A Modular Digital Assistant for Windows with Large Language Models

António Goulão[1]
a48714@alunos.isel.pt
Dinis Dias[1]
a46098@alunos.isel.pt
Artur Ferreira[1 2]
artur.ferreira@isel.pt
Nuno Leite[1]
nuno.leite@isel.pt

[1] ISEL, Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Portugal
[2] Instituto de Telecomunicações, Pólo de Lisboa, Portugal

Abstract

Personal digital assistants (PDA) have become mainstream on mobile and smart home devices, but open desktop-based solutions remain limited. In this paper, we present a modular Windows-based PDA that supports both speech and text commands through an Artificial Intelligence (AI) - driven architecture. Our system integrates Automatic Speech Recognition (ASR), a pluggable Large Language Model (LLM) for Natural Language Understanding (NLU), and Text-to-Speech (TTS) output, all orchestrated via custom interfaces to allow easy replacement of models or services. The assistant's plugin framework supports both local desktop automation and integration with online services namely Gmail, Google Calendar, and Weather. In the experimental evaluation, the prototype executed user commands with a success rate greater than 90%, demonstrating effective performance with an average response latency of only a few seconds for LLM-based queries.

1 Introduction

Advances in Artificial Intelligence (AI) have transformed how users interact with devices through PDA. Voice-driven PDA like Apple's Siri, Amazon's Alexa, and Google Assistant are now common on smartphones and smart speakers, streamlining tasks from scheduling to Web queries. However, these mainstream assistants operate within closed ecosystems with limited end-user customization. The desktop PC remains somewhat underexplored as a platform for such assistants, aside from recent proprietary efforts (e.g., Microsoft's Windows Copilot) that similarly lack openness. There is a growing need for customizable, open-source assistants that can run on personal computers, offering users greater control over functionality and data.

Recent breakthroughs in Natural Language Processing (NLP), particularly Large Language Models (LLM) [3, 5], enable more general and context-aware interactions. Models like GPT-3, GPT-4 and now GPT-5 have shown the ability to interpret diverse free-form requests and perform complex reasoning, suggesting that a desktop-based assistant could leverage such AI to handle tasks beyond hard-coded commands. In this paper, we propose a Windows-based personal digital assistant designed with modularity and extensibility in mind, aiming to fill this gap. The system leverages open-source tools for speech recognition and synthesis and integrates a pluggable LLM component to interpret user instructions and interface with various desktop and web services. This approach allows the assistant to be flexible and personalized, in contrast to one-size-fits-all commercial solutions.

2 State of the Art

Existing commercial assistants set the bar for ease of use but have notable drawbacks. Each one is locked to its vendor: Siri works only on Apple devices, Alexa on Amazon's cloud, and Google Assistant within Google's infrastructure. This creates privacy concerns, as audio data is routed through company servers, and prevents users from extending or customizing functionalities beyond what the provider supports. Microsoft's own Cortana assistant was discontinued in late 2023, and the new Windows Copilot (integrated into Windows 11) is a proprietary AI helper for tasks like document editing and Web search [4]. By contrast, the research and open-source communities are exploring more transparent, on-device designs. For instance, Microsoft recently unveiled Mu, a 330-million-parameter on-device LLM for controlling Windows settings by voice (e.g. "turn off Bluetooth") without Internet calls [6]. Similarly, community-driven projects such as Open Assistant, LM Studio, and Ollama enable local LLM execution with partial plugin support. These efforts reflect a trend: LLM are becoming available to run locally, enabling private, customizable assistants. However, a fully integrated desktop assistant that ties together input/output speech, an LLM core, and a plugin-based action framework is still an open challenge.

3 Proposed Solution

We propose a modular assistant architecture that decouples the core components of input processing, language understanding, and action execution. Figure 1 provides an overview of the system. The assistant accepts user commands via two input modes: speech, processed by the Automatic Speech Recognition (ASR) or typed text. ASR converts speech to text and Text-to-Speech (TTS) vocalizes responses. Both are abstracted, allowing different engines to be swapped without code changes.

At the system core, we have the LLM Adapter, which serves as the NLU brain. When a user's command (now in text form) is inputted, the LLM adapter forwards it to a configured LLM through a RESTful Application Programming Interface (API) call. This could be an online model (e.g., OpenAI GPT-4 or Google's Gemini) or a local model hosted on the machine. The adapter is designed to be pluggable, meaning that it can interface with any LLM backend given the appropriate API or Software Development Kit (SDK). We have assessed integrations with models of varying scale, from cloud-based services such as Gemini [2] to open-source models like LLaMA [1]. The LLM is prompted in such a way that the output is a structured intent in JavaScript Object Notation (JSON) format, stating the action to perform and the corresponding parameters. This approach allows complex reasoning and NLU to be transferred to existing AI models, while keeping the assistant's logic simple and modular.

Once an intent is extracted from the LLM's output, the assistant routes it to the appropriate module for execution. We implemented a set of pluggable modules that cover common personal assistant functions: file system navigation and application launching, email reading and composition (through Gmail API integration), calendar event querying and scheduling (via Google Calendar API), and fetching weather information from an online weather service. Each plugin is a self-contained Python component that declares the intentions it can handle and the actions it can perform (for instance, a calendar plugin can create or query events using Google's Calendar API). When the LLM returns a JSON intent like {"ac-

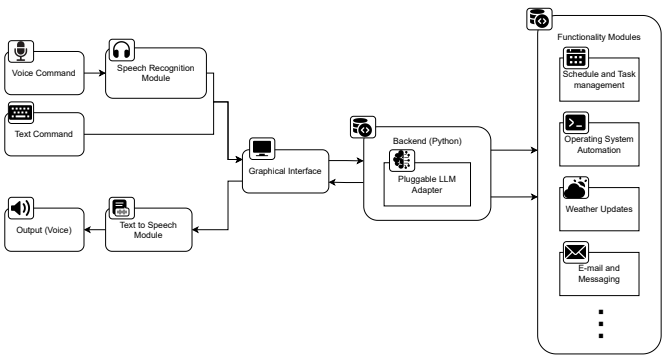

Figure 1: Block diagram of the modular Windows-based personal digital assistant.

tion": “add_event", “date": “...", “title": “..."}, the assistant dynamically sends it to the Calendar plugin, which carries out the request and returns a result or confirmation.

This plugin-based design makes the system highly extensible. New functionalities can be added by writing a new plugin that adheres to the assistant’s interface, without needing to modify the core assistant code or the LLM logic. The core coordinates the input, the LLM interpretation, and the plugin execution. Modularity allows to upgrade or replace components independently. For instance, one may replace the speech recognition module with a different library or switch the LLM from a cloud API to a local model for offline use. The entire prototype is implemented in Python on Windows, leveraging packages like `SpeechRecognition` and `pyttsx3` for speech and standard libraries (e.g., `subprocess`, `os`) for system commands, with Google’s client libraries for Gmail/Calendar integration. A simple Graphical User Interface (GUI), shown in Figure 2 was also built to allow users to see the transcribed text, the assistant’s response, and configure settings, although voice interaction is the primary mode of use.

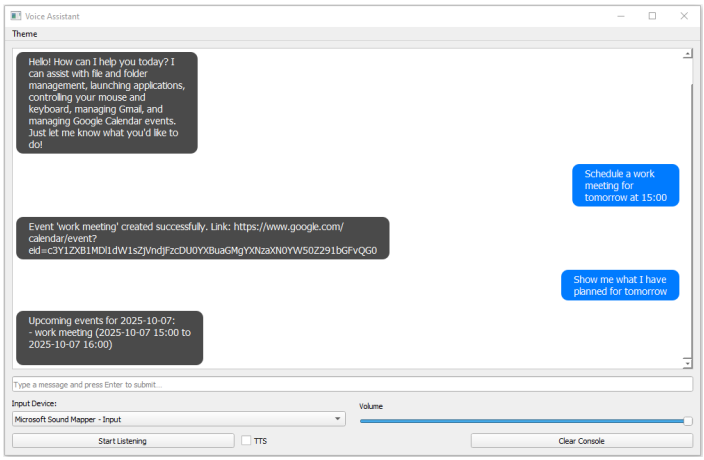

Figure 2: PDA GUI showing event scheduling.

4 Experimental Results

We evaluated the assistant’s performance on 22 command types that cover local file/folder operations, Google Calendar, Gmail, and weather queries. For each command, we created five semantically different expressions, yielding 110 test utterances per model. Table 1 illustrates this with the example of the create event command, showing the five paraphrases used to express the same intent. Each test ran through the full pipeline: LLM interpretation and plugin execution. A trial was marked successful only if the LLM produced a valid JSON intent and the corresponding plugin executed the action correctly. Table 2 summarizes the results.

Table 1: Five paraphrases used for the create event command.

#	Utterance
1	Schedule a dentist appointment for next Tuesday at 10 AM.
2	Create an event called ‘Team Brainstorm’ for November 30 th, 2025, from 2 PM to 3 PM.
3	Add an all-day event for my birthday on October 15 th.
4	Book a project review for tomorrow at 9:30 AM, lasting 30 minutes.
5	Set up a client call for July 30 th, 2025, at 2 PM, with description ‘Discuss contract details’.

The results show a clear gap: with a newer, large LLM (e.g. a 70B-parameter LLaMA model or Google’s Gemini), the assistant interprets and executes commands nearly flawlessly (95–100% of tasks). In contrast, a small model (1 to 8B parameters) often misinterprets commands or omits details, achieving around 50% success rate. This matches expectations from LLM benchmarks: larger models generalize better to novel instructions and rarely hallucinate unsupported actions, while smaller models are error-prone.

In terms of latency, response time varied significantly by provider and model size. The Gemini Flash series was the fastest, typically returning results in under one second on average (≈0.6–0.9 s). Smaller hosted models such as Llama-3.2-1B and Llama-3.3-8B averaged around 1.5–1.8 s, while the larger Llama-3-70B required roughly 2.2 s per query. These values include the end-to-end delay from the user finishing the command to the assistant producing its response. Overall, all latencies remained within interactive usability thresholds, but the gap highlights the impact of both model size and hosting infrastructure.

Table 2: Success rate by task per LLM. [L] = local, [R] = remote. L = Llama, G = Gemini.

Category	Command	L3.2-1B	L3-8B	L3-70B	G2.0 Flash	G2.5 Flash
File/Folder [L]	create folder	4/5	5/5	5/5	5/5	5/5
	create file	1/5	4/5	5/5	5/5	5/5
	write text	1/5	5/5	5/5	5/5	5/5
	append text	4/5	5/5	5/5	5/5	5/5
	read file	2/5	5/5	5/5	5/5	5/5
	delete file	4/5	5/5	5/5	5/5	5/5
	delete folder	4/5	4/5	5/5	5/5	5/5
	list dir	4/5	4/5	5/5	5/5	5/5
	rename file	3/5	5/5	4/5	5/5	5/5
	copy file	4/5	4/5	5/5	5/5	5/5
	move file	0/5	3/5	5/5	5/5	5/5
Calendar [R]	list events	1/5	5/5	5/5	5/5	5/5
	schedule event	3/5	5/5	5/5	5/5	5/5
	cancel event	0/5	5/5	5/5	5/5	5/5
Gmail [R]	list emails	2/5	4/5	5/5	5/5	5/5
	send email	2/5	3/5	5/5	5/5	5/5
	read email	3/5	4/5	5/5	5/5	5/5
	mark read	0/5	5/5	5/5	5/5	5/5
	delete email	0/5	5/5	5/5	5/5	5/5
Weather [R]	current	4/5	5/5	5/5	5/5	5/5
	forecast	3/5	5/5	5/5	5/5	5/5
	air quality	3/5	3/5	5/5	5/5	5/5
Totals		52/110	98/110	109/110	110/110	110/110

5 Conclusions

We have presented a modular, AI-powered personal digital assistant for the Windows desktop that explores an alternative to commercial assistants by emphasizing openness and extensibility. The proposed solution separates the concerns of speech input/output, language understanding, and task execution into interchangeable modules, with a LLM at its core to interpret diverse natural language commands. This design enabled us to integrate a variety of functionalities, such as file operations, email and calendar management, and weather querying, within a single coherent system. Our experimental evaluation has shown that our proposed Windows-based desktop assistant can reliably interpret and perform a wide range of common tasks with high accuracy and reasonable response times, validating the viability of the approach.

For future work, we plan to carry out structured user evaluations to assess usability, perceived latency, and the quality of interactions, complementing the technical results presented here. Another important direction is to broaden the plugin framework to include more advanced automation capabilities, such as multi-step workflows, proactive task suggestions, and integration with productivity tools beyond Gmail and Google Calendar. We also aim to enhance the assistant’s conversational abilities, allowing it to ask clarifying questions, maintain longer contexts, and adapt its behavior to user preferences. These extensions will help evolve the system from a reactive command executor into a more capable and customizable desktop assistant.

References

[1] A. Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[2] G. Comanici et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[3] T. Brown et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] J. MSV. Inside Microsoft Copilot: A look at the technology stack, 2023. `https://www.forbes.com/sites/janakirammsv/2023/05/26/`, accessed: 2025-10-08.

[5] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

[6] V. Pradeep. Introducing Mu language model and how it enabled the agent in windows settings, 2025. `https://blogs.windows.com/windowsexperience/2025/06/23/`, accessed: 2025-10-08.