis e bem formuladas, mas que são factualmente incorretas ou sem fundamento. O modelo pode, por exemplo, inventar factos históricos, produzir referências inexistentes ou até criar código inválido, transmitindo, contudo, a resposta com grande confiança [28].

Este problema ocorre porque os LLM não possuem uma noção intrínseca de verdade. O treino baseia-se apenas em padrões estatísticos da língua, e não em conhecimento verificado. Assim, quando o modelo enfrenta uma falta de informação ou um contexto ambíguo, tende a preencher com uma continuação linguística plausível, mesmo que seja irreal. A forma de descodificação também influencia: algoritmos mais aleatórios aumentam a probabilidade de erros, enquanto abordagens determinísticas podem reforçar incorreções presentes nos dados de treino.

As consequências destas falhas variam consoante o domínio. Em conversas informais, podem gerar apenas confusão, mas em contextos críticos, como saúde, direito ou finanças [32, 41], podem ter impacto significativo, sendo este um dos principais fatores que limita a adoção de LLM em cenários de maior risco.

Diversas estratégias têm sido exploradas para mitigar o problema, incluindo o uso de fontes externas através de *retrieval augmentation*, que permite ao modelo consultar dados atualizados em vez de depender apenas da memória estática, ou técnicas de alinhamento como o RLHF [34] e a *Constitutional AI* [6], que procuram ensinar o modelo a reconhecer incerteza e, quando apropriado, evitar dar respostas infundadas. Embora nenhuma solução elimine totalmente o problema, estas abordagens têm vindo a reduzir a frequência e a gravidade das alucinações, contribuindo para tornar os LLM mais fiáveis em aplicações práticas.

2.6.3 Panorama Atual de Modelos LLM

Nos últimos anos, surgiram diversos modelos de linguagem em larga escala desenvolvidos por grandes empresas e pela comunidade de código aberto. A lista que se segue não é exaustiva, mas apresenta algumas das famílias mais representativas, destacando as suas características e o papel que desempenham no ecossistema atual.

GPT (OpenAI): Uma das linhas mais conhecidas, inaugurou os LLM generalistas e evoluiu até ao atual GPT-5, com fortes capacidades multimodais e de raciocínio. Estes modelos estão disponíveis apenas via serviços *cloud*, sem acesso público ao modelo e seus parâmetros.

BERT (Google): Introduziu a abordagem bidirecional para tarefas de compreensão textual, tornando-se base de inúmeras variantes (como RoBERTa ou DistilBERT). Apesar de não ser adequado para geração de texto, continua muito usado em aplicações de classificação e análise semântica.

PaLM e Gemini (Google): Representam a aposta da Google em modelos multimodais de larga escala, capazes de lidar com texto, imagens, áudio e vídeo. São disponibilizados sobretudo através da *cloud*.

LLaMA (Meta): Popularizou os LLM de código aberto ao disponibilizar os pesos dos modelos. Deu origem a múltiplas adaptações (como Alpaca ou Vicuna) e é uma referência no ecossistema *open source*.

Claude (Anthropic): Focado em segurança e alinhamento, é disponibilizado apenas como serviço em *cloud*, posicionando-se como alternativa aos modelos da OpenAI e da Google.

Mistral (Mistral AI): Destaca-se pela eficiência em modelos relativamente pequenos, como o Mistral 7B e o Mixtral, ambos *open source* e otimizados para contextos de uso local.

Qwen (Alibaba): Inclui variantes afinadas para conversação, programação e multimodalidade, sendo uma das linhas mais versáteis no ecossistema atual, com versões abertas e acesso via API.

DeepSeek (DeepSeek AI): Exemplo recente de modelos *open source* de grande escala, que competem em desempenho com alternativas proprietárias, reforçando a importância da comunidade aberta.

Estes exemplos ilustram a diversidade de abordagens, desde soluções proprietárias, como GPT ou Claude, até alternativas *open source*, como LLaMA, Mistral ou DeepSeek. Esta variedade traduz-se em diferentes níveis de desempenho, acessibilidade e privacidade, refletindo a amplitude do ecossistema atual. A Tabela 2.1 apresenta alguns exemplos representativos destas famílias, incluindo modelos e número de parâmetros sempre que divulgado publicamente. Esta síntese permite contextualizar rapidamente a diversidade de opções disponíveis.

Tabela 2.1 Famílias representativas de LLM

Família	**Modelo**	**N.º de Parâmetros**
GPT (OpenAI)	GPT-3 [10]	175B
	GPT-4 [42]	Não divulgado
	GPT-5 [43]	Não divulgado
BERT (Google)	BERT Base / Large [17]	110M / 340M
PaLM / Gemini (Google)	PaLM [13]	540B
	Gemini 2.0 Pro / Flash [24]	Não divulgado
	Gemini 2.5 Pro / Flash [14]	Não divulgado
LLaMA (Meta)	LLaMA-2 (7B, 13B, 70B) [50]	7B; 13B; 70B
	LLaMA-3 (8B, 70B) [18]	8B; 70B
Claude (Anthropic)	Claude 1 / 2	Não divulgado
	Claude 4 [4]	Não divulgado
Mistral (Mistral AI)	Mistral 7B [30]	7.3B
	Mixtral 8$\times$7B [31]	46.7B total; 12.9B ativos
Qwen (Alibaba)	Qwen2.5 [54]	0.5B–72B
	Qwen3 [53]	0.6B–235B
	Qwen3-Coder [49]	480B total; 35B ativos
DeepSeek (DeepSeek AI)	DeepSeek 7B / 67B [16]	7B; 67B

No contexto do desenvolvimento de assistentes digitais, esta diversidade de modelos permite uma escolha adaptada a cada cenário, desde serviços *cloud*, que oferecem desempenho máximo e recursos avançados, até soluções locais, que privilegiam maior privacidade e controlo.

Capítulo 3

Análise e Especificação do Sistema

Este capítulo visa apresentar a análise e a especificação do sistema proposto, estabelecendo os seus objetivos, requisitos e visão arquitetónica de alto nível. Pretende-se, assim, criar a ponte entre a revisão do estado da arte (Capítulo 2) e o processo de desenvolvimento detalhado (Capítulo 4).

A Secção 3.1 define os objetivos gerais e específicos do sistema, clarificando as metas a atingir. Na Secção 3.2 são descritos os requisitos funcionais e não funcionais que guiam o desenvolvimento do assistente digital. A Secção 3.3 apresenta uma visão geral da arquitetura, destacando os principais componentes e a forma como estes se interligam para suportar a operação do sistema. Por fim, a Secção 3.4 descreve os princípios arquitetónicos e as principais decisões de projeto, explicitando as alternativas consideradas e os compromissos assumidos.

3.1 Objetivos

O projeto tem como propósito a conceção e implementação de um AD para a plataforma *Windows*, combinando interação por voz e texto com capacidades de processamento de linguagem natural baseadas em Modelos de Linguagem de Grande Escala (LLM). O objetivo geral consiste em desenvolver um sistema modular e extensível capaz de executar tarefas de automação em ambiente de *desktop* para uma experiência de utilização intuitiva e natural.

Para alcançar este objetivo geral, foram definidos os seguintes objetivos específicos:

1. Estruturar uma arquitetura de software modular, com separação clara entre interface, núcleo central, motor de NLP e módulos de funcionalidades.
2. Integrar ASR e TTS, permitindo interação multimodal.
3. Implementar um adaptador de LLM configurável, suportando múltiplos fornecedores de modelos (e.g., Gemini e LLaMA).
4. Desenvolver módulos de funcionalidades para tarefas comuns, incluindo gestão de *e-mails*, eventos de calendário e operações no sistema de ficheiros.

5. Avaliar o desempenho do sistema em termos de latência de resposta e robustez na execução de comandos em cenários reais.

3.2 Requisitos do Sistema

Os requisitos definidos para o sistema estão organizados em duas categorias: funcionais e não funcionais. Os primeiros descrevem as capacidades que o assistente deve oferecer ao utilizador, enquanto os segundos estabelecem restrições de qualidade, desempenho e design da solução.

3.2.1 Requisitos Funcionais

A Tabela 3.1 apresenta os requisitos funcionais e a sua obrigatoriedade.

Tabela 3.1 Requisitos Funcionais do Assistente Digital

ID	Descrição	Obrigatório?
RF01	Processar comandos em linguagem natural, recebidos por texto ou voz.	Sim
RF02	Integrar um LLM para linguagem natural.	Sim
RF03.1	Executar funções relacionadas com gestão de ficheiros (criar, ler, escrever, mover, eliminar).	Sim
RF03.2	Executar funções relacionadas com controlo do sistema operativo (abrir aplicações, interações com rato e teclado).	Opcional
RF03.3	Executar funções relacionadas com gestão de eventos no calendário (integração com Google Calendar).	Sim
RF03.4	Executar funções relacionadas com gestão de *e-mails* (integração com Gmail).	Sim
RF04	Gerar respostas em linguagem natural, tanto em modo conversacional como para confirmação de tarefas.	Sim
RF05	Manter histórico de interações para preservar contexto em comandos subsequentes.	Sim
RF06	Ser executável na plataforma Windows.	Sim
RF07	Suportar múltiplos idiomas.	Opcional

3.2.2 Requisitos Não Funcionais

A Tabela 3.2 apresenta os requisitos não funcionais e a sua obrigatoriedade. Embora os requisitos definidos estabeleçam a base para o desenvolvimento do sistema, importa destacar uma limitação relevante. O requisito **RNF03**, que prevê o armazenamento seguro das credenciais de serviços externos, não foi plenamente cumprido. Na implementação atual, estas credenciais são guardadas em ficheiros de configuração locais, uma solução que simplifica o desenvolvimento mas não assegura a proteção adequada dos dados sensíveis. Esta limitação

é reconhecida e considerada aceitável no contexto deste trabalho, devendo ser endereçada em desenvolvimentos futuros.

Tabela 3.2 Requisitos Não Funcionais do Assistente Digital

ID	Descrição	Obrigatório?
RNF01	A arquitetura deve ser modular, permitindo adicionar, remover ou atualizar módulos de forma independente.	Sim
RNF02	A solução deve suportar de forma flexível a integração de novos fornecedores de LLM.	Sim
RNF03	O sistema deve assegurar a proteção de credenciais e dados sensíveis utilizados em integrações externas.	Sim

3.3 Arquitetura e Componentes

A arquitetura do sistema foi desenhada para ser modular, permitindo que cada componente evolua de forma independente. A Figura 3.1 apresenta a visão de alto nível. A interação inicia-se quando o utilizador envia um comando por voz ou texto através da Interface Gráfica. No caso da voz, o módulo de Reconhecimento de Fala converte o áudio em texto. O núcleo (*backend*) encaminha o comando para o adaptador de LLM, que pode recorrer a diferentes modelos de linguagem para interpretar a intenção do utilizador. A intenção inferida é devolvida ao *backend*, que delega a execução a um dos Módulos de Funcionalidade, responsáveis por tarefas como gestão de ficheiros, calendário ou *e-mail*. Por fim, a resposta é apresentada ao utilizador na Interface Gráfica e, se o TTS estiver ativo, também convertida em fala pelo módulo de Síntese de Fala.

Para dar uma visão mais detalhada, o fluxo de interação quando o utilizador pretende que o assistente realize uma ação encontra-se representado no diagrama de sequência da Figura 3.2. O processo inicia-se com a emissão de um comando pelo utilizador, em texto ou voz (neste caso convertido em texto pelo ASR). O *backend* acrescenta o contexto da sessão e a lista de ações disponíveis antes de enviar o pedido ao adaptador de LLM. O modelo interpreta o comando e devolve a intenção em formato estruturado (JSON), onde cada entrada indica de forma explícita a ação a executar e os respetivos parâmetros (por exemplo, `{"action":"create_file","filename":"notes.txt"}`), este formato normalizado garante previsibilidade, independentemente do LLM utilizado. O *backend* valida o JSON recebido e encaminha a execução para o módulo apropriado. Por fim, o resultado é devolvido à interface e pode ainda ser convertido em fala pelo TTS para fornecer *feedback* auditivo imediato.

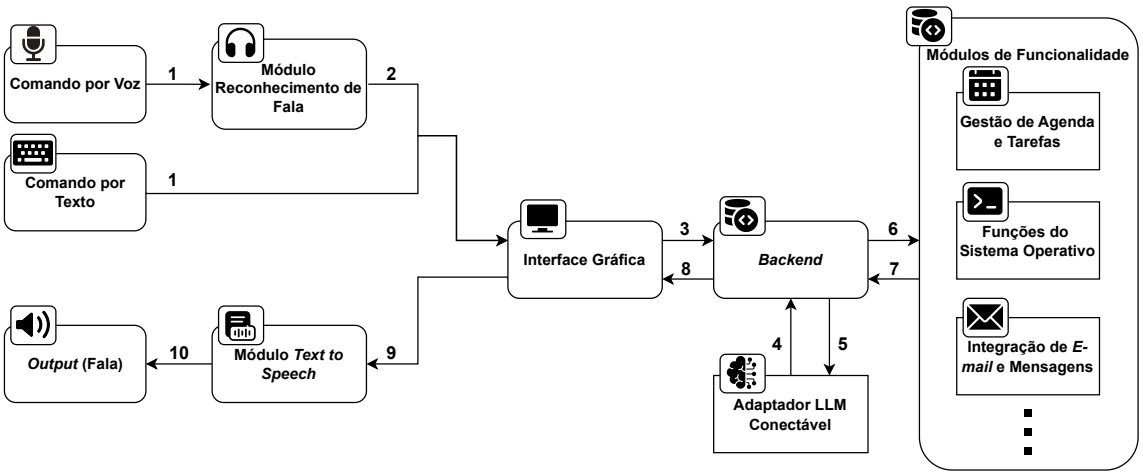

Figura 3.1 Visão de alto nível da arquitetura 1) Utilizador envia um comando por voz ou texto através da Interface Gráfica; 2) No caso de fala, o módulo de Reconhecimento de Fala converte o áudio em texto; 3) *Backend* encaminha o texto para o Adaptador de LLM; 4-5) Adaptador de LLM interpreta o comando e devolve a ação interpretada ao *Backend*; 6) *Backend* delega a execução da ação ao Módulo de Funcionalidade apropriado; 7-8) Resultado é devolvido e apresentado na Interface Gráfica; 9) Opcionalmente, a resposta é sintetizada em fala pelo módulo de TTS

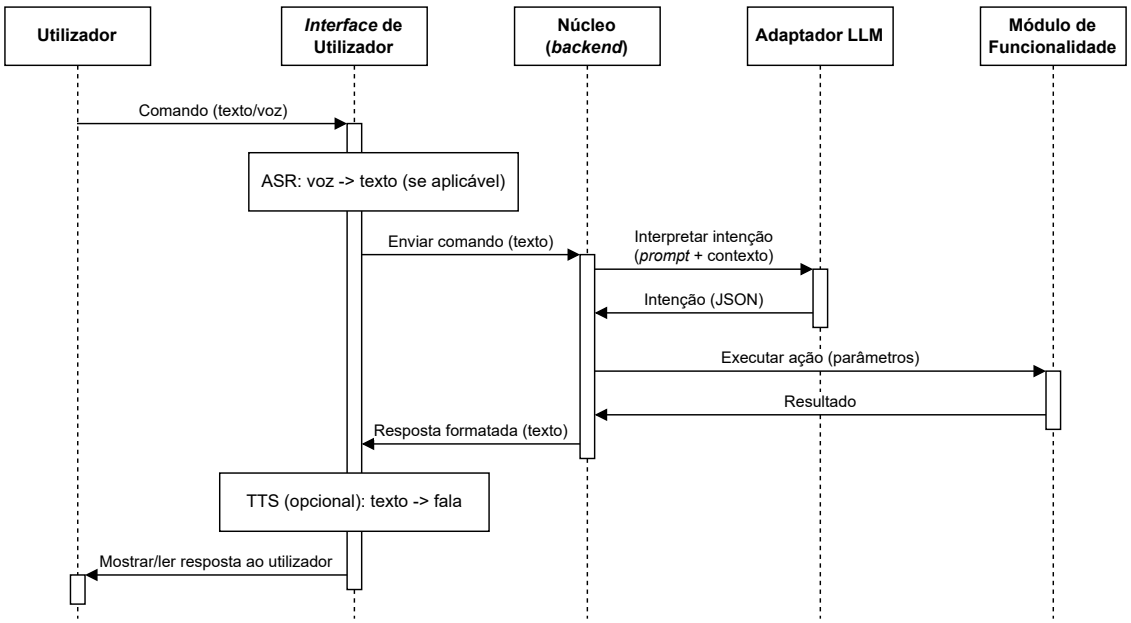

Figura 3.2 Fluxo de interação (diagrama de sequência)

A Figura 3.3 detalha a organização dos ficheiros do projeto e as suas relações. As responsabilidades de cada módulo são as seguintes:

- **UI (**`gui.py`**)** - gere a interação com o utilizador (texto/voz), apresenta respostas e controla ASR/TTS.
- **Backend (**`backend.py`**)** - mantém histórico, compõe pedidos ao LLM (texto + catálogo de

ações), valida o JSON e encaminha a execução.

- **LLM** - fornece um adaptador abstrato (`llm_adapter.py`) com fornecedores intercambiáveis (Gemini, Hugging Face, Awan, Novita), todos com a mesma interface.
- **Módulos de funcionalidades** - herdam de `base_functions.py`, expõem a descrição do módulo e as suas ações. Inclui módulos para execução de funções do sistema operativo, Google Calendar e Gmail.

Esta organização permite trocar o fornecedor de LLM e adicionar novos módulos sem alterações ao núcleo.

3.4 Princípios e Decisões de Projeto

Nesta secção apresentam-se as principais escolhas tecnológicas e conceptuais realizadas no desenvolvimento do protótipo, explicitando as alternativas ponderadas e os compromissos assumidos. O objetivo é clarificar o raciocínio por detrás das decisões do projeto.

3.4.1 Linguagem de Programação e Princípio de Modularidade

A implementação foi realizada em Python 3.10, motivada pela ampla disponibilidade de bibliotecas para PLN e integração com serviços externos. A linguagem é amplamente utilizada em aprendizagem automática e IA, o que facilitou a integração de modelos de linguagem e de bibliotecas de processamento de fala. Além disso, a sintaxe simples contribuiu para um desenvolvimento mais ágil e legível, relevante no contexto de um protótipo experimental como o aqui proposto.

A nível arquitetónico, o sistema foi concebido segundo o princípio da modularidade. Cada componente encontra-se encapsulado num módulo autónomo. Este desenho reduz o acoplamento e facilita a extensão, permitindo a substituição ou adição de funcionalidades sem impactar os restantes componentes.

3.4.2 Interface e Experiência do Utilizador

Foi decidido que o protótipo deveria incluir uma interface gráfica nativa (*GUI*), em vez de depender apenas da linha de comandos ou de uma interface baseada em página web. Esta decisão foi tomada porque uma GUI nativa permite oferecer uma experiência de utilização mais consistente com os assistentes digitais modernos, combinando interação por voz e texto com suporte visual integrado.

Entre as alternativas consideradas estiveram o Tkinter [46] (biblioteca gráfica nativa do Python) e soluções baseadas em tecnologias *web*, como HTML/JavaScript. No entanto, optou-se pelo PyQt5, por oferecer um aspeto mais moderno às interfaces e disponibilizar um conjunto mais alargado de ferramentas para construção e personalização. A integração com o Qt Designer (ferramenta visual disponibilizada pelo PyQt) tornou a prototipagem da interface mais

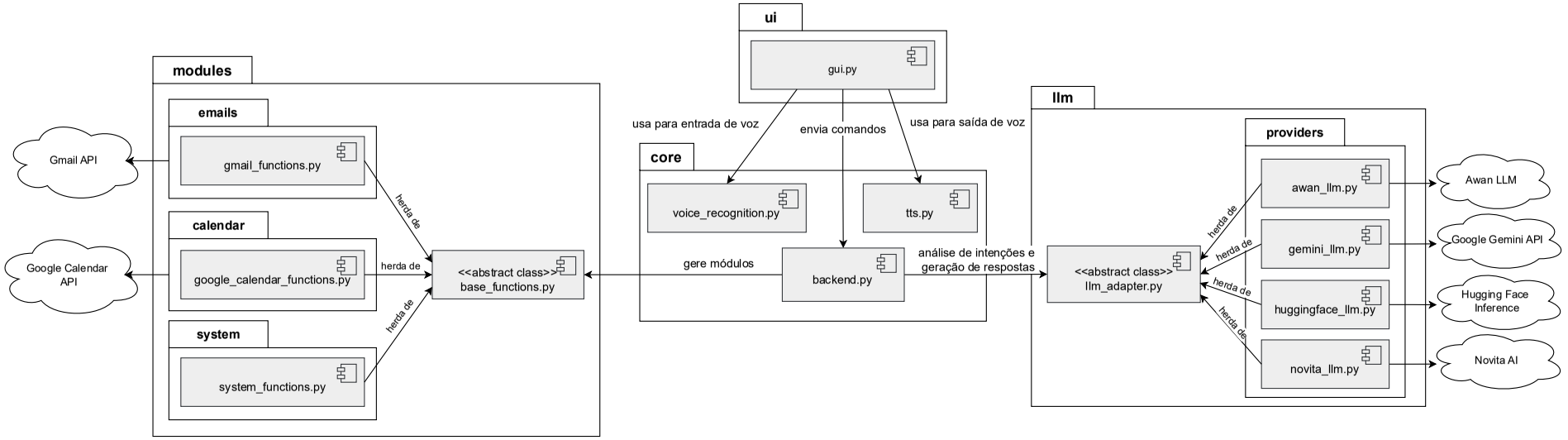

Figura 3.3 Componentes principais do sistema e relações

rápida e prática, simplificando o desenvolvimento. Além disso, tem o benefício de, sendo multiplataforma, não impedir futuras migrações, enquanto também assegura uma experiência consistente no *Windows*, que é o foco do projeto.

3.4.3 Interação por Voz: ASR e TTS

A interação multimodal exigiu a inclusão de módulos de ASR e de TTS. No caso do ASR, optou-se pela biblioteca SpeechRecognition[1], que suporta múltiplos mecanismos de reconhecimento de fala, desde serviços *online* (como o *Google Speech Recognition*) até soluções *offline* (como PocketSphinx ou Vosk). Esta diversidade facilita a integração de diferentes serviços e dispensa a necessidade de treinar modelos locais. Além disso, como o módulo foi concebido de forma encapsulada e intercambiável, permite futuras substituições sem impacto no resto do sistema, em linha com os requisitos RNF01 e RNF05.

Quanto ao TTS, foi escolhida a biblioteca pyttsx3[2], que funciona de forma *offline* e multiplataforma, disponibilizando motores para o Linux, macOS e o Windows. Tanto a gTTS[3] (Google Text-to-Speech) como a pyttsx3 são soluções fiáveis, contudo, a primeira requer ligação à Internet para obter os dados de áudio, enquanto a segunda opera totalmente em modo local. Por essa razão, privilegiou-se a pyttsx3. Tal como no módulo de ASR, a natureza modular da arquitetura permite, no entanto, substituir facilmente este componente por alternativas como a gTTS, caso se considere mais adequado.

Assim, a combinação destas duas bibliotecas representou um compromisso adequado entre rapidez de prototipagem e modularidade futura, cumprindo o Objetivo Específico 2.

3.4.4 Integração de LLM

Uma decisão central no projeto foi a criação de uma camada de abstração para interação com LLM, em vez de acoplar o sistema a um único fornecedor ou implementação. Esta escolha serviu para garantir flexibilidade e longevidade ao protótipo, permitindo a integração de diferentes modelos sem a necessidade de alterar a lógica principal.

A motivação foi dupla. Em primeiro lugar, a evolução dos LLM é extremamente rápida dado que fornecedores introduzem frequentemente novos modelos, atualizam APIs ou descontinuam serviços. Um sistema rigidamente dependente de uma API específica ficaria rapidamente obsoleto ou exigiria grandes alterações de código para se adaptar. Ao definir uma interface abstrata, a substituição de um modelo resume-se a implementar um novo adaptador que respeite o mesmo contrato de comunicação. Em segundo lugar, a abstração protege contra o risco de dependência excessiva de um único fornecedor. Diferentes modelos podem oferecer vantagens em termos de custo, latência, qualidade de resposta ou políticas de privacidade. Assim, em vez de obrigar o utilizador a uma solução proprietária, a arquitetura abre espaço

[1] https://pypi.org/project/SpeechRecognition/

[2] https://pypi.org/project/pyttsx3/

[3] https://pypi.org/project/gTTS/

para escolha; pode-se configurar o assistente para usar um modelo *cloud*, um modelo local executado no próprio PC, ou mesmo alternar entre ambos conforme o contexto de utilização.

Esta decisão também reforça a modularidade do sistema (RNF01 e RNF02), já que o núcleo de interpretação de intenções não precisa de conhecer detalhes específicos de cada API. Para o *backend*, qualquer LLM é apenas um fornecedor de duas operações essenciais: interpretar intenções a partir de texto em linguagem natural e gerar respostas em linguagem natural quando necessário. Essa uniformização simplifica a manutenção, facilita a experimentação e garante que futuras evoluções da tecnologia possam ser integradas sem reestruturações profundas.

Embora, na versão protótipo, todos os modelos testados exijam ligação à Internet, a arquitetura permite que numa evolução futura sejam usados motores totalmente locais, com recurso a bibliotecas como GPT4All[4] e o Ollama[5]. Esta capacidade de intercâmbio cumpre diretamente o requisito RNF05, garantindo que novos fornecedores de LLM podem ser integrados de forma flexível.

Outra decisão relevante consistiu em adotar uma solução própria de interpretação de intenções, em vez de recorrer a mecanismos proprietários como o *function calling* da OpenAI. Embora este último permita definir funções com parâmetros e impor que a resposta seja devolvida em formato JSON estruturado, trata-se de uma funcionalidade específica de determinados modelos e fornecedores. Em vez disto, a solução implementada baseia-se em instruções explícitas no *prompt* de sistema, que obrigam o modelo a devolver um objeto JSON válido representando a ação a executar. Esta abordagem significa que funciona de forma consistente entre diferentes LLM, incluindo aqueles que não suportam nativamente *function calling*, assegurando maior portabilidade e independência face a fornecedores.

Em suma, a combinação da abstração de LLM com um sistema próprio de *parsing* garante que qualquer modelo capaz de seguir instruções pode ser integrado, mantendo-se o contrato de comunicação simples e estável: receber um comando em linguagem natural e devolver intenções estruturadas em JSON. Esta decisão reforça a modularidade (RNF01) e assegura a sustentabilidade da arquitetura perante a evolução acelerada dos modelos de linguagem.

3.4.5 Alternativas Consideradas

Durante o design do sistema, foram ponderadas algumas alternativas em termos de ferramentas já existentes e estratégias de implementação:

Frameworks de Assistentes Digitais - Soluções como o Rasa[6], o LangChain[7] e o Haystack[8] foram avaliadas superficialmente. Todas fornecem mecanismos de modularidade e integração com LLMs, mas assentam em arquiteturas pré-definidas que poderiam limitar a liberdade de experimentação e o controlo granular desejado. Usar uma *framework* estabele-

[4] https://www.nomic.ai/gpt4all
[5] https://ollama.ai/
[6] https://rasa.com/
[7] https://www.langchain.com
[8] https://haystack.deepset.ai/

cida teria significado adaptar o projeto a uma estrutura externa, em vez de desenvolver uma arquitetura própria. Por esse motivo, optou-se por desenvolver um protótipo de raiz, recorrendo a bibliotecas genéricas e a uma arquitetura própria, de forma a assegurar total liberdade na escolha dos LLM e no desenho dos módulos de funcionalidades orientados ao *desktop*.

Ferramentas como o Ollama - O Ollama é um orquestrador que facilita a execução local de modelos como o LLaMA em ambiente *desktop*. Embora seja promissor para cenários totalmente *offline*, optou-se, nesta fase, pela utilização de interfaces REST já consolidadas (Google API[9], Hugging Face Inference[10]), de modo a simplificar a prototipagem e reduzir a complexidade associada à execução local de modelos de grande dimensão. Ainda assim, a arquitetura não exclui a possibilidade de integrar o Ollama ou outro *back-end* local no futuro, bastando implementar um novo adaptador LLM compatível com a interface definida.

3.4.6 Privacidade, Segurança e Persistência de Dados

Várias decisões de projeto tiveram em consideração a proteção de dados do utilizador e a segurança do sistema, ainda que com alguns compromissos dada a natureza experimental do protótipo. Optou-se por não armazenar de forma persistente o histórico de interações (**RF07** limita o histórico à memória volátil). Esta decisão simplificou a implementação e, ao mesmo tempo, evitou que comandos potencialmente sensíveis ficassem registados em disco. Além disso, em APIs externas como a Gmail API e a Google Calendar API, a informação relevante já se encontra armazenada nas respetivas plataformas, pelo que a duplicação local seria redundante. Do ponto de vista da privacidade, isto significa que, ao encerrar o assistente, toda a conversa é descartada, não havendo risco de acesso posterior a interações anteriores. Em contrapartida, o contexto conversacional é perdido entre sessões, um compromisso considerado aceitável, dado que se privilegia a segurança e a simplicidade sobre a conveniência de um histórico permanente.

Por outro lado, reconhece-se uma limitação em termos de armazenamento seguro de credenciais. O requisito **RNF03**, que estipula proteção de credenciais e dados sensíveis, não foi completamente satisfeito na implementação atual. As chaves de API e fichas de autenticação (*tokens* OAuth da Google) são guardadas em ficheiros de configuração locais, nomeadamente um ficheiro `config.py` (para chaves estáticas, como a da API do LLM) e um ficheiro `token.json` gerado pela biblioteca Google Auth (para dados de sessão do Gmail/Google Calendar). Esta abordagem foi adotada por pragmatismo durante o desenvolvimento, sendo que facilitou os testes e a configuração do sistema. No entanto, do ponto de vista de segurança, não é a solução ideal, pois um utilizador mal-intencionado com acesso ao sistema poderia potencialmente extrair esses ficheiros e comprometer contas associadas. Em desenvolvimentos futuros, poder-se-ia melhorar este aspeto integrando um armazenamento cifrado das credenciais (por exemplo, utilizando o *Credential Locker*[11] ou armazenando chaves em variáveis de ambiente e garantindo

[9] https://aistudio.google.com/app/apikey

[10] https://huggingface.co/docs/inference-providers/index

[11] https://learn.microsoft.com/en-us/windows/apps/develop/security/credential-locker

que não são expostas no código). Apesar desta limitação, é importante salientar que as credenciais nunca são transmitidas para fora dos serviços a que dizem respeito, por exemplo, a chave da API do LLM é enviada apenas ao serviço de modelo, e os *tokens* do Gmail/Google Calendar apenas aos servidores da Google através das bibliotecas oficiais, sempre sobre conexões seguras (HTTPS).

Por fim, foram implementados mecanismos de controlo de erros e de segurança nos módulos de funcionalidades. Cada ação executada, seja apagar um ficheiro, enviar um e-mail ou criar um evento no calendário, retorna uma resposta explícita ao utilizador, incluindo mensagens de insucesso caso algo corra mal (por exemplo, `[FAIL] Failed to delete file.`). Este padrão assegura que, mesmo que ocorra uma exceção durante a execução de um comando, o sistema não entra em falha nem permanece num estado inconsistente. Em vez disso, o erro é reportado de forma controlada, cumprindo o requisito de robustez e tolerância a falhas.

Adicionalmente, limitou-se o conjunto de ações disponíveis ao mínimo necessário e esperado: todas as ações são previamente definidas e aprovadas pelo utilizador (através do mapa de ações suportadas carregado no *backend*). Esta abordagem evita que o LLM possa desencadear comportamentos não previstos ou potencialmente perigosos fora do escopo estabelecido. Por exemplo, se o utilizador solicitar a eliminação de um ficheiro, o assistente executa essa ação concreta (já implementada e testada), mas se o LLM devolver uma ação inexistente, esta é ignorada ou tratada como comando inválido, não colocando o sistema em risco.

Capítulo 4

Desenvolvimento

Este capítulo descreve o processo de desenvolvimento do protótipo do Assistente Digital Pessoal, seguindo uma abordagem cronológica que reflete a evolução das principais decisões técnicas e das funcionalidades implementadas.

O objetivo é apresentar de forma detalhada as diferentes etapas do projeto, desde a construção inicial do núcleo de interação por fala, passando pela criação de uma interface gráfica de teste, até à integração de LLM e de módulos de funcionalidades específicos.

A narrativa expõe não apenas os aspetos técnicos da implementação, mas também as motivações e justificações para cada decisão tomada ao longo do percurso. Desta forma, procura-se evidenciar a forma como as opções de design e as adaptações feitas em resposta a desafios técnicos contribuíram para a construção de um sistema modular e extensível.

O capítulo encontra-se organizado da seguinte forma. A Secção 4.1 apresenta o núcleo inicial baseado em ASR e TTS. A Secção 4.2 descreve o desenvolvimento de uma primeira interface gráfica de suporte. A Secção 4.3 analisa a implementação do primeiro módulo de funcionalidades, correspondentes às funções do sistema operativo. A Secção 4.4 aborda a integração de LLM para interpretação de intenções. As Seções 4.5 e 4.6 detalham os módulos de integração com Google Calendar e Gmail. Por fim, a Secção 4.7 apresenta a versão final da interface de utilizador.

4.1 Protótipo Inicial: Núcleo com ASR e TTS

Para que um assistente digital consiga interagir de forma natural com o utilizador, é necessário suportar entrada e saída de fala. Assim, a primeira etapa do desenvolvimento consistiu na implementação de um núcleo funcional mínimo, integrando ASR e TTS. O objetivo desta fase inicial foi validar a viabilidade técnica da captura de comandos falados, a sua transcrição em texto e a devolução de respostas sintetizadas em fala.

O diagrama da Figura 4.1 mostra a interação entre os três componentes: o módulo de Reconhecimento de Fala (`voice_recognition.py`) responsável por captar áudio e transcrevê-lo em texto, o Núcleo Central (`backend.py`) que processa os comandos através de respostas estáticas, e o módulo de Síntese de Fala (`tts.py`), que transforma o texto em fala sintetizada.

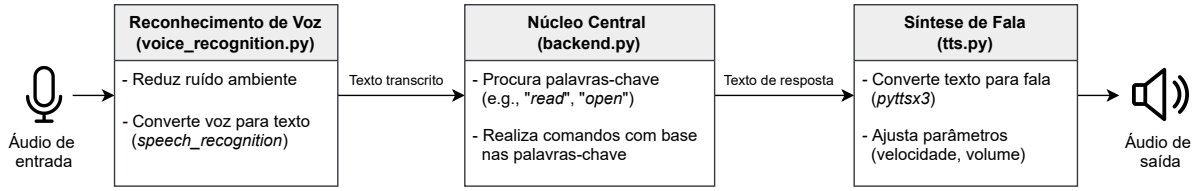

Figura 4.1 Arquitetura inicial do núcleo com ASR, *Backend* e TTS

O ciclo de funcionamento do sistema nesta fase era simples e sequencial. O módulo de Reconhecimento de Fala captava o áudio do microfone e convertia-o em texto, que de seguida era analisado pelo Núcleo Central através de um mecanismo baseado em palavras-chave. Este núcleo verificava a presença de termos previamente associados a ações e, quando encontrados, extraía argumentos básicos, como nomes de ficheiros ou pastas, para completar a instrução. Assim, ao receber o comando *"read example.txt"*, o sistema identificava a palavra *"read"* e procedia à leitura do ficheiro `example.txt`, enquanto a instrução *"create folder reports"* resultava na criação de uma nova pasta chamada `reports`. Por fim, o módulo de Síntese de Fala convertia a resposta em fala, devolvendo mensagens de confirmação como *"File created: reports"* ou o conteúdo do ficheiro solicitado.

Para suportar este funcionamento, recorreu-se à biblioteca `SpeechRecognition`, responsável por captar áudio do microfone, calibrar o ruído ambiente e transcrevê-lo em texto através da API Google Speech Recognition. A síntese de fala foi implementada com a biblioteca `pyttsx3`, que permitia converter texto em fala e ajustar parâmetros como velocidade e volume.

Os primeiros testes abrangeram duas línguas, Inglês e Português, com o intuito de avaliar a flexibilidade do sistema em diferentes configurações linguísticas. Contudo, rapidamente se concluiu que manter o suporte para ambos os idiomas complicava significativamente a fase subsequente de interpretação de comandos em linguagem natural. Por esse motivo, optou-se por restringir a implementação ao Inglês, de modo a reduzir a complexidade e assegurar maior consistência no processamento.

Apesar de rudimentar, este núcleo provou ser suficiente para validar a integração entre captação de voz, transcrição e resposta falada. O sistema não possuía ainda capacidade para interpretar intenções ou executar ações úteis, mas demonstrou a viabilidade da interação por voz, servindo como fundação sobre a qual os módulos seguintes seriam construídos e estabelecendo de forma estável os componentes de ASR e TTS para o resto do desenvolvimento.

4.2 Interface gráfica inicial

Após a validação do núcleo de reconhecimento e síntese de fala, o passo seguinte foi criar uma interface gráfica simples para dispor uma forma prática de interagir com o sistema e observar o seu comportamento. Para tal, foi utilizada a ferramenta `Qt Designer` disponibilizada pelo `PyQT5` para definir uma interface básica com botões e campos de texto, suficiente para validar a integração entre os módulos e apoiar as implementações que seguiriam. Esta interface pode

ser observada na Figura 4.2.

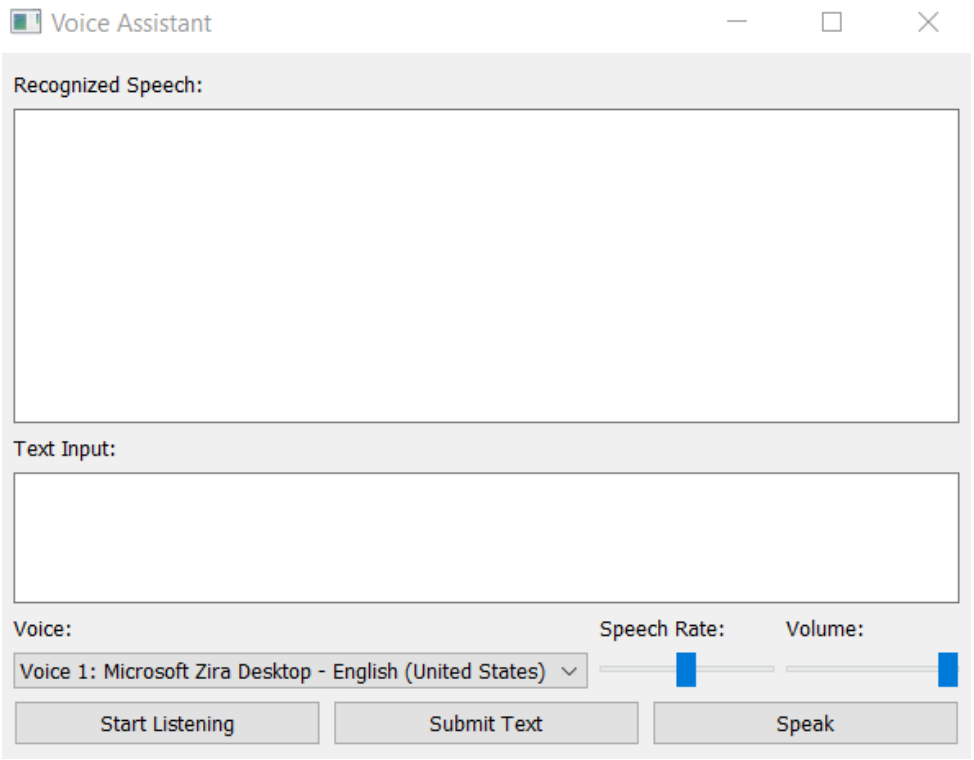

Figura 4.2 Interface gráfica inicial

Esta versão inicial apresentava apenas as funcionalidades essenciais: uma caixa de texto para mostrar as transcrições do reconhecimento de voz, uma área para inserir texto manualmente, controlos para selecionar a voz do sintetizador, ajustar a velocidade e o volume, bem como botões para iniciar a escuta, submeter texto e gerar fala. O seu propósito não era oferecer uma experiência final ao utilizador, mas sim servir de ferramenta prática para validar o fluxo de entrada e saída de dados, observando em tempo real o comportamento dos módulos de ASR, *Backend* e TTS.

A ligação entre esta interface e o núcleo do sistema era feita de forma direta através do `backend.py`, que funcionava como ponto de coordenação. Sempre que o utilizador interagia com a interface, por exemplo, ao premir um botão ou introduzir texto, o evento era encaminhado para o *backend*, que acionava os módulos de reconhecimento ou de síntese de voz conforme necessário. Da mesma forma, as respostas geradas eram devolvidas à interface e apresentadas em campo próprio, o que permitia acompanhar em tempo real o fluxo de entrada e saída.

Embora limitada em termos de estética e funcionalidades, esta versão inicial foi útil porque permitiu visualizar de forma mais clara o sistema em funcionamento, facilitando a realização de testes e a integração dos módulos já desenvolvidos. Dessa forma, serviu como base para avançar para a construção de uma interface final mais completa. Com isto, seguiu-se para a implementação dos primeiros módulos funcionais, começando pelas funcionalidades do sistema.

4.3 Módulo de funcionalidades do sistema

O primeiro módulo funcional a ser implementado foi o módulo do sistema, implementado no ficheiro `system_functions.py`, cujo objetivo era permitir a interação direta com o sistema operativo através de comandos simples. A sua criação marcou um ponto de viragem importante no desenvolvimento, pois correspondeu à separação das ações que até então estavam integradas no `backend.py`. Esta decisão foi motivada pela necessidade de estruturar o sistema de forma modular, permitindo que novas ações ou módulos pudessem ser adicionados ou alterados sem impactar o funcionamento dos restantes componentes.

As ações suportadas encontram-se resumidas na Tabela 4.1. Incluem operações sobre ficheiros e pastas, como criar, ler e eliminar, bem como interações de baixo nível com rato e teclado. Estas últimas foram herdadas das experiências iniciais com o núcleo baseado em palavras-chave e, embora não constituam funcionalidades centrais, foram mantidas por já se encontrarem funcionais.

Tabela 4.1 Ações suportadas pelo módulo de funcionalidades do sistema

Comando	**Descrição**
`create file <nome>`	Cria um novo ficheiro com o nome indicado.
`create folder <nome>`	Cria uma nova pasta.
`write file <ficheiro>: <texto>`	Escreve (sobrescreve) o conteúdo num ficheiro.
`append file <ficheiro>: <texto>`	Acrescenta conteúdo a um ficheiro existente.
`read file <ficheiro>`	Lê e devolve o conteúdo de um ficheiro de texto.
`delete file <ficheiro>`	Elimina o ficheiro especificado.
`delete folder <pasta>`	Elimina a pasta indicada (recursivamente).
`list directory <pasta>`	Lista os conteúdos da pasta indicada.
`rename file <antigo> to <novo>`	Renomeia um ficheiro ou pasta.
`copy file <origem> to <destino>`	Copia um ficheiro para outro local.
`move file <origem> to <destino>`	Move um ficheiro para outro local.
`open application <nome/caminho>`	Abre uma aplicação por nome (se no PATH) ou caminho absoluto.
`move mouse to <x> <y>`	Move o cursor do rato para as coordenadas indicadas.
`click`	Executa um clique do rato (na posição atual).
`click at <x> <y>`	Executa um clique do rato nas coordenadas indicadas.
`type <texto>`	Introduz texto no campo atualmente focado.
`press <tecla>`	Prime uma tecla específica (e.g., `enter`, `esc`, `alt`).

A implementação das ações utilizou bibliotecas padrão do Python e ferramentas auxiliares. Para manipulação de ficheiros e diretórios foi usada a `pathlib`, complementada pela `shutil` para cópias e remoções recursivas. As interações com o rato e teclado foram realizadas com a `pyautogui`, que permite simular movimentos, cliques e introdução de texto. A execução de aplicações foi suportada pelo módulo `os`, que possibilita abrir programas instalados no sistema. Esta combinação de bibliotecas assegurou que o módulo conseguisse executar operações diversificadas, mantendo o código simples e direto.

O fluxo de execução deste módulo é ilustrado na Figura 4.3. O processo inicia-se na Interface Gráfica, que envia o comando textual (ou proveniente de ASR) para o Núcleo Central (*backend.py*). Este analisa o texto e, quando identifica uma palavra-chave definida anteriormente, extrai os argumentos necessários (por exemplo, o nome de um ficheiro ou de uma pasta) e encaminha a execução para o módulo de funcionalidades do sistema. Assim, num comando como *"create file notes.txt"*, a palavra-chave *"create file"* era detetada e a ação resultava na criação do ficheiro `notes.txt`, enquanto *"click"* era interpretado como a instrução para ativar o botão do rato; após a conclusão da tarefa, o resultado era devolvido ao utilizador, com possibilidade de *feedback* por TTS.

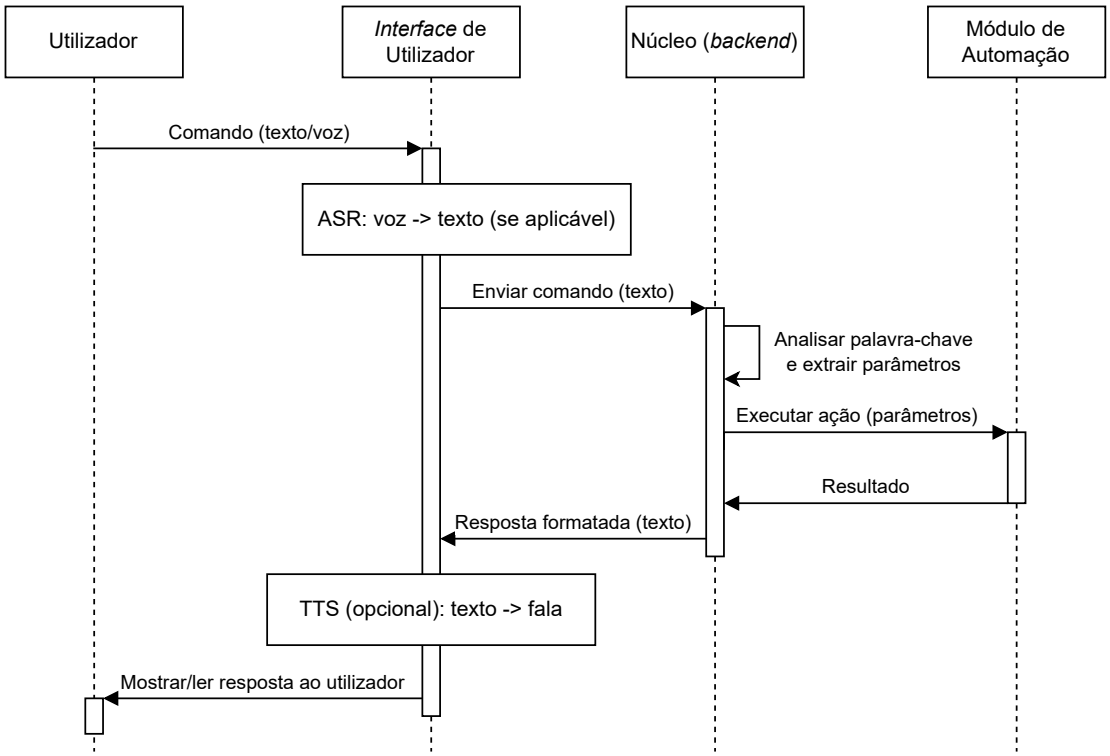

Figura 4.3 Fluxo de execução do módulo de funcionalidades do sistema (`system_functions.py`).

Este módulo permitiu separar a lógica de execução em componentes independentes, de forma a que a adição ou alteração de módulos não afetasse o funcionamento dos restantes. No entanto, rapidamente se percebeu que a abordagem baseada em palavras-chave tinha limitações significativas: o sistema só conseguia reagir a comandos muito específicos e pré-definidos, sem capacidade para lidar com variações linguísticas ou frases mais naturais. Por exemplo, embora conseguisse interpretar *"create file notes.txt"*, não era capaz de compreender instruções semanticamente equivalentes como *"make a new file called notes"*. Esta rigidez tornava o sistema pouco escalável e pouco prático para interações mais próximas da linguagem humana.

Diante desta limitação, tornou-se claro que seria necessário recorrer a LLM capazes de interpretar intenções de forma mais flexível, mapeando comandos expressos em linguagem

natural para ações concretas. Esta necessidade abriu caminho para a integração de LLM na arquitetura.

4.4 Integração de LLM

Até aqui o sistema baseava-se em palavras-chave rígidas para acionar comandos, o que se revelou insuficiente para lidar com a variabilidade da linguagem natural. Para ultrapassar esta limitação, foi necessário integrar Modelos de Linguagem de Grande Escala (LLM), capazes de interpretar intenções de forma mais flexível e próxima da forma como o utilizador realmente se expressa.

Optou-se por recorrer a LLM disponibilizados por API em vez de modelos executados localmente. Esta decisão foi tomada por duas razões principais: por um lado, evita que o utilizador tenha de instalar e configurar modelos no seu computador; por outro, permite alternar entre diferentes fornecedores de forma simples, facilitando a realização de testes. No entanto, esta escolha implica a necessidade de uma ligação estável à Internet e resulta numa maior latência nas respostas face a uma execução local.

Para que esta integração fosse modular, foi desenvolvido um adaptador comum em `llm_adapter.py`, responsável por encapsular a interação com o LLM através da API externa e expor ao `backend` duas operações principais: a interpretação de intenções e a geração de respostas em linguagem natural. Sempre que o utilizador introduz um comando, o `backend` envia o texto recebido para este adaptador, que o reencaminha para o LLM juntamente com um conjunto de regras pré-definidas (o *system parser*) e a lista de ações disponíveis.

Como diferentes LLM podem apresentar formatos de resposta distintos, estabeleceu-se que todas as mensagens de entrada e saída deveriam seguir um padrão consistente em formato JSON, garantindo assim previsibilidade no processamento. Para impor este padrão foi criado o *system parser*, um conjunto de instruções fornecidas ao modelo no início de cada pedido que define de forma explícita como a resposta deve ser estruturada. O *system parser* guia o comportamento do LLM ao impor regras claras, como devolver apenas um array JSON de intenções e utilizar exclusivamente as ações listadas no catálogo fornecido pelo sistema. Estas restrições evitam que o modelo invente comandos ou varie na forma de resposta, assegurando consistência. Um exemplo simplificado de resposta esperada é apresentado na Listagem 4.1.

Listagem 4.1 Exemplo de resposta JSON esperada

```
[
  {
    "action": "create_file",
    "filename": "notes.txt"
  }
]
```

Na prática, contudo, os modelos podem devolver saídas com texto extra ou blocos em `markdown`, como ilustrado na Listagem 4.2.

Listagem 4.2 Resposta JSON com texto extra e bloco markdown

```
Sure! Here's what I found:

```json
[
  {
    "action": "create_file",
    "filename": "notes.txt"
  }
]
```

Para lidar com estes casos, o adaptador implementa um mecanismo de extração e validação que identifica o primeiro e último delimitador JSON válido, remove texto adicional e tenta novamente interpretar o conteúdo. Este processo garante que, mesmo com pequenas variações na resposta, o resultado final entregue ao *backend* é sempre um JSON válido e coerente com o formato esperado. Caso a mensagem do utilizador seja apenas conversacional (por exemplo, uma pergunta genérica) ou um comando que não corresponda a qualquer ação disponível, cabe ao próprio modelo responder em modo conversacional ou pedir uma clarificação em vez de devolver uma ação inválida.

Para tornar possível esta validação, os módulos de funcionalidades foram estendidos de modo a descrever explicitamente as ações que suportam. Cada módulo implementa dois métodos adicionais: `get_description()`, que fornece uma breve descrição das suas capacidades, e `get_supported_actions()`, que devolve um dicionário com os nomes das ações, a descrição de cada uma e um exemplo JSON com os parâmetros esperados. Por exemplo, no módulo de funcionalidades do sistema, o método `get_supported_actions()` inclui entradas como na Listagem 4.3.

Listagem 4.3 Exemplo de catálogo de ações do módulo de sistema

```
"create_folder": {
    "method_name": "create_folder",
    "description": "Creates a new folder at the specified path.",
    "example_json": "{"action":"create_folder","folder":"DIRECTORY"}"
},
"create_file": {
    "method_name": "create_file",
    "description": "Creates a new empty file at the specified path.",
    "example_json": "{"action":"create_file","filename":"DIRECTORY/FILENAME"}"
}
```

Este catálogo de ações é compilado pelo `backend` no arranque e enviado ao LLM em cada pedido, funcionando como guia para mapear corretamente as intenções para métodos concretos dos módulos.

O fluxo resultante encontra-se representado na Figura 4.4. O processo mostra, de forma resumida, como um comando introduzido pelo utilizador é interpretado pelo LLM, traduzido em intenções e, posteriormente, executado pelos módulos de funcionalidades, com o resultado final devolvido à interface.
```

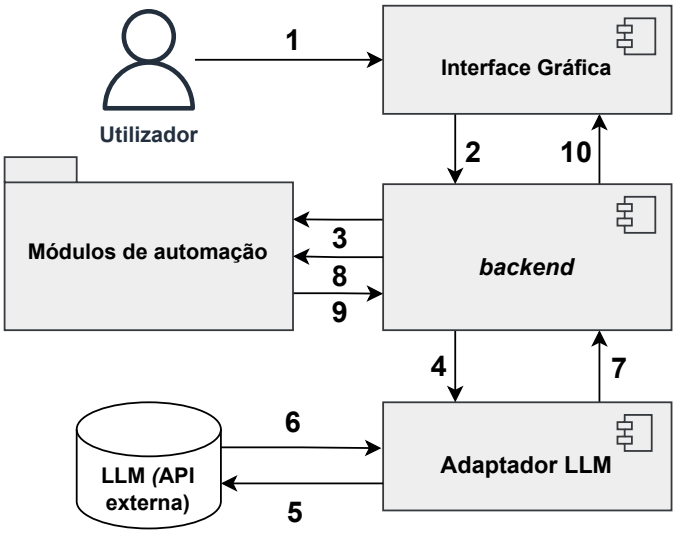

Figura 4.4 Fluxo de interpretação de intenções e execução de ações via LLM 1) Utilizador envia comando; 2) Interface transmite ao *backend*; 3) *Backend* recolhe ações dos módulos; 4) Envia texto + catálogo de ações ao adaptador LLM; 5–6) Pedido e resposta da API externa; 7) Validação do JSON pelo adaptador; 8–9) Execução da ação nos módulos; 10) Resultado devolvido à interface e apresentado ao utilizador

Numa primeira fase, a interação com o LLM era feita diretamente dentro do `llm_adapter`, que acumulava a lógica de autenticação, pedido HTTP e extração da resposta. Mais tarde, esta lógica foi separada em classes independentes de fornecedores, cada uma responsável por lidar com os detalhes específicos de um serviço. Este desenho tornou a arquitetura mais flexível e fácil de manter, já que o `backend` apenas comunica com uma interface comum, enquanto cada *provider* trata da autenticação, formatação do pedido e extração da resposta no seu formato nativo, devolvendo sempre intenções num formato uniforme.

O sistema foi inicialmente testado com o modelo *Meta-Llama-3.1-70B-Instruct*, através da API da Awan LLM [35]. Mais tarde, foram adicionados também os modelos Gemini, via API oficial da Google [22], e a Novita [2], que disponibiliza diferentes modelos compatíveis com o formato OpenAI. A escolha do fornecedor passou assim a ser apenas uma questão de configuração, permitindo experimentar diferentes modelos sem alterar o resto do sistema.

Para suportar esta diversidade, a lógica de comunicação foi separada em ficheiros de fornecedores, cada um responsável por lidar com os detalhes específicos de um serviço, como autenticação, formatação do pedido e extração da resposta. Todos os fornecedores implementam a mesma interface definida no `llm_adapter.py`, expondo apenas dois métodos principais: `parse_intents()`, usado para interpretar comandos de funcionalidades e devolver intenções em formato estruturado, e `generate_response()`, destinado a produzir respostas em linguagem natural em interações conversacionais. Assim, o `backend` não precisa de conhecer as particularidades de cada fornecedor, mantendo sempre a mesma forma de comunicação. Esta arquitetura tornou possível alternar entre modelos diferentes apenas com uma alteração

de configuração, mantendo a integração modular e transparente para o resto do sistema.

Com esta infraestrutura estabelecida, o sistema deixou de depender de regras rígidas baseadas em palavras-chave e passou a interpretar instruções de forma mais flexível e próxima da linguagem natural. Esta mudança criou as bases necessárias para integrar módulos mais avançados, como o de gestão de calendário ou de *e-mail*, que exigem interpretação de intenções mais complexas e tratamento de dados estruturados.

4.5 Gestão de agenda e tarefas com o *Google Calendar*

A integração com o *Google Calendar* constituiu o primeiro módulo de funcionalidades dependente de uma aplicação externa, baseado na API oficial do *Google Calendar* [23]. Para além de gerir eventos de agenda, este módulo pode também apoiar a organização de tarefas pessoais, como lembretes ou compromissos simples. Este caso de uso foi escolhido por representar um cenário onde a interpretação de linguagem natural é indispensável; a criação, listagem e eliminação de eventos exige compreender expressões temporais relativas como "amanhã às 15h", "na próxima semana" ou "reunião de sexta-feira", que não podem ser tratadas apenas com correspondência de palavras-chave.

A implementação foi realizada no ficheiro `calendar_functions.py`, que segue a mesma estrutura modular introduzida anteriormente. Tal como os restantes módulos, define os métodos `get_description()` e `get_supported_actions()`, herdados de `base_functions.py`, permitindo ao `backend` compilar a lista de ações disponíveis e transmiti-la ao LLM em cada pedido.

As principais ações disponibilizadas encontram-se resumidas na Tabela 4.2, que apresenta os parâmetros obrigatórios e opcionais de cada uma. Cada uma destas ações é traduzida diretamente pelo módulo para o método correspondente da API do *Google Calendar*. Sempre que o LLM devolve uma intenção, os parâmetros recebidos, já normalizados em formato ISO-8601 ou texto simples, são convertidos para o formato esperado pela API. O resultado é depois transformado em mensagens padronizadas: em caso de sucesso, são devolvidos resumos legíveis (por exemplo, *"Event 'Dentist Apointment' created successfully. Link: ..."*); em caso de erro, o sistema responde sempre com mensagens consistentes iniciadas em "`[FAIL]`", assegurando clareza e previsibilidade na interação.

As ações do módulo incluem regras adicionais relevantes. Na criação de eventos, se o parâmetro `start_time` for apenas uma data no formato `YYYY-MM-DD`, o evento é tratado como de dia inteiro; se incluir data e hora mas faltar o `end_time`, assume-se uma duração padrão de uma hora. O parâmetro `time_period` aceita diferentes formatos, podendo indicar uma data única, uma data/hora completa ou uma duração. No caso da eliminação de eventos, a procura é feita por correspondência parcial do campo `summary`, de forma insensível a maiúsculas/minúsculas; se não for fornecido `time_period`, é removida apenas a primeira ocorrência futura encontrada.

A comunicação com o serviço é realizada através da API REST do *Google Calendar*, com autenticação via OAuth2.0. Para configurar o acesso, é necessário criar um projeto no *Google Cloud Console*, ativar a API do *Google Calendar* e gerar credenciais do tipo "*OAuth*

Tabela 4.2 Ações suportadas pelo módulo *Google Calendar*

Ação	Descrição	Parâmetros Obrigatórios	Parâmetros Opcionais
`create_event`	Cria um novo evento no calendário.	`summary` (nome do evento), `start_time` (ISO 8601)	`end_time` (ISO 8601), `description`
`list_events`	Lista eventos num período especificado.	`time_period`	–
`delete_event`	Elimina um evento pelo seu `summary` (nome/título).	`summary` (nome do evento)	`time_period` (data ou duração, ISO 8601)

client ID" (aplicação de *desktop*), cujo ficheiro `client_secret.json` contém o `client_id` e `client_secret`. Durante este processo, são também definidos os *scopes* do OAuth2.0, que determinam o nível de acesso concedido. Neste caso, foi utilizado o `.../auth/calendar.events`, que permite criar, listar, alterar e eliminar eventos, alinhando-se diretamente com as funcionalidades implementadas e evitando permissões desnecessárias. Na primeira execução, o utilizador autoriza a aplicação através do fluxo de consentimento da Google, após o qual é gerado um `token.json` com os *refresh/access tokens* usados em chamadas subsequentes. Embora este processo inicial seja relativamente demorado, trata-se de um requisito imposto pela Google para garantir segurança e controlo de acesso. A alternativa seria disponibilizar a todos os utilizadores o mesmo projeto no Google Cloud, o que centralizaria quotas e responsabilidades numa única conta, algo desaconselhado e pouco escalável. Importa ainda referir que este procedimento é realizado apenas uma vez; depois de obtido o `token.json`, as credenciais são reutilizadas automaticamente, não havendo necessidade de repetir o procedimento. O mesmo `client_secret.json` pode, além disso, ser partilhado entre módulos que dependam de serviços Google, como o de *e-mail*, simplificando a integração.

Um dos principais desafios na implementação foi a resolução de expressões temporais relativas. Por definição, os LLM não têm consciência da data ou hora atual no momento da execução, a não ser que essa informação lhes seja explicitamente fornecida no contexto. Isto significava que, durante os primeiros testes, conduzidos com o modelo *Meta-Llama-3.1-70B-Instruct* via *Awan LLM*, tornou-se evidente que expressões como "amanhã" ou "na próxima semana" não podiam ser interpretadas corretamente sem esse suporte adicional. Para resolver esta limitação, o `backend` passou a enviar sempre um bloco de contexto temporal (`CURRENT_DATE_CONTEXT`), que inclui a data e hora atuais, a semana corrente e o mês corrente. Com base neste contexto, o *system parser* instrui o modelo a devolver sempre datas no formato ISO-8601. Assim, uma instrução como "marcar reunião amanhã às 10h" é convertida para ‘‘`start_time`’’:‘‘`2025-10-1T10:00:00`’’ (assumindo que a data atual é 2025-09-30).

Apesar disso, permanecem casos em que o contexto não é suficiente, como quando o utilizador refere apenas "sexta-feira", que pode corresponder à semana corrente ou à seguinte.

Para estes cenários, o sistema adota uma estratégia em várias camadas. Primeiro, o *system parser* instrui o LLM a normalizar dias da semana para a sua próxima ocorrência futura, com base no `CURRENT_DATE_CONTEXT`. Assim, se o pedido for feito numa terça-feira, a expressão "sexta-feira" é interpretada como a sexta mais próxima; se já for sexta-feira, assume-se a data do próprio dia. Em seguida, se a data for identificada sem hora específica, o módulo cria um evento de dia inteiro, enquanto que, no caso de incluir hora mas faltar `end_time`, é assumida uma duração padrão de uma hora. Finalmente, quando o LLM não consegue resolver a ambiguidade ou deteta um erro (como tentar marcar um evento em "31 de setembro"), devolve ‘‘`action`’’:‘‘`none`’’ e responde em linguagem natural, pedindo ao utilizador que clarifique ou corrija a instrução.

Com esta integração, o sistema passou a gerir eventos e tarefas pessoais de forma intuitiva, traduzindo instruções em linguagem natural diretamente em operações no calendário. O uso de contexto temporal e a normalização para ISO-8601 permitiram lidar com expressões como "amanhã" ou "próxima semana" de maneira fiável, sem exigir que o utilizador fornecesse datas exatas, abrindo caminho para integrações futuras, como a gestão de *e-mails*.

4.6 Gestão de *e-mails* com o *Gmail API*

A integração com o *Gmail* constituiu o segundo módulo de funcionalidades dependente de uma aplicação externa. O correio eletrónico representa uma funcionalidade essencial no quotidiano e, tal como no caso da agenda, a sua gestão beneficia de comandos em linguagem natural, mais intuitivos do que sequências rígidas de palavras-chave. Este módulo foi desenvolvido para suportar operações comuns como listar, ler, enviar, marcar como lidos e eliminar *e-mails*, permitindo ao assistente atuar como um cliente de *e-mail* simplificado, mas funcional.

A comunicação com o serviço é realizada através da API REST do *Gmail*, reutilizando o mesmo processo de autenticação OAuth 2.0 configurado no *Google Cloud Console*. Como o projeto e o ficheiro `client_secret.json` já tinham sido criados para o módulo do *Google Calendar*, não foi necessária nenhuma configuração adicional além de ativar a API do Gmail no mesmo projeto e definir os *scopes* adequados. Neste caso, optou-se pelo `https://mail.google.com/`, uma vez que é o único que concede acesso total à caixa de correio, permitindo listar, ler, enviar, marcar como lidas e eliminar mensagens. Outros *scopes* mais restritivos (`readonly`, `compose`, `modify`) não permitem, por exemplo, a eliminação de mensagens. Embora este nível de permissão seja mais abrangente, revelou-se indispensável para suportar todas as funcionalidades previstas no módulo.

Esta escolha, contudo, levanta questões de segurança, já que o acesso total ao correio expõe informações sensíveis em caso de utilização indevida ou comprometimento das credenciais. Para mitigar este risco, a Google impõe a autenticação explícita do utilizador no primeiro acesso e o armazenamento seguro de um `token.json`, que deve ser protegido contra partilha ou fuga de informação. Isto reforça a importância de preservar cuidadosamente os ficheiros `client_secret.json` e `token.json` contra qualquer divulgação indevida.

A implementação encontra-se no ficheiro `gmail_functions.py`, que segue a estrutura modular introduzida anteriormente, expondo as ações através dos métodos `get_description()` e `get_supported_actions()`. A comunicação é efetuada com a biblioteca oficial `googleapiclient.discovery` para construir e enviar pedidos à API, enquanto as respostas são transformadas em mensagens padronizadas: em caso de sucesso, devolvem-se resumos legíveis (por exemplo, *"Gmail: Email sent successfully to ..."*); em caso de erro, a saída é sempre iniciada por `[FAIL]`.

As principais ações disponibilizadas pelo módulo encontram-se resumidas na Tabela 4.3. A ação `list_emails` converte os filtros recebidos em instruções compatíveis com a pesquisa avançada do Gmail (por exemplo, `from:utilizador@dominio.com`, `is:unread`, `after:YYYY/MM/DD`). O resultado inclui os campos principais de cada mensagem, como o remetente, o assunto, a data e um excerto do corpo. As ações `mark_email_as_read` e `delete_email` podem operar diretamente sobre uma lista de identificadores ou, na ausência destes, aplicar os critérios de pesquisa para determinar as mensagens relevantes. A ação `read_email` permite obter o conteúdo completo de uma mensagem individual, enquanto `send_email` constrói uma mensagem `MIME`, codificada em `base64`, que é enviada através do método `users.messages.send`.

Tabela 4.3 Ações suportadas pelo módulo *Gmail*

Ação	**Descrição**	**Parâmetros Obrigatórios**	**Parâmetros Opcionais**
`list_emails`	Lista *e-mails* de uma label (ex.: INBOX, UNREAD, SENT), com filtros opcionais de remetente, período e estado de leitura.	–	`label` (default=INBOX), `max_results` (default=5), `sender`, `date_period`, `all_results` (bool), `is_unread` (bool)
`send_email`	Envia um *e-mail* para um destinatário.	`to`, `subject`, `body`	–
`read_email`	Lê o conteúdo de um *e-mail* específico pelo ID.	`email_id`	–
`mark_email_as_read`	Marca *e-mails* como lidos, por IDs ou por critérios de pesquisa.	Pelo menos um: `email_ids` **ou** (`sender`, `date_period`, `is_unread`)	`label` (default=INBOX)
`delete_email`	Apaga *e-mails*, por IDs ou por critérios de pesquisa.	Pelo menos um: `email_ids` **ou** (`sender`, `date_period`, `is_unread`)	`label` (default=INBOX)

Para além destas operações base, o módulo apresenta algumas particularidades importantes de funcionamento. O parâmetro `date_period` pode indicar tanto uma data única (`YYYY-MM-DD`) como um intervalo (`YYYY-MM-DD/YYYY-MM-DD`), sendo que o limite superior é tratado como exclusivo, acrescentando-se internamente um dia. O campo `label` deve ser sempre

fornecido em maiúsculas, como `INBOX`, `UNREAD` ou `SENT`. No caso da listagem de mensagens, o parâmetro `all_results=true` permite expandir os resultados até 500 entradas, enquanto o valor por omissão devolve apenas cinco. A leitura de mensagens (`read_email`) limita-se a extrair o corpo em `text/plain`, o que pode levar a que *e-mails* apenas em HTML surjam sem conteúdo. Finalmente, a ação `mark_email_as_read` atua simplesmente removendo a label `UNREAD` das mensagens encontradas, e `delete_email` realiza uma eliminação definitiva, sem mover os itens para a pasta de lixo.

Com a integração do módulo de *Gmail*, ficaram concluídos os módulos de funcionalidades previstos. Graças à arquitetura modular adotada, caso se pretendam adicionar novas funcionalidades, criar um novo módulo torna-se um processo relativamente simples. Basta definir uma nova classe a partir de `base_functions.py`, implementar os métodos obrigatórios e expor as ações suportadas. Desta forma, o sistema mantém-se flexível e extensível, preparado para acomodar futuras integrações com outros serviços ou aplicações. A partir deste ponto, a principal área que permanecia em aberto era a interface de utilizador, cuja melhoria se tornou o passo seguinte no desenvolvimento.

4.7 Interface gráfica de utilizador

A interface gráfica inicial encontrava-se limitada em termos de estética e usabilidade, oferecendo apenas uma interação funcional mas pouco apelativa. Nesta fase, tornou-se necessário melhorar a sua apresentação de modo a oferecer uma experiência de utilizador mais agradável e intuitiva. O resultado foi a versão final ilustrada na Figura 4.5, que combina um painel estilo *chat* com suporte a texto e voz, garantindo uma interação mais fluida e próxima da de um assistente moderno.

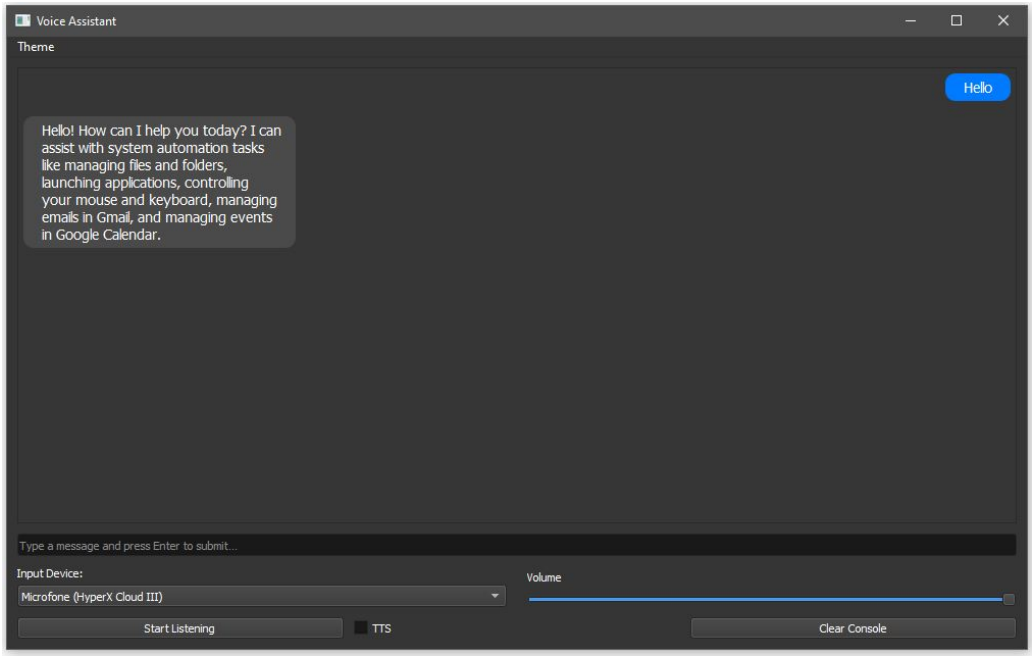

Figura 4.5 Interface gráfica final do assistente (modo escuro)

A implementação da interface encontra-se no ficheiro `gui.py`, estruturada em três áreas principais que incluem a caixa de mensagens, a caixa de entrada de texto e um conjunto de controlos auxiliares. A caixa de mensagens mostra as mensagens trocadas, com balões estilizados de forma distinta para o utilizador (à direita, em azul) e para o assistente (à esquerda, em cinzento). A caixa de entrada permite introduzir comandos manualmente e é complementada por botões que permitem iniciar ou parar a escuta por voz, ativar ou desativar o TTS automático e limpar o histórico da sessão. Já os controlos adicionais possibilitam escolher o dispositivo de entrada de áudio, ajustar o volume da síntese de fala e alternar entre tema claro e escuro.

A interação com o núcleo do sistema é mediada pela classe `Backend`, responsável por coordenar os módulos de funcionalidades e o adaptador LLM. Sempre que o utilizador submete uma instrução, a interface cria uma bolha de mensagem correspondente e inicia a animação de pontos suspensos ($. \rightarrow .. \rightarrow ...$), de forma a fornecer *feedback* visual e indicar que o pedido está a ser processado. Durante este período, a caixa de entrada de texto e o botão *Start Listening* ficam temporariamente desativados, impedindo o envio de novas mensagens até que a resposta esteja pronta. Quando o resultado é recebido, a animação é substituída pela resposta do assistente, que pode ainda ser sintetizada em fala, caso o modo TTS esteja ativo. Este mecanismo, ilustrado na Figura 4.6, melhora a experiência do utilizador ao transmitir de forma clara que o sistema continua ativo e em processamento.

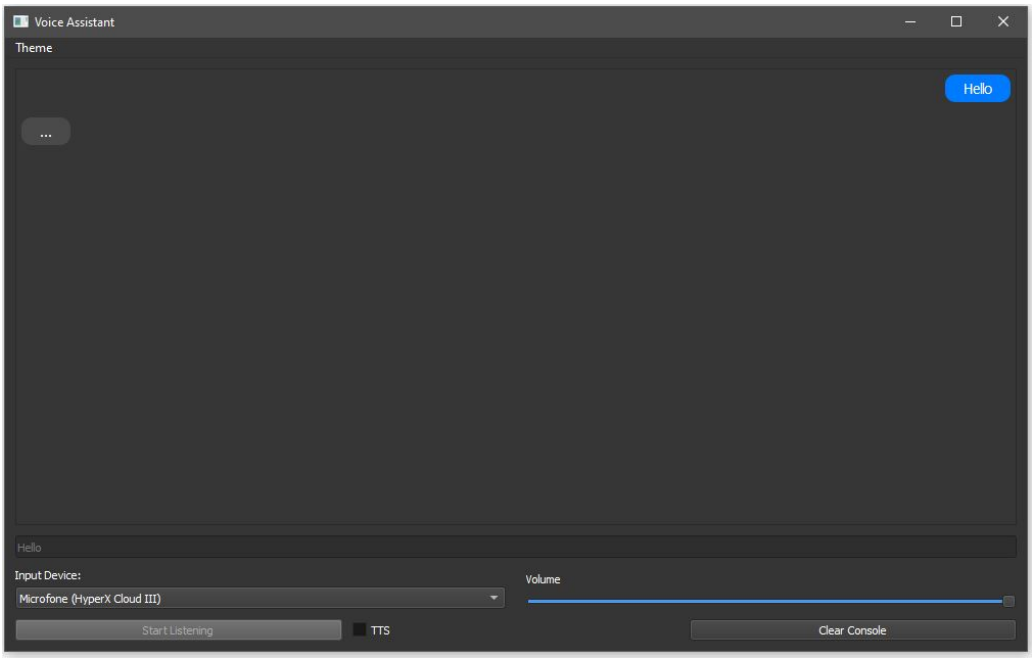

Figura 4.6 Interface gráfica durante o processamento de uma mensagem (modo escuro)

Entre as funcionalidades adicionais destacam-se a possibilidade de limpar rapidamente o histórico de mensagens (`Clear Console`), que também reinicia a memória de conversação do `backend`, e a opção de alternar entre tema escuro e claro, útil para ajustar a legibilidade ao contexto de uso, como ilustrado na Figura 4.7. O design segue um estilo minimalista, com bolhas arredondadas e cores contrastantes.

Com esta interface, o sistema passou a oferecer uma experiência de utilização satisfatória,

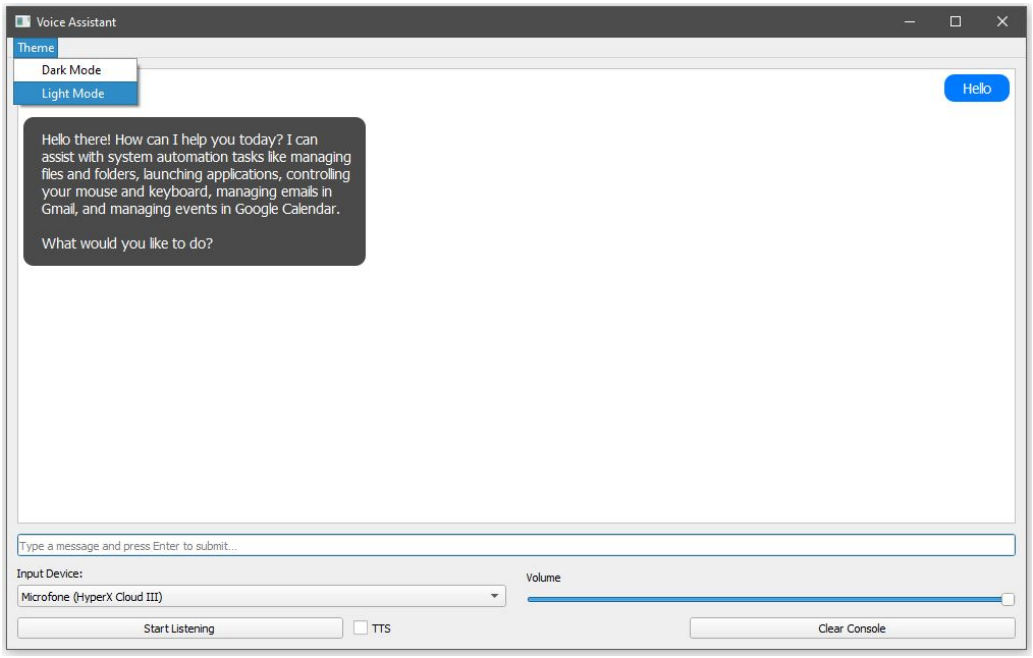

Figura 4.7 Interface em modo claro e menu de seleção de tema

combinando entrada por texto e fala, TTS, gestão visual de histórico e personalização do aspeto. A GUI completou, assim, a integração dos módulos desenvolvidos, funcionando como camada final de interação entre o utilizador e as funcionalidades internas do sistema, dando-se por encerrada a fase de implementação do sistema. A partir deste ponto, tornou-se essencial avaliar o desempenho da componente central que suporta todas as interações, os LLM. Dado que a execução das ações depende fortemente da capacidade dos LLM em interpretar instruções em linguagem natural, a etapa seguinte consistiu na realização de testes experimentais. Pretende-se analisar como a escolha do modelo influencia o desempenho global do sistema e a fiabilidade de cada ação.

Capítulo 5

Avaliação Experimental

Esta secção apresenta a metodologia e os resultados da avaliação do AD proposto. Na Secção 5.1 descreve-se o ambiente experimental (*hardware* e *software*) onde o assistente foi testado, bem como as ferramentas e configurações utilizadas. A Secção 5.2 expõe os resultados dos testes funcionais nas funcionalidades principais do assistente. A Secção 5.3 analisa o desempenho do sistema, destacando as taxas de sucesso e tempos médios de resposta obtidos com diferentes LLM. A Secção 5.4 discute a modularidade e extensibilidade da arquitetura diante da adição de novas funcionalidades. A Secção 5.5 descreve os resultados de um questionário de usabilidade aplicado a um grupo de utilizadores com base no método SUS. Por fim, a Secção 5.6 discute os pontos fortes e limitações observadas, avaliando em que medida os objetivos funcionais propostos foram alcançados.

5.1 Configuração Experimental

A avaliação do sistema foi feita num computador portátil com Windows 10, equipado com processador AMD Ryzen 5 4600H (6 núcleos, 3.00 GHz), 16 GB de RAM e uma placa gráfica NVIDIA GeForce GTX 1650 Ti. As interações por voz usaram um headset HyperX Cloud III, que serviu como microfone e auscultadores.

A metodologia consistiu em casos de uso manuais previamente definidos, projetados para cobrir todas as funcionalidades principais do assistente. Cada caso consistia num comando em linguagem natural ditado ou digitado pelo utilizador, cuja interpretação e execução eram verificadas. Em cada teste, avaliou-se se o LLM selecionado convertia corretamente a solicitação numa intenção estruturada (no formato JSON exigido pelo *parser* do sistema) e se o módulo correspondente executava a ação desejada. Foram ainda incluídos cenários de erro para testar a robustez: comandos inválidos, parâmetros em falta ou falhas simuladas nas API externas.

Para avaliar a capacidade do LLM, cada ação funcional foi testada com cinco variações linguísticas do mesmo pedido, cobrindo desde comandos simples a instruções mais descritivas. Também foram incluídos casos com expressões temporais ambíguas (por exemplo, “this week”, “next month”), resolvidas através da injeção do contexto temporal atual no *prompt* do LLM.

Como métricas de avaliação de desempenho, foram usadas a taxa de sucesso de coman-

dos e o tempo médio de execução. A taxa de sucesso corresponde à percentagem de comandos interpretados e executados corretamente pelos módulos do assistente. O tempo médio de execução mede o intervalo entre a submissão de um comando pelo utilizador e a devolução da resposta final pelo sistema. Este tempo inclui a latência introduzida pelo LLM, o processamento interno no núcleo do assistente e a execução da ação no módulo correspondente. Desta forma, o tempo de execução reflete a capacidade de interpretação do modelo e a eficiência global da arquitetura.

5.2 Resultados de Testes Funcionais

Os testes funcionais tiveram como principal objetivo verificar o correto funcionamento das funcionalidades implementadas e avaliar a capacidade do sistema em transformar instruções em linguagem natural em intenções estruturadas (JSON) que pudessem ser executadas pelos módulos de funcionalidades. Esta etapa não pretendeu ainda comparar diferentes modelos de linguagem, mas apenas confirmar a fiabilidade da arquitetura desenvolvida e do mecanismo de *intent parsing*. Para o efeito, foram testados todos os domínios implementados: gestão de ficheiros locais, integração com o Google Calendar e integração com o Gmail.

Na gestão de ficheiros, o sistema foi validado em operações como criação, escrita, leitura e eliminação de ficheiros, bem como manipulação de diretórios. As instruções, formuladas em linguagem natural com variações de redação, foram consistentemente interpretadas e executadas. A Figura 5.1 ilustra um exemplo concreto, em que o utilizador solicitou a criação de um ficheiro e a escrita de conteúdo no seu interior. O processo correspondeu à geração dos objetos JSON mostrados nas Listagens 5.1 e 5.2, que traduzem a intenção de criar e escrever num ficheiro. Estes exemplos são representativos do comportamento observado em todas as operações do módulo, confirmando a capacidade do assistente em mapear corretamente instruções variadas para ações específicas.

Listagem 5.1 JSON da intenção de criação de ficheiro

```
[
    {
        "action": "create_file",
        "filename": "test.txt"
    }
]
```

Listagem 5.2 JSON da intenção de escrita em ficheiro

```
[
    {
        "action": "write_file",
        "filename": "test.txt",
        "content": "Cheesecake Recipe:\n\nCrust:\n 1 1/2 cups graham..."
    }
]
```

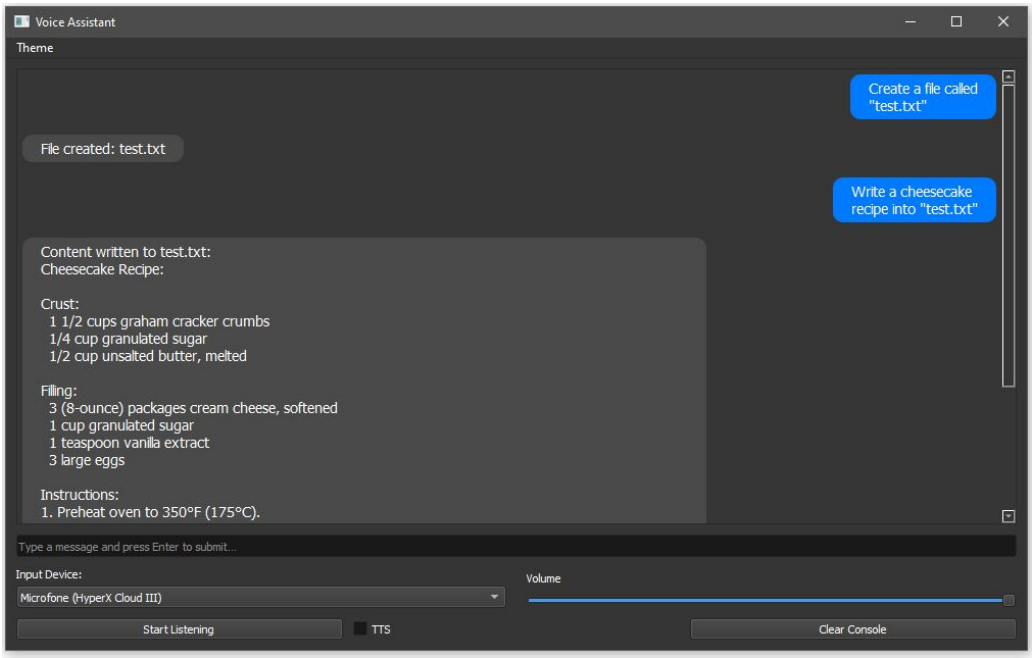

Figura 5.1 Exemplo de criação e escrita de ficheiro

Na integração com o Google Calendar, foi avaliada a criação, listagem e eliminação de eventos. Os testes incluíram expressões temporais absolutas e relativas (e.g., “amanhã”, “esta semana”) para verificar a eficácia da injeção de contexto temporal no *prompt*. A Figura 5.2 mostra um exemplo representativo, em que foi criado um evento, sendo possível ver a sua presença na listagem da agenda para a semana. As Listagens 5.3 e 5.4 apresentam os objetos JSON correspondentes, mostrando a normalização correta de datas e horas no formato ISO-8601. Estes resultados mostram que o módulo consegue traduzir instruções de calendário em operações coerentes sobre a API.

Listagem 5.3 JSON da intenção de criação de evento

```
[
    {
        "action": "create_event",
        "summary": "Dentist Appointment",
        "start_time": "2025-09-25T15:00:00"
    }
]
```

Listagem 5.4 JSON da intenção de listagem de eventos

```
[
    {
        "action": "list_events",
        "time_period": "2025-09-22/2025-09-28"
    }
]
```

Por fim, na integração com o Gmail, foram testadas a listagem, leitura, envio, marcação como lidas e eliminação de mensagens. Os resultados confirmaram que os filtros por remetente,

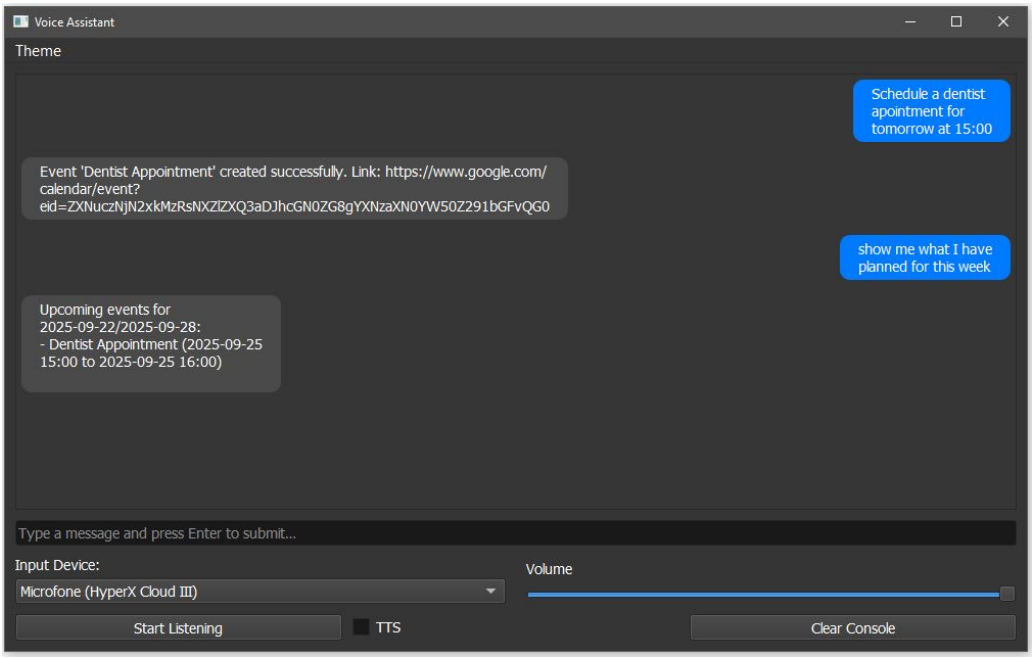

Figura 5.2 Exemplo de criação e listagem de eventos no Google Calendar

período temporal e estado de leitura foram corretamente aplicados, levando a operações consistentes na API. A Figura 5.3 apresenta um exemplo de listagem de mensagens e a Listagem 5.5 mostra o JSON correspondente. Estes resultados mostram que o módulo de *e-mail* cumpre as funcionalidades previstas.

Listagem 5.5 JSON da intenção de listagem de emails

```
[
    {
        "action": "list_emails",
        "label": "INBOX",
        "date_period": "2025-09-24"
    }
]
```

Em síntese, os testes funcionais confirmaram que o sistema é capaz de executar corretamente as operações previstas em cada módulo, traduzindo instruções em linguagem natural em intenções estruturadas e acionáveis. Estes testes foram realizados com o modelo Gemini 2.0 Flash, escolhido como referência para validar as funcionalidades. No entanto, como se verá na secção seguinte, o desempenho do sistema depende do LLM utilizado, sendo por isso relevante comparar diferentes modelos em termos de precisão de *parsing* e tempos de resposta.

5.3 Análise de Desempenho

Para avaliar o efeito dos diferentes LLM no desempenho do sistema, foi definido um conjunto de comandos representativos de uso real. Para cada ação prevista nos módulos, foram formuladas cinco instruções diferentes, semanticamente equivalentes, mas escritas de forma ligeiramente

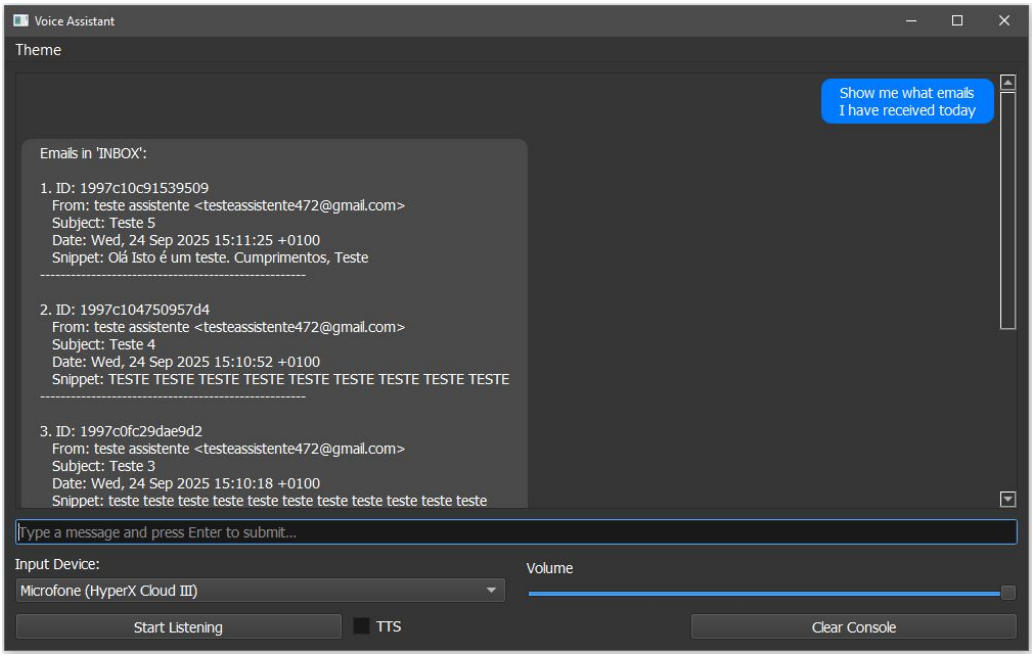

Figura 5.3 Exemplo de listagem de *e-mails*

distinta. O objetivo foi verificar se os LLM conseguiam reconhecer corretamente a intenção e os parâmetros mesmo quando os pedidos eram expressos de forma variada.

A escolha dos modelos para estes testes foi feita de forma a garantir uma representação variada de tamanhos, incluindo modelos mais pequenos (Llama 3.2 1B Instruct, Mistral 7B Instruct, Llama 3 8B Instruct), um modelo intermédio (Llama 3.3 70B Instruct) e modelos de grande dimensão (Gemini 2.0 Flash, Gemini 2.5 Flash e DeepSeek V3). Esta diversidade permitiu observar de que forma a escala do modelo influencia a capacidade de interpretar instruções. Outro critério foi a acessibilidade, pelo que se privilegiaram serviços gratuitos disponíveis através de API públicas, assegurando que os testes pudessem ser reproduzidos sem necessidade de infraestrutura dedicada.

A Tabela 5.1 apresenta os resultados detalhados, mostrando a taxa de sucesso por comando e modelo. Observa-se que, em todas as categorias, os modelos de maior dimensão mantêm desempenho consistente, enquanto os menores apresentam falhas distribuídas em várias tarefas. Os resultados mostram que a taxa de sucesso depende fortemente do modelo utilizado. O Llama 3.2 1B Instruct, sendo o mais pequeno, apresentou o pior desempenho, com falhas em mais de metade dos comandos testados. As dificuldades estiveram sobretudo ligadas ao cumprimento do formato JSON exigido pelo *parser*, sendo frequente a omissão de parâmetros obrigatórios, a invenção de atributos ou a devolução de respostas fora do formato esperado. O Mistral 7B Instruct e o Llama 3.3 8B Instruct tiveram resultados mais equilibrados, mas as falhas que ocorreram tiveram a mesma origem, relacionadas com erros de formatação das intenções. Em contraste, os modelos de maior dimensão, como o Llama 3.3 70B Instruct, Gemini 2.0 Flash, Gemini 2.5 Flash e DeepSeek V3, atingiram taxas de sucesso perfeitas, conseguindo interpretar todas as instruções. Estes resultados confirmam que a escala do modelo influencia diretamente a precisão da interpretação.

Tabela 5.1 Taxa de sucesso por tarefa e LLM. [L] = Local, [R] = Remote

Categoria	Comando	Llama 3.2 1B Instruct	Mistral 7B Instruct	Llama 3 8B Instruct	Llama 3.3 70B Instruct	Gemini 2.0 Flash	Gemini 2.5 Flash	DeepSeek V3 0324
Gestão de ficheiros [L]	create a folder	80%	100%	100%	100%	100%	100%	100%
	create a file	20%	80%	80%	100%	100%	100%	100%
	write text into file	20%	80%	100%	100%	100%	100%	100%
	append text to file	80%	100%	100%	100%	100%	100%	100%
	read the contents of file	40%	100%	100%	100%	100%	100%	100%
	delete file	80%	100%	100%	100%	100%	100%	100%
	delete folder	80%	100%	80%	100%	100%	100%	100%
	list contents of directory	80%	80%	80%	100%	100%	100%	100%
	rename file	60%	60%	100%	100%	100%	100%	100%
	copy file to another folder	80%	80%	80%	100%	100%	100%	100%
	move file to another folder	0%	100%	60%	100%	100%	100%	100%
Google Calendar [R]	list scheduled events	20%	100%	100%	100%	100%	100%	100%
	schedule an event	60%	80%	100%	100%	100%	100%	100%
	cancel a scheduled event	0%	80%	100%	100%	100%	100%	100%
Integração Gmail [R]	list emails	40%	80%	80%	100%	100%	100%	100%
	send email	40%	100%	60%	100%	100%	100%	100%
	read email	60%	100%	80%	100%	100%	100%	100%
	mark email as read	0%	60%	100%	100%	100%	100%	100%
	delete email	0%	100%	100%	100%	100%	100%	100%
Totais		**44%**	**88%**	**89%**	**100%**	**100%**	**100%**	**100%**

Para além da taxa de sucesso, foi também avaliado o tempo médio de execução de cada comando, apresentado na Tabela 5.2. Este valor corresponde ao tempo total decorrido desde o envio do comando, passando pela comunicação com a API do LLM, pelo processamento da instrução e pela execução da ação no módulo correspondente, até à apresentação da mensagem de resposta ao utilizador. Cada valor apresentado resulta da média de cinco execuções por comando. Os resultados permitem comparar de forma direta a rapidez de resposta entre diferentes modelos e fornecedores. No caso dos modelos Gemini, os testes foram realizados através da API oficial da Google e os restantes modelos foram acedidos através do serviço Novita AI.

Os resultados mostram diferenças significativas entre modelos. Apesar de ser um dos maiores modelos, o menor tempo médio foi registado pelo Gemini 2.0 Flash (0.93 s). O Mistral 7B Instruct (1.33 s) e o Llama 3.2 1B Instruct (1.55 s) também apresentaram tempos reduzidos, mas estes não compensam as falhas frequentes na interpretação observadas na Tabela 5.1. Em contraste, modelos de grande dimensão, como o DeepSeek V3 (2.28 s) e o Llama 3.3 70B Instruct (2.70 s), revelaram tempos bastante superiores, possivelmente devido à sua maior complexidade e à latência acrescida do fornecedor. Estes resultados confirmam que o tempo de resposta não depende apenas da escala do modelo.

Tabela 5.2 Tempo médio (s) por comando e LLM (média de 5 execuções por comando)

Categoria	Comando	Llama 3.2 1B Instruct	Mistral 7B Instruct	Llama 3 8B Instruct	Llama 3 70B Instruct	Gemini 2.0 Flash	Gemini 2.5 Flash	DeepSeek V3 0324
Gestão de ficheiros [L]	create a folder	1.12	1.12	2.01	2.21	0.68	1.18	1.92
	create a file	1.29	1.00	2.00	2.06	0.61	1.14	1.87
	write text into file	1.69	1.33	2.04	3.02	0.65	1.27	2.52
	append text to file	1.40	1.29	1.89	2.61	0.76	1.40	2.17
	read the contents of file	1.48	1.20	1.79	2.41	0.59	1.48	2.12
	delete file	1.31	1.04	1.99	2.09	0.62	1.25	1.92
	delete folder	1.47	1.31	2.07	2.55	0.68	1.46	2.23
	list contents of directory	1.35	1.12	1.86	2.33	0.66	1.29	2.06
	rename file	1.53	1.27	2.08	2.73	0.75	1.46	2.33
	copy file to another folder	1.46	1.25	2.08	2.62	0.72	1.42	2.25
	move file to another folder	1.64	1.36	2.16	2.76	0.78	1.55	2.44
Google Calendar [R]	list scheduled events	1.55	1.38	2.17	2.84	1.08	1.84	2.30
	schedule an event	1.53	1.41	2.16	3.11	1.20	1.88	2.44
	cancel a scheduled event	1.48	1.36	2.16	3.18	1.42	1.81	2.31
Integração Gmail [R]	list emails	2.74	1.52	2.52	3.42	2.08	3.03	2.55
	send email	1.43	1.57	1.86	3.73	1.42	2.43	2.62
	read email	1.50	1.24	1.75	2.45	0.94	1.84	2.14
	mark email as read	1.55	1.79	1.95	2.68	1.11	2.33	2.71
	delete email	1.88	1.74	2.08	2.47	0.97	2.24	2.40
Tempo médio total		**1.55**	**1.33**	**2.03**	**2.70**	**0.93**	**1.70**	**2.28**

Estes valores estão sujeitos às condições da ligação à internet no momento dos testes, em particular durante a instanciação de cada pedido. Outros fornecedores não incluídos na avaliação poderão ainda oferecer tempos de resposta mais rápidos. Ainda assim, os resultados obtidos permitem ter uma visão comparativa clara entre os modelos e fornecedores analisados.

Para efeitos de utilização geral, o Gemini 2.0 Flash revelou-se a opção mais equilibrada, combinando uma taxa de sucesso perfeita com o menor tempo médio de execução entre todos os modelos avaliados. Ainda assim, é importante ter em conta que todos estes modelos estão sujeitos a limites diários de uso, o que pode exigir a utilização de alternativas ou a alternância entre fornecedores em cenários de maior intensidade de utilização.

5.4 Avaliação de Modularidade e Extensibilidade

Um dos objetivos centrais do projeto foi garantir uma arquitetura modular e fácil de expandir. Os resultados confirmam que este objetivo foi alcançado. A estrutura baseada em *plugins* mostrou-se eficaz dado que novos módulos podem ser adicionados sem a necessidade de alterar o núcleo do sistema. Isto é possível graças à utilização de uma classe base abstrata para a

definição de ações, da qual todos os módulos herdam. Assim, cada nova funcionalidade, como a integração com um serviço externo, é encapsulada num ficheiro separado que implementa a interface comum.

Sempre que o sistema é iniciado, o núcleo carrega dinamicamente os módulos definidos no ficheiro de configuração. Assim, para expandir as funcionalidades, basta adicionar a referência a um novo módulo na configuração e garantir que o respetivo ficheiro está na diretoria correta, sem necessidade de alterações no resto do código. Cada módulo declara de forma explícita as ações que suporta, incluindo descrições e exemplos de utilização. Isto permite ao LLM conhecer imediatamente as novas ações disponíveis, sem necessidade de alterar manualmente as instruções fornecidas pelo sistema.

A separação clara entre componentes de entrada e saída, interpretação da linguagem natural e execução das ações assegura baixa dependência entre módulos, reduzindo os riscos na introdução de novas funcionalidades e facilitando a manutenção do código. Além disso, a existência de uma interface abstrata para os LLM confirmou a flexibilidade do sistema. Durante os testes, foi possível alternar entre diferentes modelos (Gemini, Llama, DeepSeek, entre outros) apenas com alterações na configuração, sem impactar o funcionamento dos restantes componentes.

Estas características foram validadas nos testes efetuados. Os módulos de integração com Google Calendar e Gmail, descritos nas Secções 4.5 e 4.6, foram incorporados simplesmente através da criação dos respetivos ficheiros de módulo, sem qualquer modificação no código do núcleo. A arquitetura proposta é flexível e escalável, permitindo a evolução contínua do sistema sem comprometer funcionalidades existentes. A avaliação confirma que as decisões de projeto adotadas foram eficazes e garantem a sustentabilidade do assistente no futuro.

5.5 Questionário de Usabilidade

Para avaliar a facilidade de uso e validar o assistente desenvolvido com base em impressões de utilizadores reais, foi realizado um questionário de usabilidade com base no *System Usability Scale* (SUS) [9]. O inquérito foi implementado através da plataforma Google Forms, tendo recolhido um total de 15 respostas. A maioria dos participantes (11) tinha entre 18 e 25 anos, 3 situavam-se na faixa dos 41 aos 65 anos e 1 participante tinha até 18 anos. Na Figura 5.4 é possível visualizar o gráfico que representa as faixas etárias, podendo então concluir que o questionário abrangeu diferentes públicos.

De forma a minimizar o esforço técnico necessário por parte dos participantes e garantir condições de teste homogéneas, foi fornecido um pacote `.zip` com todos os ficheiros necessários para a execução do assistente. Este pacote estava configurado para usar o modelo Gemini 2.0 Flash, previamente identificado com melhor equilíbrio entre desempenho e tempo de resposta, bem como uma conta Google já autenticada para permitir a integração direta com Gmail e Google Calendar.

Para simplificar ainda mais o processo de arranque, foi incluído um ficheiro `.bat` que

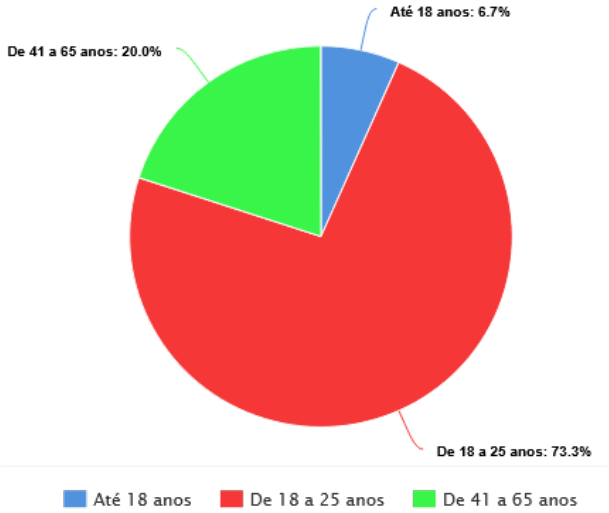

Figura 5.4 Caracterização dos participantes através da idade

automatiza a instalação das dependências e inicia o assistente. A única ação requerida por parte dos utilizadores era a instalação prévia do Python 3.10. Esta abordagem visou assegurar que mesmo participantes com pouca familiaridade com programação pudessem interagir com o sistema sem dificuldades técnicas, focando-se na utilização prática e avaliação da experiência de uso.

5.5.1 Tarefas Propostas

Antes das questões de usabilidade, foi pedido aos participantes que executassem um conjunto de tarefas práticas, com o objetivo de avaliar o comportamento do assistente em situações reais de utilização. Estas tarefas foram organizadas por módulos funcionais, Google Calendar, Gmail, Sistema de Ficheiros e Abertura de Aplicações, e tinham como objetivo verificar se os utilizadores conseguiam, sem instruções técnicas adicionais, alcançar os objetivos pretendidos com base em comandos em linguagem natural. A seguir, descreve-se cada conjunto de tarefas, acompanhado de uma análise dos respetivos resultados.

Agenda (Google Calendar)

1. Agendar uma consulta no dentista para amanhã às 15:00.
2. Agendar uma reunião para a próxima semana às 10:25.
3. Pedir ao assistente para mostrar os eventos marcados para este mês e verificar a sua presença.
4. Apagar os eventos marcados anteriormente.

Todos os utilizadores conseguiram completar estas tarefas sem dificuldades significativas. A integração com o Google Calendar mostrou-se robusta e responsiva, mesmo quando as instruções foram formuladas de formas distintas. A única limitação identificada surgiu na segunda tarefa; Ao pedir para agendar uma reunião “para a próxima semana às 10:25”, o

assistente agendava automaticamente para segunda-feira, sem pedir a confirmação do dia específico. Para mitigar este comportamento, foi adicionada ao *prompt* do LLM a seguinte instrução: *"If a user asks to schedule an event without a clear day, such as 'next week' or 'next month', ask to specify the exact date"*. Esta modificação garante que, sempre que houver ambiguidade temporal, o assistente solicita esclarecimento adicional ao utilizador antes de executar a ação.

E-mails (Gmail)

1. Enviar um e-mail de teste para a conta associada com o assunto e a mensagem à escolha.
2. Pedir ao assistente para mostrar a caixa de entrada e ler um dos e-mails já existentes.

As tarefas relacionadas com o Gmail foram concluídas com sucesso por todos os participantes, sem quaisquer dificuldades ou falhas reportadas. O envio e leitura de mensagens foi executado com precisão e o assistente demonstrou compreender corretamente os parâmetros especificados em linguagem natural, como o assunto e o corpo da mensagem. Como se tratavam de tarefas simples e com passos bem definidos, não foram detetadas ambiguidades ou problemas de usabilidade.

Ficheiros e Pastas (Sistema Operativo)

1. Criar um ficheiro chamado "teste.txt" numa pasta à escolha.
2. Escrever no ficheiro um texto à escolha.
3. Mostrar o conteúdo do ficheiro.
4. Apagar o ficheiro.

Foi neste módulo que se registaram mais dificuldades por parte dos utilizadores. A principal limitação surgiu na segunda tarefa, em que os participantes pediam ao assistente para escrever algo livremente, por exemplo, "escreve no ficheiro algo que aches interessante". O assistente, interpretando de forma demasiado literal, escrevia exatamente "algo que aches interessante" no ficheiro. Esta interpretação limitada decorre da tendência dos LLMs para cumprir instruções literalmente quando não são instruídos de forma explícita a inferir intenções. Para resolver este problema, foi adicionada ao *prompt* do LLM uma instrução adicional para evitar copiar literalmente o que o utilizador diz quando este pede conteúdos criativos ou abertos.

Outro problema identificado nesta secção relacionou-se com a especificação de diretorias. Quando os participantes pediam para criar ou guardar ficheiros no "ambiente de trabalho", o assistente tentava utilizar o caminho `C:\Users\Utilizador\Desktop`, o que falhava por incompatibilidades com o nome do utilizador real ou permissões do sistema. Para resolver isto, foi novamente adicionada ao *prompt* uma instrução para pedir o caminho ao utilizador caso não tenha a certeza.

Abrir Aplicações e Pergunta Genérica

1. Pedir ao assistente para abrir o Bloco de Notas.

2. Com o cursor no Bloco de Notas, ditar uma frase à escolha para o assistente escrever.
3. Colocar uma pergunta aberta ao assistente, como “Qual é a capital da França?”

Estas tarefas foram executadas com sucesso por todos os utilizadores. A abertura de aplicações funcionou corretamente, e os participantes conseguiram ditar ou escrever instruções sem dificuldades. A funcionalidade de escrita em aplicações externas, como o Bloco de Notas, revelou-se eficaz e responsiva. As perguntas abertas foram bem compreendidas, com respostas coerentes e completas por parte do modelo Gemini 2.0 Flash.

5.5.2 Resultados do Questionário

O questionário de usabilidade aplicado baseia-se no *System Usability Scale* (SUS) [9], uma métrica criada por John Brooke em 1996, utilizada para avaliar a facilidade de utilização de um sistema. A escala é composta por 10 afirmações, com respostas numa escala de Likert de 1 a 5, onde 1 representa “discordo totalmente” e 5 representa “concordo totalmente”. O valor 3 corresponde a uma posição neutra.

O cálculo da pontuação do SUS é feito da seguinte forma: nas perguntas ímpares (positivas), subtrai-se 1 ao valor da resposta e nas perguntas pares (negativas), subtrai-se o valor da resposta a 5. Com isto, cada pergunta contribui com um valor entre 0 e 4. A soma total dessas pontuações é depois multiplicada por 2.5, resultando numa escala final entre 0 e 100. Quanto mais alta for a pontuação, melhor é a usabilidade percebida. A Figura 5.5 apresenta a escala usada para interpretar o resultado.

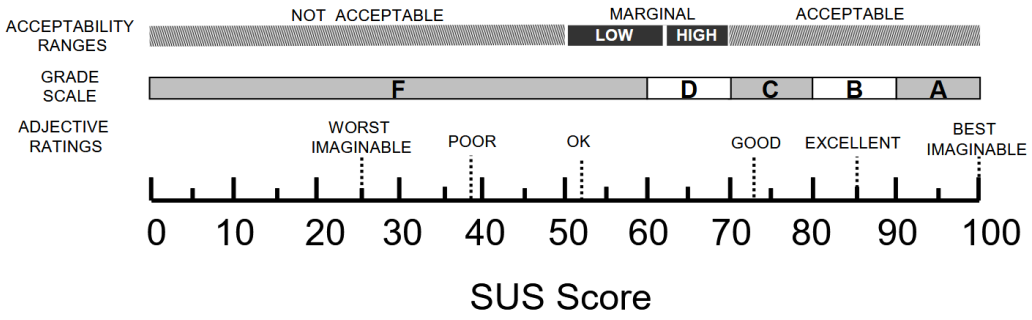

Figura 5.5 Escala de interpretação do SUS Score [7]

Segue abaixo o cálculo da pontuação média de cada pergunta do SUS com base nas 15 respostas recolhidas:

- Pergunta 1 – Gostaria de utilizar este assistente com frequência.
  $((1-1)+(4\times 3-4)+(4\times 4-4)+(3\times 5-3))/15 = 32/15 = 2.1333$
- Pergunta 2 – Achei o assistente desnecessariamente complexo.
  $((5-1)\times 10+(5-2)\times 5)/15 = 55/15 = 3.6667$
- Pergunta 3 – Achei o assistente fácil de usar.
  $((1\times 3-1)+(7\times 4-7)+(7\times 5-7)/15 = 51/15 = 3.4000$

- Pergunta 4 – Precisaria de ajuda técnica para utilizar o assistente.
  $((5-1)\times 8+(5-2)\times 6+(5-3)\times 1)/15 = 52/15 = 3.4667$
- Pergunta 5 – As funcionalidades estão bem integradas.
  $((1\times 3-1)+(8\times 4-8)+(6\times 5-6))/15 = 50/15 = 3.3333$
- Pergunta 6 – O assistente apresenta muita inconsistência.
  $((5-1)\times 5+(5-2)\times 7+(5-3)\times 3)/15 = 47/15 = 3.1333$
- Pergunta 7 – Imagino que a maioria das pessoas aprenderia a usar este assistente rapidamente.
  $((1\times 2-1)+(6\times 4-6)+(8\times 5-8))/15 = 51/15 = 3.4000$
- Pergunta 8 – Achei este assistente difícil de usar.
  $((5-1)\times 12+(5-2)\times 3)/15 = 59/15 = 3.8000$
- Pergunta 9 – Senti-me confiante ao utilizar o assistente.
  $((2\times 3-2)+(5\times 4-5)+(7\times 5-7))/15 = 55/15 = 3.1333$
- Pergunta 10 – Precisei de aprender muitas coisas novas antes de usar o assistente.
  $((5-1)\times 10+(5-2)\times 5)/15 = 55/15 = 3.6667$
- Resultado Usabilidade:
  $(2.1333+3.6667+3.4000+3.4667+3.3333+3.1333+3.4000+3.8000+3.3571)\times 2.5 = 33.1333\times 2.5 = 82.83$

Assim, a pontuação final do SUS foi de **82.83**, o que indica uma perceção muito boa, próxima de excelente, de usabilidade do assistente por parte dos participantes. Este resultado valida a abordagem adotada e demonstra o potencial do assistente para ser utilizado em ambientes reais.

5.5.3 Análise das Respostas Abertas

Para além das respostas quantitativas da escala SUS, o questionário incluía três questões abertas que procuravam recolher opiniões mais detalhadas sobre a experiência de utilização do assistente. Estas respostas permitiram identificar os pontos fortes e as limitações sentidas pelos participantes. As três perguntas colocadas foram as seguintes:

1. O que achou mais fácil e mais difícil ao utilizar o assistente?
2. Há alguma funcionalidade que sentiu falta ou que gostaria que fosse melhorada?
3. Tem algum comentário ou sugestão adicional sobre a experiência de utilização?

Na primeira questão, os participantes referiram que tarefas diretas e instruções simples foram geralmente bem compreendidas e executadas. O envio de e-mails e a interação com

o Google Calendar foram frequentemente descritas como fáceis. No entanto, surgiram dificuldades em tarefas que exigem mais contexto ou interpretação, como foi o caso da escrita em ficheiros, cuja literalidade na resposta já foi discutida e resolvida anteriormente. Houve ainda referências a problemas pontuais na criação de ficheiros em diretórios específicos e na abertura de algumas aplicações.

Relativamente à segunda questão, alguns utilizadores referiram que o assistente deveria ser capaz de adaptar a resposta à língua usada pelo utilizador. Outros sugeriram melhorias na interface gráfica, como o aumento do tamanho dos elementos ou a adição de um texto introdutório que indicasse como começar. Foram ainda sugeridas funcionalidades adicionais como integração com a meteorologia, criação de alarmes, envio de *e-mails* com base em *templates*, e maior compatibilidade com ficheiros na *cloud*.

Na última questão, dedicada a sugestões livres, os participantes manifestaram, de forma geral, uma opinião positiva, reconhecendo utilidade e facilidade de uso no assistente. Ainda assim, destacaram a importância de tornar o sistema mais flexível, menos literal e com mais opções adaptadas ao dia a dia.

5.6 Discussão

A avaliação experimental mostrou que o sistema desenvolvido atinge os principais objetivos definidos para o projeto. O assistente foi capaz de interpretar comandos em linguagem natural, executar ações em diferentes módulos e fornecer respostas adequadas ao utilizador. As funcionalidades essenciais, gestão de ficheiros, integração com Google Calendar e Gmail, foram validadas com sucesso, confirmando que a arquitetura modular e a estratégia de integração de LLM funcionam de forma eficaz.

Entre os pontos fortes destacam-se três aspetos centrais. Em primeiro lugar, a correta interpretação de linguagem natural, mesmo quando os pedidos foram formulados de forma diferente, confirma a eficácia da integração dos LLM. Em segundo lugar, a arquitetura modular permitiu adicionar novas funcionalidades sem alterações no núcleo, validando a extensibilidade do sistema. Por fim, a interface gráfica mostrou-se intuitiva, combinando entrada por texto e voz com mecanismos de *feedback* visual e auditivo.

Apesar dos resultados positivos, algumas limitações devem ser destacadas. O desempenho do sistema depende fortemente dos LLM e das API externas utilizadas, o que introduz latência e eventuais falhas em cenários de maior exigência temporal.

Questões de segurança e privacidade também permanecem relevantes. Tal como descrito na Secção 3.4, as chaves de API e os *tokens* de autenticação são atualmente guardados em ficheiros locais de configuração, uma solução prática para o desenvolvimento mas pouco adequada para a proteção de credenciais sensíveis. Em cenários reais, faria mais sentido recorrer a mecanismos de armazenamento cifrado ou a variáveis de ambiente, reduzindo o risco de exposição.

No geral, o sistema demonstrou ser um assistente útil e eficaz, capaz de apoiar o utilizador

em tarefas do dia a dia como a gestão de ficheiros, a organização do calendário e o tratamento de *e-mails*. Todas as funcionalidades planeadas foram alcançadas, mostrando que o protótipo é fiável na execução das ações solicitadas em linguagem natural. A arquitetura modular revelou-se flexível e escalável, permitindo a integração de novos serviços sem comprometer o funcionamento existente. Ao mesmo tempo, a interface abstrata para os LLM assegurou compatibilidade com vários modelos, facilitando a evolução futura. A avaliação confirma que as decisões de projeto, desde a modularidade até à integração incremental de módulos, tiveram impacto direto no bom desempenho do assistente e no cumprimento dos objetivos definidos.

Para além da avaliação funcional, os resultados do questionário de usabilidade trouxeram uma perspetiva complementar focada na experiência prática dos utilizadores. Com uma pontuação média de 82.83 na métrica SUS, os participantes demonstraram um nível elevado de satisfação com o assistente. As respostas abertas reforçaram esta perceção, referindo a facilidade de utilização e a utilidade das funcionalidades disponíveis. Ao mesmo tempo, surgiram sugestões relevantes para melhorias futuras, como maior adaptabilidade à linguagem usada, maior clareza na interface e novas integrações. Estes resultados mostram que o sistema funciona em cenários reais e validam o seu potencial como assistente digital útil e acessível.

Capítulo 6

Conclusão e Trabalho Futuro

Este capítulo está organizado em duas partes. Na Secção 6.1 apresentam-se as principais conclusões do trabalho desenvolvido, com destaque para o cumprimento dos objetivos definidos. Na Secção 6.2 são discutidas possíveis linhas de evolução, incluindo melhorias e novas funcionalidades a explorar.

6.1 Conclusão

Neste trabalho, pretendeu-se desenvolver um assistente digital para ambiente de trabalho com arquitetura modular, capaz de integrar ASR, TTS e uma interface gráfica, recorrendo a LLM para interpretar comandos em linguagem natural. Para isso, foi adotado um design modular, em que diferentes componentes especializados interpretam as intenções do utilizador através do LLM e executam autonomamente as ações correspondentes no sistema operativo ou em serviços externos. O protótipo resultante permite interação por voz ou por texto e a execução automática de tarefas no PC de acordo com as funcionalidades previstas, demonstrando ser flexível e expansível conforme planeado.

Os testes realizados mostraram que o desempenho do sistema depende diretamente do modelo de linguagem utilizado. Com LLM de menor dimensão verificaram-se falhas ocasionais, sobretudo no cumprimento rigoroso do formato esperado, mas com modelos de maior escala os resultados foram praticamente perfeitos. No total de 95 cenários de teste avaliados, o assistente cumpriu com sucesso as tarefas previstas em todos os módulos, desde operações locais no sistema operativo até à integração com serviços externos como o Google Calendar e o Gmail, mantendo tempos médios de resposta baixos, na ordem dos décimos de segundo.

Para além dos testes automáticos, foi realizado um questionário de usabilidade com 15 participantes para perceber como o assistente é recebido por utilizadores reais. Os resultados foram bastante positivos, com uma pontuação SUS final de **82.83**, o que reflete uma excelente perceção de usabilidade. As tarefas mais diretas, como enviar e-mails ou marcar eventos no calendário, foram geralmente concluídas sem dificuldades. Houve alguns casos iniciais em que o assistente interpretava comandos de forma demasiado literal, mas essas situações foram corrigidas com ajustes ao sistema. As respostas abertas mostraram ainda sugestões úteis para

futuras melhorias, como a possibilidade de maior personalização e mais funcionalidades. De forma geral, as respostas dos participantes mostraram que o assistente é fácil de usar, prático e útil para o tipo de tarefas que pretende resolver.

Em síntese, a experiência de desenvolvimento e os testes realizados mostram que a arquitetura escolhida permite evoluir o sistema sem comprometer a sua integridade. A modularidade garante que cada funcionalidade está isolada num módulo independente, o que torna possível adicionar ou remover serviços e trocar de LLM apenas com ajustes de configuração, sem alterar o núcleo. Graças a esta flexibilidade, o assistente demonstrou ser funcional, eficaz e facilmente extensível a outros contextos e integrações, sendo uma solução promissora no domínio dos assistentes pessoais baseados em LLM.

6.2 Trabalho Futuro

Numa futura iteração, será importante reforçar a segurança do sistema, em particular na gestão de credenciais. Recomenda-se o uso de variáveis de ambiente ou serviços dedicados para armazenar informações sensíveis (*tokens* e chaves de API). Isso evitaria expor credenciais em texto em claro e facilitaria a configuração do sistema em diferentes ambientes de execução.

Para além disso, a arquitetura modular adotada facilita a expansão do sistema, tornando simples a adição de novas funcionalidades. Assim, futuros desenvolvimentos devem explorar a integração de novos módulos, seguindo as sugestões deixadas pelos utilizadores, como a integração com informações meteorológicas, definição de alarmes, envio de *e-mails* com base em *templates* e compatibilidade alargada com ficheiros armazenados na *cloud*, garantindo que o assistente se mantém relevante e alinhado com as necessidades dos utilizadores.

Referências

[1] Adiwardana, D., Luong, M.-T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y., and Le, Q. V. (2020). Towards a human-like open-domain chatbot.

[2] AI, N. (2025). Novita ai – novita. `https://novita.ai/`. Acedido em 24 de novembro de 2025.

[3] Alagha, E. and Helbing, R. (2019). Evaluating the quality of voice assistants' responses to consumer health questions about vaccines: An exploratory comparison of alexa, google assistant and siri. *BMJ health care informatics*, 26.

[4] Anthropic (2025). Introducing claude 4. `https://www.anthropic.com/news/claude-4`. Acedido em 24 de novembro de 2025.

[5] Ayanouz, S., Abdelhakim, B. A., and Benhmed, M. (2020). A smart chatbot architecture based nlp and machine learning for health care assistance. In *Proceedings of the 3rd International Conference on Networking, Information Systems & Security*, NISS '20, New York, NY, USA. Association for Computing Machinery.

[6] Bai, Y., Kadavath, S., Kundu, S., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv:2212.08073*.

[7] Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Stud.*, 4:114–123.

[8] Berdasco, A., López, G., Díaz-Oreiro, I., Quesada, L., and Guerrero, L. A. (2019). User experience comparison of intelligent personal assistants: Alexa, google assistant, siri and cortana. In *International Conference on Ubiquitous Computing and Ambient Intelligence*.

[9] Brooke, J. (1995). Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189.

[10] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

[11] Caldarini, G., Jaf, S., and McGarry, K. (2022). A literature survey of recent advances in chatbots. *Information*, 13(1).

[12] Chowdhary, K. and Chowdhary, K. (2020). Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649.

[13] Chowdhery, A., Narang, S., Devlin, J., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv:2204.02311*.

[14] Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., and Ram, O. (2025). Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities.

[15] Cui, L., Huang, S., Wei, F., Tan, C., Duan, C., and Zhou, M. (2017). Superagent: A customer service chatbot for e-commerce websites. In *Proceedings of ACL 2017, system demonstrations*, pages 97–102.

[16] DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., and Zhang, C. (2025). Deepseek-v3 technical report.

[17] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

[18] Dubey, A. and et al. (2024). The Llama 3 Herd of Models. *arXiv:2407.21783*.

[19] Epstein, J. and Klinkenberg, W. (2001). From eliza to internet: a brief history of computerized assessment. *Computers in Human Behavior*, 17(3):295–314.

[20] Fallows, J. (2008). "clippy" update – now, with organizational anthropology! `https://www.theatlantic.com/technology/archive/2008/04/-quot-clippy-quot-update-now-with-organizational-anthropology/8006/`. Acedido em 24 de novembro de 2025.

[21] Foster, K. (2023). What is automatic speech recognition? a comprehensive overview of asr technology. `https://www.assemblyai.com/blog/what-is-asr/`. Acedido em 24 de novembro de 2025.

[22] Google (2025a). Gemini developer api – google ai. `https://ai.google.dev/gemini-api/docs`. Acedido em 24 de novembro de 2025.

[23] Google (2025b). Google calendar api overview. `https://developers.google.com/workspace/calendar/api/guides/overview`. Acedido em 24 de novembro de 2025.

[24] Google Cloud (2025). Gemini 2.0 flash model card. Acedido em 24 de novembro de 2025.

[25] Goulão, A., Dias, D., Ferreira, A., and Leite, N. (2025a). A modular digital assistant for windows with large language models. In *RECPAD 2025 – Portuguese Conference on Pattern Recognition*, Aveiro, Portugal.

[26] Goulão, A., Dias, D., Ferreira, A., and Leite, N. (2025b). A personal digital assistant enhanced with artificial intelligence techniques. In *INForum 2025 – Simpósio de Informática*, Évora, Portugal.

[27] Hoy, M. (2018). Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, 37:81–88.

[28] Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., and Liu, T. (2024). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43.

[29] Jaybhaye, D., Manchekar, A., Dixit, A., Ingle, A., and Khatib, F. (2023). Comparative analysis of voice powered assistants. *International Journal for Research in Applied Science and Engineering Technology*, 11:1892–1898.

[30] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b.

[31] Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., and Chaplot, D. S. (2024a). Mixtral of experts. *arXiv:2401.04088*.

[32] Jiang, G., Shi, X., and Luo, Q. (2024b). LLM-collaboration on automatic science journalism for the general audience.

[33] Khurana, D., Koli, A., Khatter, K., and Singh, S. (2022). Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3):3713–3744.

[34] Lambert, N., Castricato, L., von Werra, L., and Havrilla, A. (2022). Illustrating reinforcement learning from human feedback (rlhf). *Hugging Face Blog*. https://huggingface.co/blog/rlhf.

[35] LLM, A. (2025). Awan LLM: Unlimited tokens, cost-effective LLM inference api platform. `https://www.awanllm.com/`. Acedido em 24 de novembro de 2025.

[36] Maedche, A., Legner, C., Benlian, A., and et al. (2019). Ai-based digital assistants: Opportunities, threats, and research perspectives. *Business and Information Systems Engineering*, 11(4):285–297.

[37] Microsoft (2025). End of support for cortana — microsoft support. `https://support.microsoft.com/en-us/topic/end-of-support-for-cortana-d025b39f-ee5b-4836-a954-0ab646ee1efa`. Acedido em 24 de novembro de 2025.

[38] Moskvitch, K. (2017). The machines that learned to listen. `https://www.bbc.com/future/article/20170214-the-machines-that-learned-to-listen`. Acedido em 24 de novembro de 2025.

[39] Nadelson, T. (1987). The inhuman computer / the too-human psychotherapist. *American Journal of Psychotherapy*, 41(4):489–498. PMID: 3434644.

[40] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2024). A comprehensive overview of large language models.

[41] Niu, Q., Chen, K., Li, M., Feng, P., Bi, Z., Yan, L. K., Zhang, Y., Yin, C. H., Fei, C., Liu, J., Peng, B., Wang, T., Wang, Y., Chen, S., and Liu, M. (2024). From text to multimodality: Exploring the evolution and impact of large language models in medical practice.

[42] OpenAI (2024). GPT-4 technical report.

[43] OpenAI (2025). Introducing GPT-5. https://openai.com/index/introducing-gpt-5/. Acedido em 24 de novembro de 2025.

[44] Palanica, A. and Fossat, Y. (2021). Medication name comprehension of intelligent virtual assistants: A comparison of amazon alexa, google assistant, and apple siri between 2019 and 2021. *Frontiers in Digital Health*, 3.

[45] Pangarkar, T. (2025). Intelligent virtual assistant statistics 2025 by support, technology, tasks. https://scoop.market.us/intelligent-virtual-assistant-statistics/. Acedido em 24 de novembro de 2025.

[46] Python Software Foundation (2025). Tkinter — python interface to tcl/tk. https://docs.python.org/3/library/tkinter.html. Acedido em 24 de novembro de 2025.

[47] Rella, S. (2022). Essential guide to automatic speech recognition technology. https://developer.nvidia.com/blog/essential-guide-to-automatic-speech-recognition-technology/. Acedido em 24 de novembro de 2025.

[48] Singh, S. (2024). Understanding the power and potential of natural language processing. https://codestax.medium.com/understanding-the-power-and-potential-of-natural-language-processing-5276c2dc6cba. Acedido em 24 de novembro de 2025.

[49] Team, Q. (2025). Qwen3-coder: Agentic coding in the world. https://qwenlm.github.io/blog/qwen3-coder/. Acedido em 24 de novembro de 2025.

[50] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., and Babaei, Y. (2023). Llama 2: Open foundation and fine-tuned chat models.

[51] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[52] White, R. W. (2019). Multi-device digital assistance. https://cacm.acm.org/opinion/multi-device-digital-assistance/. Acedido em 24 de novembro de 2025.

[53] Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., and Lv, C. (2025). Qwen3 technical report.

[54] Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., and Huang, F. (2024). Qwen2.5 technical report. *CoRR*, abs/2412.15115.

[55] Zheng, T., Grosse, E., Morana, S., and Glock, C. (2024). A review of digital assistants in production and logistics: applications, benefits, and challenges. *International Journal of Production Research*, 62:1–27.

[56] Zhou, T. (2023). How do voice assistants like alexa siri actually work? `https://medium.com/@zhouxiaogan0/how-do-voice-assistants-like-alexa-siri-actually-work-1885dce1f683`. Acedido em 24 de novembro de 2025.

Anexo A

Questionário de Usabilidade

Este anexo apresenta o questionário de usabilidade utilizado na avaliação do AD, disponibilizado através da plataforma Google Forms. A Figura A.1 mostra o ecrã inicial do formulário, onde é apresentado o objetivo do inquérito. A Figura A.2 apresenta o guia de instalação e utilização do assistente, incluindo o link do repositório e instruções para iniciar a aplicação. A Figura A.3 descreve as funcionalidades disponíveis no sistema. Na Figura A.4, são listadas as tarefas a realizar, que permitem testar as diferentes capacidades do assistente. A Figura A.5 apresenta o quadro com as questões de usabilidade, com base na métrica SUS. Por fim, a Figura A.6 inclui a secção final do formulário, com perguntas abertas onde os participantes puderam partilhar comentários, dificuldades e sugestões para melhorias futuras.

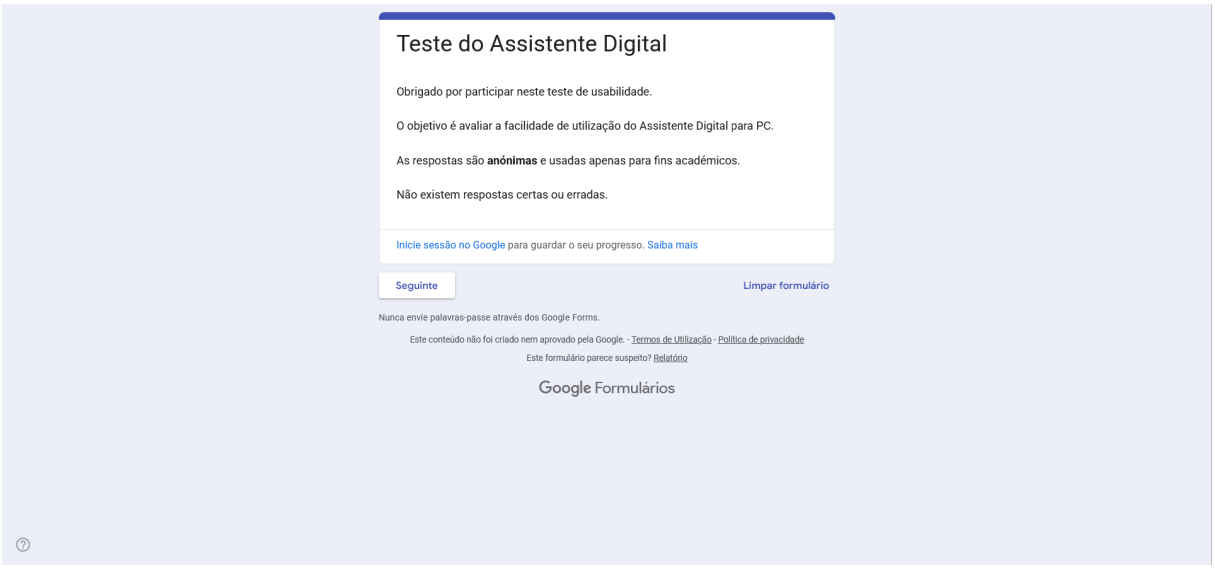

Figura A.1 Ecrã inicial do formulário de usabilidade

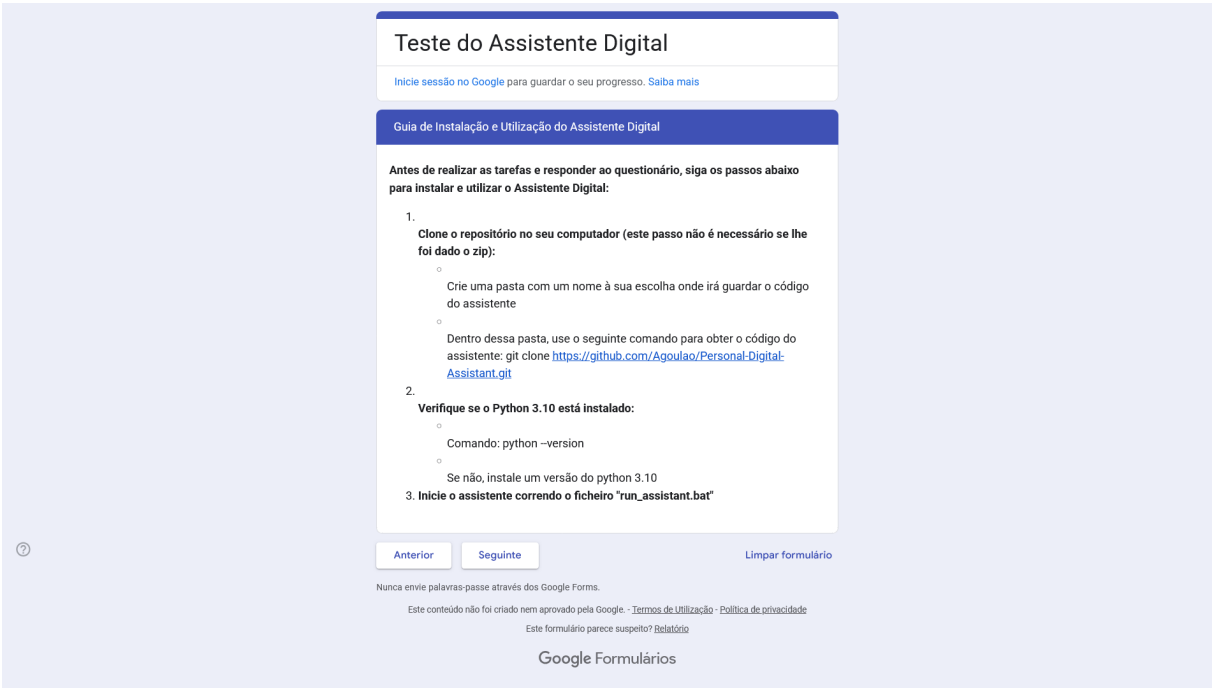

Figura A.2 Guia de instalação e utilização do Assistente Digital

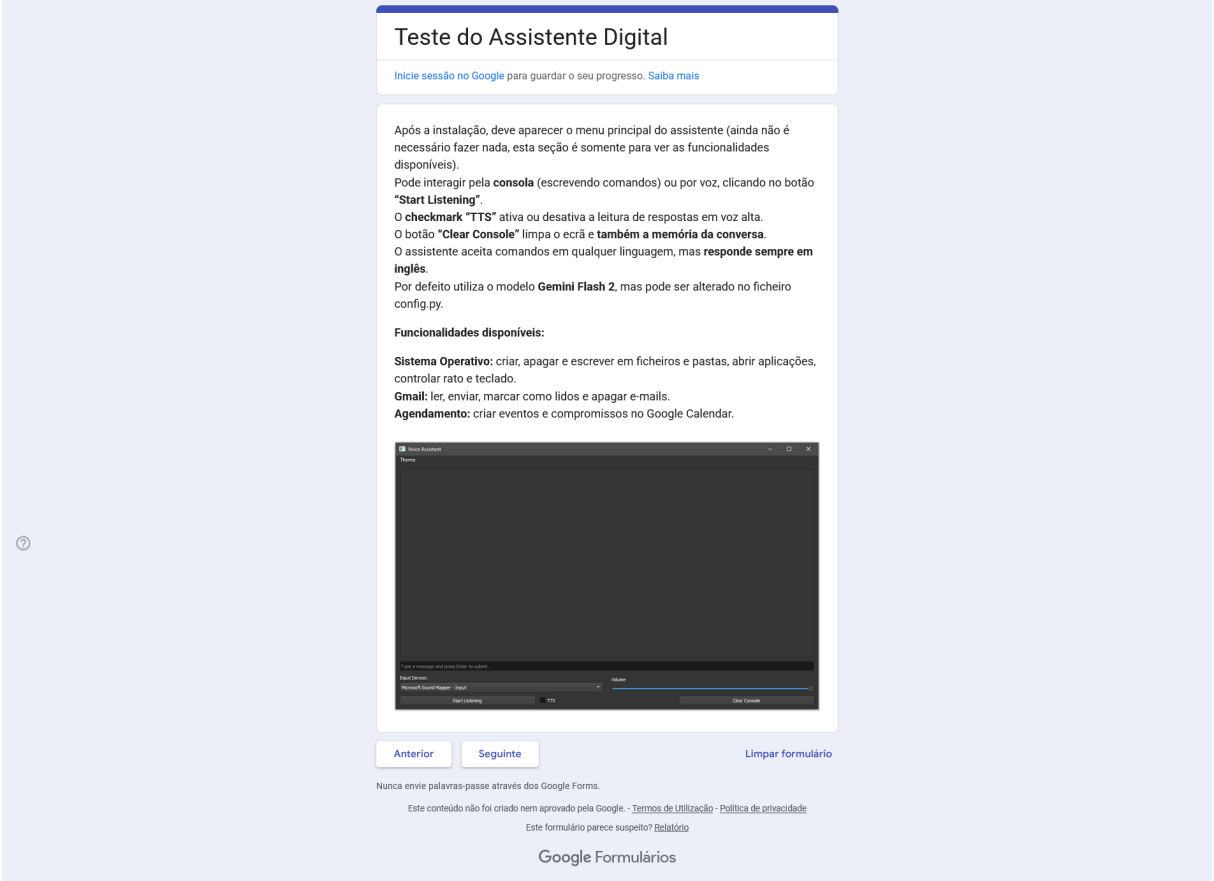

Figura A.3 Descrição das funcionalidades do Assistente Digital

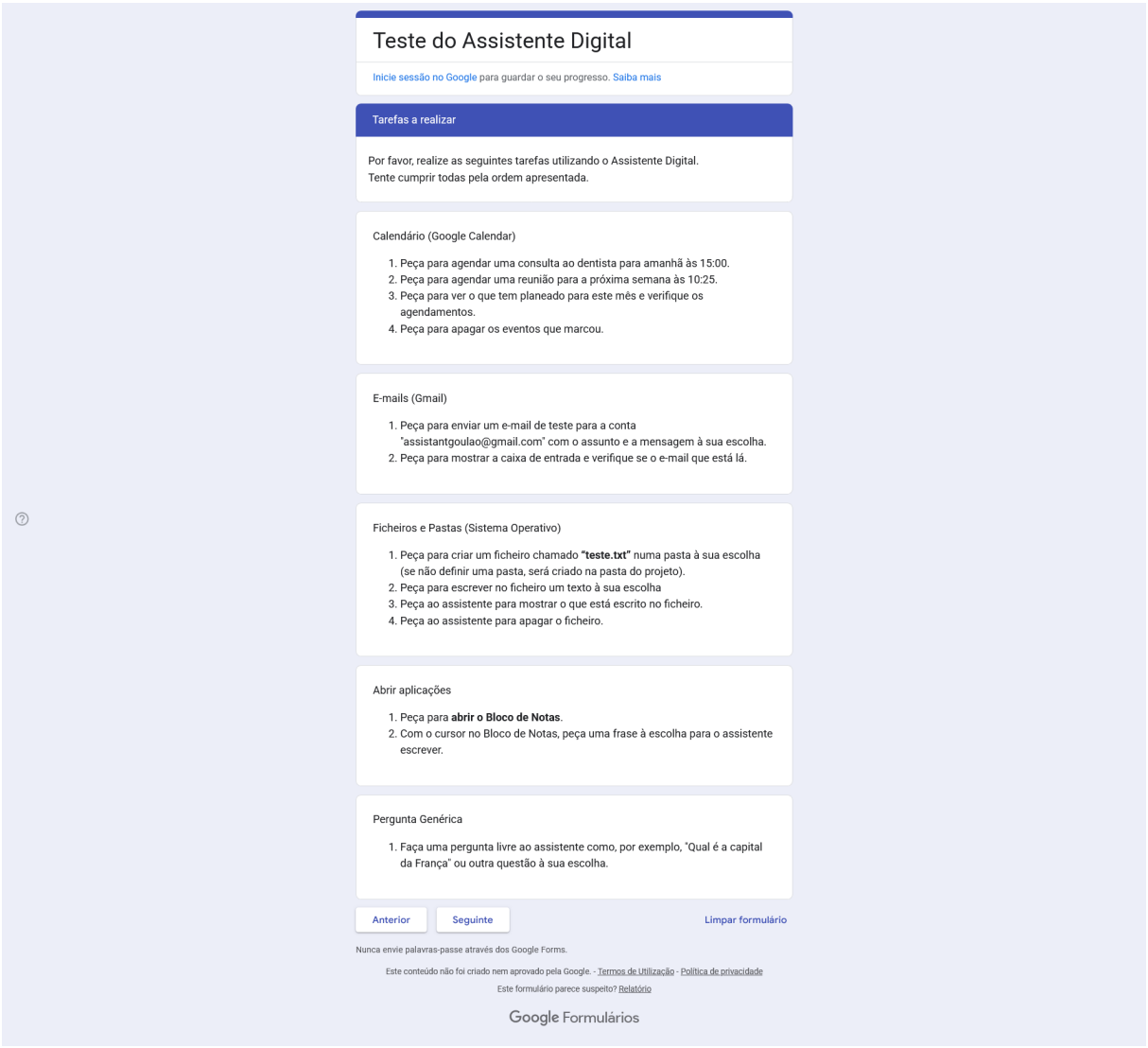

Figura A.4 Tarefas a realizar no formulário

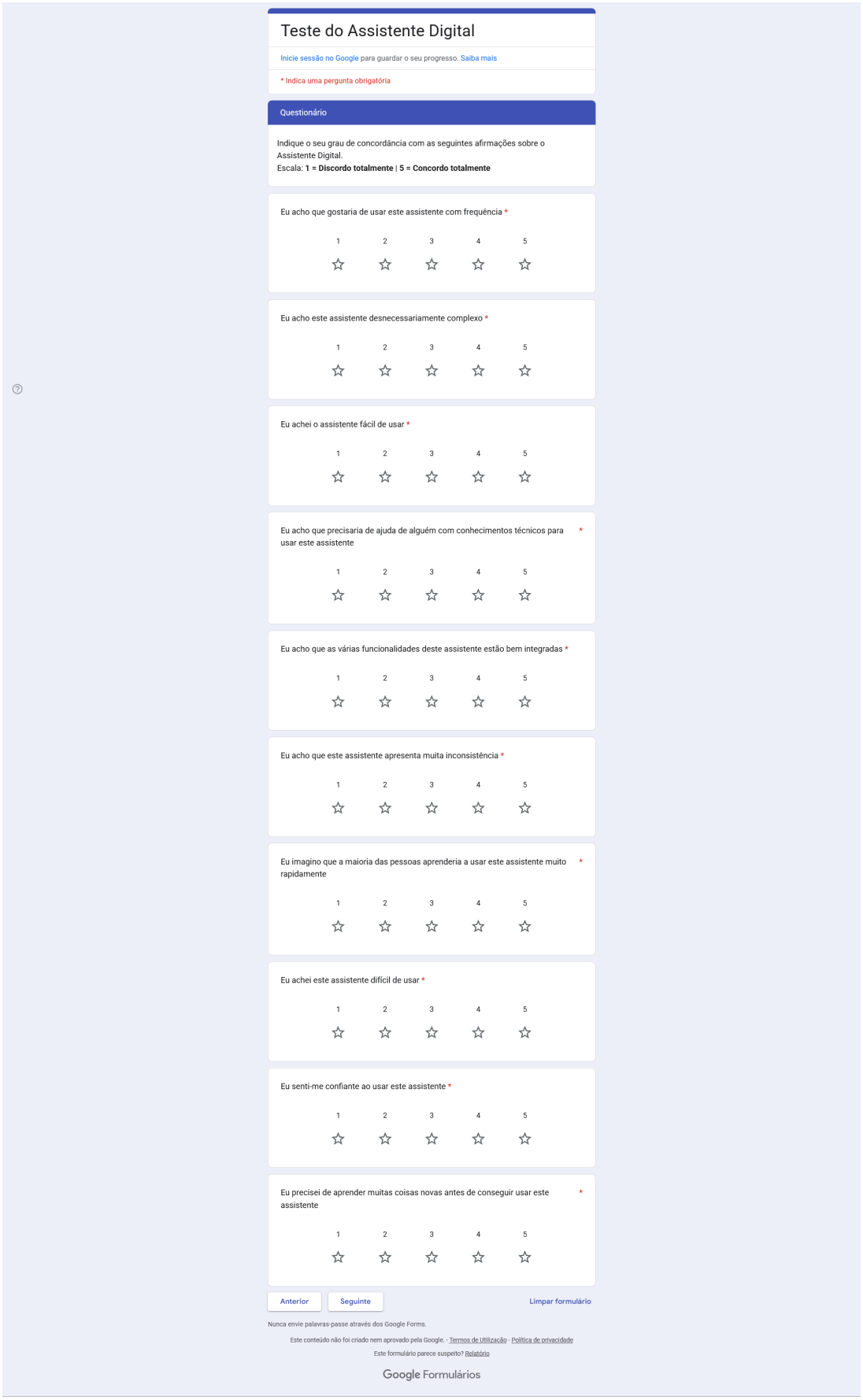

Figura A.5 Questionário de usabilidade

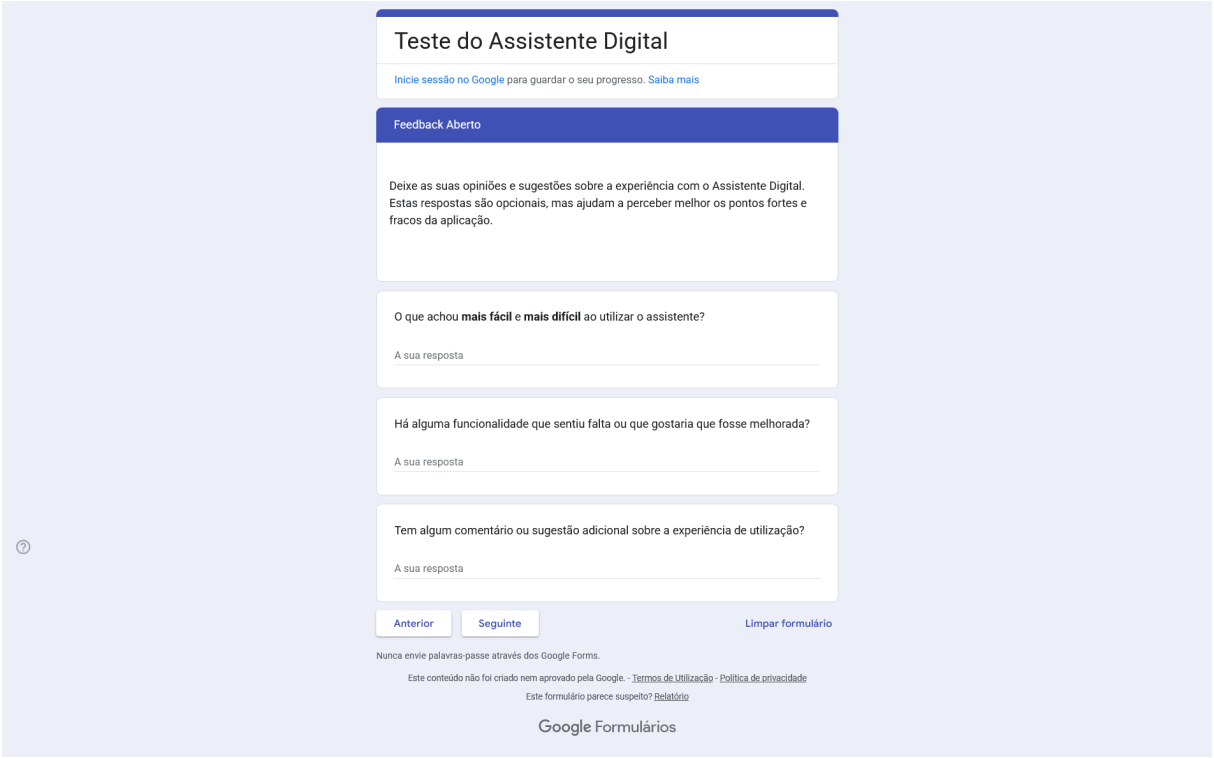

Figura A.6 Feedback aberto e finalização do formulário