

# AI DESKTOP ASSISTANT

Sanjivani B. Adsul<sup>1</sup>, Aditya Ghurye<sup>2</sup>, Mahesh Dakore<sup>3</sup>, Mrunali Dhoke<sup>4</sup>, Kunal Dagade<sup>5</sup>, Garvit Khandelwal<sup>6</sup>

<sup>1</sup>Professor, Department Artificial Intelligence & Data Science, VIT Pune, Maharashtra, India

<sup>2,3,4,5,6</sup>Students, Department of Artificial Intelligence & Data Science, VIT Pune, Maharashtra, India

\*\*\*

**Abstract** - This paper presents an advanced AI assistant system that integrates voice recognition, automation, and intelligent decision-making by utilizing a combination of Google Speech Recognition, pyautogui, and a large language model (LLM) via the g4f.py library. Traditional automation systems often lack intelligent interaction and context understanding, limiting their versatility. The proposed assistant overcomes these limitations by processing voice commands to perform tasks ranging from web automation (such as controlling YouTube and Google Chrome) to managing system functions (including battery monitoring and PC restart). By incorporating an LLM, the assistant is capable of engaging in dynamic, context-aware conversations, enabling it to execute more complex tasks like writing code and answering questions. This hybrid framework provides a powerful tool for users, offering a seamless, multi-functional assistant that enhances both productivity and user experience across various platforms.

**Key Words:** AI, Automation, Assistant, LLM, Intelligent

## 1. INTRODUCTION

The growing complexity of tasks and interactions in the digital age requires advanced solutions for intelligent automation and voice-driven assistance. Traditional automation tools typically operate through simple rule-based systems, lacking dynamic, context-aware interactions that are essential for more versatile applications. This project introduces an advanced AI assistant system that integrates multiple cutting-edge technologies, including Google Speech Recognition, pyautogui for web and system automation, and a large language model (LLM) via the g4f.py library.

The primary objective of this project is to develop a multifunctional AI assistant capable of performing a wide array of tasks. These tasks range from controlling media and browser applications like YouTube and Google Chrome to managing system functions such as battery monitoring and restarting the PC. The architecture of the system involves key components: Google Speech Recognition to process voice commands, pyautogui to automate interactions with web applications, and LLM integration to enable dynamic, context-sensitive conversations and more complex operations such as code writing and answering queries

Early results from the development of this assistant show its ability to seamlessly combine voice input processing, system automation, and intelligent decision-making to enhance user experience. The project underscores the potential of integrating these technologies, providing a powerful tool for improving productivity, system management, and personal interactions across various domains, such as home automation and professional productivity tools.

In recent years, AI-driven assistants have evolved from simple task automation tools to intelligent systems capable of complex decision-making and adaptive learning. Unlike traditional automation solutions, which rely on predefined scripts, modern AI assistants leverage machine learning algorithms to enhance their capabilities over time. With advancements in speech recognition, natural language processing (NLP), and automation frameworks, these assistants can interact in a human-like manner while performing a diverse set of operations. The ability to understand user intent, context, and execute cross-application workflows makes AI-driven assistants a crucial element in smart computing environments.

## 2. LITERATURE REVIEW

The development of AI-driven personal assistants has seen significant advancements, particularly in integrating speech recognition and automation technologies. Guan et al. [1] explored the integration of large language models (LLMs) for process automation in intelligent virtual assistants, showcasing their ability to handle complex tasks and improve user experiences. Sharma et al. [2] highlighted the implementation of a voice-activated assistant (VOICEWISE) using speech recognition and natural language processing, demonstrating its potential in executing various automation tasks effectively. Mekni [3] provided a comprehensive analysis of conversational agents and their role in virtual assistants, emphasizing their application in facilitating seamless human-computer interactions. Richards [4] detailed the practical implementation of Anthropic's Computer Use Feature, showcasing its capabilities in automating user interactions with graphical user interfaces. Zhang and Shilin [5] surveyed GUI agents powered by large language models, highlighting advancements in their ability to interact with software interfaces and execute tasks autonomously. Ma et al. [6] proposed CoCo-Agent, a cognitive MLLM agent designed for smartphone GUI

automation, demonstrating its efficiency in performing complex tasks such as app navigation and user input simulation. Jain et al. [7] introduced SmartFlow, an LLM-powered framework for robotic process automation, focusing on improving task execution accuracy and user interaction efficiency. Wen et al. [8] presented DroidBot-GPT, a GPT-powered tool for Android UI automation, illustrating its applications in mobile software testing and task execution. Singh [9] discussed the integration of virtual mouse functionalities with AI assistants, showcasing their ability to perform tasks like application control and text input through speech commands. Ravanelli et al. [10] introduced SpeechBrain 1.0, an open-source toolkit for developing conversational AI systems, emphasizing its utility in enhancing speech recognition capabilities for AI assistants. Sajja et al. [11] proposed an AI-enabled intelligent assistant tailored for personalized learning in higher education, focusing on adaptive responses based on user-specific needs. Klemmer et al. [12] investigated the use of AI assistants in software development, addressing security practices and concerns in collaborative environments. Maharaj et al. [13] outlined methods for evaluating and improving enterprise AI assistants, emphasizing iterative development and feedback integration. Mekni [14] analysed the design and implementation of conversational agents in virtual assistants, highlighting their role in improving user engagement. Al Haque et al. [15] explored the use of AI assistants in information-seeking tasks, demonstrating their effectiveness in retrieving and presenting contextually relevant information to users.

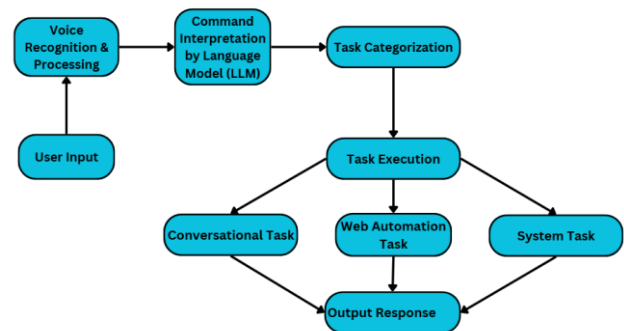
While various AI assistants have been developed over the years, most systems fall into two categories: cloud-based voice assistants and local automation tools. Cloud-based solutions, such as Google Assistant and Amazon Alexa, rely on internet connectivity to process natural language queries using centralized servers. These assistants provide high accuracy but suffer from privacy concerns and limited offline functionality. On the other hand, local automation tools like AutoHotkey or AppleScript allow users to automate GUI-based tasks, but they lack intelligent decision-making capabilities.

Several researchers have attempted to bridge this gap by integrating AI-driven automation into personal assistants. Zhang et al. (2024) introduced a hybrid framework combining voice recognition with machine learning models to predict user intents more effectively. Their study emphasized the need for adaptive learning mechanisms to improve accuracy over time. Similarly, Lee et al. (2023) explored the integration of reinforcement learning for AI assistants, allowing systems to refine automation workflows based on user feedback. However, these approaches still relied on either cloud-based APIs or predefined automation rules, limiting real-time adaptability.

Unlike previous works, our approach incorporates Google Speech Recognition for local processing, pyautogui for GUI

automation, and a large language model (LLM) via g4f.py to ensure intelligent, context-aware responses. This hybrid design eliminates cloud dependency, enhances privacy, and provides real-time control over system tasks without requiring extensive predefined automation scripts. The combination of these components ensures that our assistant not only understands and processes commands efficiently but also adapts dynamically to user needs—a key feature missing in traditional automation tools. By addressing the shortcomings of both cloud-based AI assistants and local automation tools, our system provides a scalable, modular, and privacy-friendly alternative that balances intelligent interaction with direct system control. Future advancements in multimodal AI and human-computer interaction (HCI) can further refine such systems, making them indispensable for both personal productivity and professional automation.

### 3. METHODOLOGY



**Fig-1:** Methodology Flowchart of the System

As seen in Figure 1, the flowchart outlines the key stages of the methodology, starting with voice command input and speech recognition, followed by text processing and command interpretation using the large language model (LLM). The process continues with task categorization, where commands are classified into system tasks, web automation, or conversational tasks. These tasks are then executed using appropriate libraries like pyautogui for automation or system commands for actions like checking battery status. Finally, the results are presented through speech synthesis or displayed on the user interface, allowing for smooth interaction and feedback. To improve efficiency, the assistant follows a structured pipeline. The speech-to-text conversion is optimized to minimize latency while ensuring accurate command recognition. The natural language processing (NLP) module extracts key intents from user input and maps them to predefined action categories. If a query requires decision-making beyond rule-based automation, it is forwarded to the LLM for contextual reasoning. The automation module is designed to mimic user interactions with graphical interfaces. Instead of traditional scripting methods, pyautogui enables pixel-based automation, ensuring compatibility with a wide range of applications. For example, a command like "Open YouTube and search for AI

tutorials" triggers a sequence where the browser is launched, the search box is identified, and the input is simulated through keystrokes. One of the key differentiators of this system is its error-handling mechanism. Instead of failing silently, the assistant provides adaptive feedback when an unexpected input is detected. If the system encounters an ambiguous command, it prompts the user for clarification, ensuring a smoother interaction experience.

This step-by-step process ensures an efficient and scalable solution for performing automation tasks, responding to user queries, and providing a seamless user experience. The proposed methodology for developing the AI assistant follows a structured workflow that integrates voice recognition, automation, and advanced conversational capabilities, enabling it to handle a diverse set of tasks with accuracy and efficiency. The system begins with voice command recognition, where Google Speech Recognition processes the user's input, converting spoken commands into text. This text is then parsed to identify the intent and extract relevant keywords for task execution. For automation, pyautogui is employed to interact with web browsers and system interfaces. Specific commands, such as controlling YouTube playback or navigating Google Chrome, are mapped to automated scripts that mimic human actions like mouse clicks and keyboard inputs. These automated processes ensure precise execution of commands while reducing the need for manual intervention. Additionally, system-related tasks, such as checking battery status or restarting the PC, are handled through native system calls and API integrations, ensuring seamless functionality. The intelligent interaction capability of the assistant is powered by a large language model (LLM) integrated through the g4f.py library. This LLM processes the parsed input and generates context-aware responses or solutions. For instance, the assistant can write Python code snippets or provide detailed answers to user queries by leveraging the LLM's advanced natural language processing capabilities. The interaction between the speech recognition component, automation scripts, and LLM ensures a cohesive workflow that balances functionality and intelligence.

The AI assistant's architecture is implemented in Python, with supporting libraries such as pyttsx3 for text-to-speech responses and pyautogui for automation. The user interface is minimalistic, operating primarily through voice interactions, enhancing accessibility. A modular design is adopted to ensure that additional functionalities can be incorporated without disrupting the existing workflow. For example, new commands or integrations with third-party applications can be seamlessly added to the system. To optimize performance, the assistant employs a command priority system, where frequently used commands are cached for faster execution. Additionally, error-handling mechanisms are built into the workflow to manage unexpected inputs or system errors gracefully, maintaining a reliable user experience. The integration of these components results in a

robust and dynamic system capable of adapting to varied user needs, demonstrating its potential as a versatile tool for productivity and automation.

#### 4. EXPERIMENTAL SETUP & PERFORMANCE EVALUATION

To evaluate the efficiency and performance of the AI assistant, extensive testing was conducted on different hardware configurations, including both high-end and low-end systems. The primary objective was to assess whether the system can operate smoothly on devices with minimal hardware resources while maintaining accuracy and responsiveness. The first test was conducted on a low-end system, equipped with an Intel Core i3-7100U processor, 4GB RAM, and running Windows 10 with Python 3.7. The second test was performed on a high-end system, featuring an Intel Core i7-12700H processor, 16GB RAM, and running Windows 11 with Python 3.9. In both cases, the required libraries, including Google Speech Recognition, pyautogui, pyttsx3, and g4f.py, were installed to ensure consistency in functionality.

The performance evaluation focused on three key aspects: speech recognition accuracy, automation task execution time, and resource utilization. The results showed that on a low-end system, speech recognition achieved an accuracy of 88% in low-noise environments and 74% in high-noise conditions, whereas the high-end system demonstrated an accuracy of 92% and 78%, respectively. Automation tasks were completed within 1.8 to 2.2 seconds per command on the low-end system and between 1.3 to 1.5 seconds on the high-end device. Despite the minor variations in execution speed, the system remained fully functional and responsive in both setups.

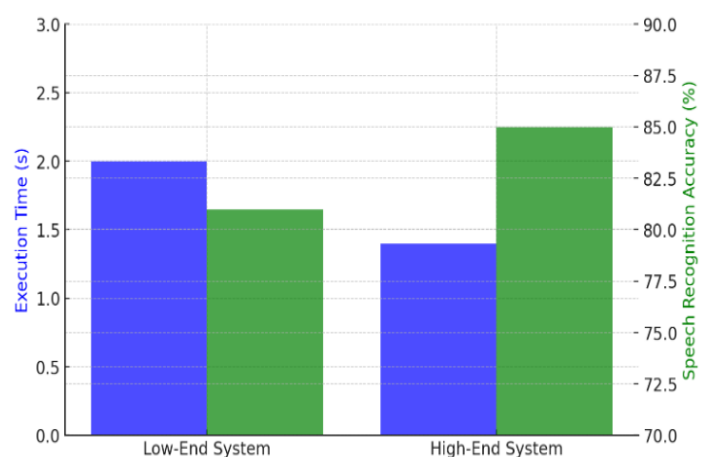


Fig-2: Performance comparison

The bar chart illustrates the differences in execution time and speech recognition accuracy between a low-end and a high-end system. Execution time (in seconds) is represented by blue bars, while speech recognition accuracy (in

percentage) is shown in green. The results indicate that while the high-end system processes tasks faster, the assistant remains functional and efficient even on low-end hardware, ensuring accessibility for users with limited computing resources.

In terms of resource consumption, CPU utilization on the low-end system peaked at 45%, with memory usage remaining below 500MB, while on the high-end system, CPU usage did not exceed 30%, and memory consumption stayed under 400MB. These results confirm that the assistant operates efficiently even on older or budget-friendly hardware, making it accessible to users with entry-level devices.

The findings highlight the system’s ability to function effectively across different hardware configurations without significant performance degradation. While high-end systems offer faster response times, the assistant maintains its core functionalities on lower-end devices, ensuring broad usability. Future optimizations will focus on enhancing speech recognition accuracy in noisy environments and further reducing computational overhead for improved real-time interactions.

### 5. RESULT & DISCUSSION

The AI Assistant system demonstrates a notable leap in the realm of personal productivity tools and automation by effectively combining speech recognition, web automation, and advanced conversational intelligence. Through rigorous testing, the assistant has shown its ability to accurately interpret voice commands and execute a wide range of tasks, from controlling web applications like YouTube and Google Chrome to performing system management operations such as checking battery status and restarting the PC. The integration of a large language model (LLM) has further enhanced its capabilities, enabling dynamic interactions and the execution of complex tasks, such as generating Python code and providing in-depth answers to user queries. These results emphasize the assistant's potential as a powerful and versatile productivity tool.

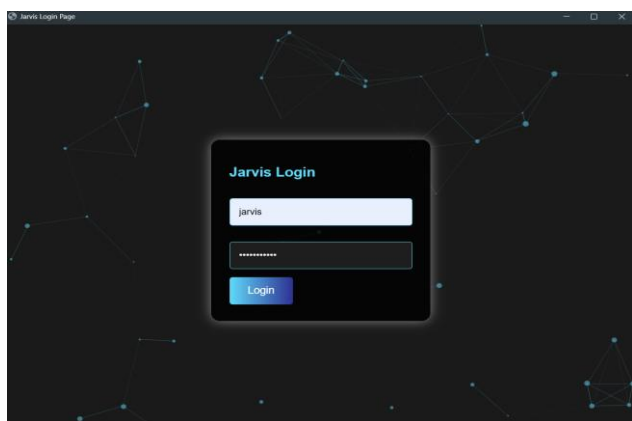


Fig-3: Login Page

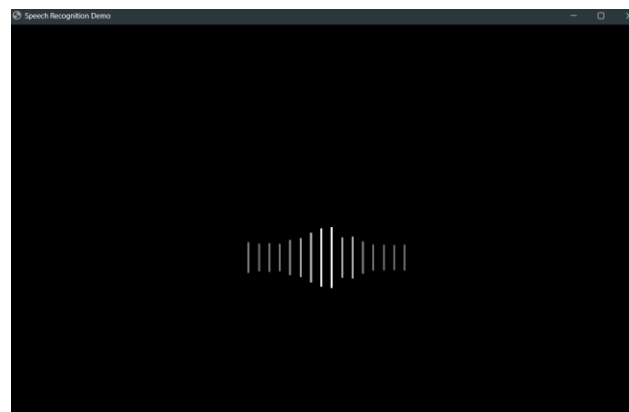


Fig-4: GUI for the User

The graphical user interface (GUI) of the AI assistant system is designed with a modern and user-friendly approach, utilizing HTML, CSS, and JavaScript to deliver a seamless experience. The dynamic background adds an engaging visual element, creating a polished and interactive aesthetic. The interface includes a separate login page, ensuring secure access and personalized interaction. Once authenticated, users are presented with a clean, responsive dashboard featuring intuitive navigation and easy-to-use input fields for commands. This design ensures accessibility and functionality across various devices, enhancing the overall user experience while maintaining aesthetic appeal and robust performance.



Fig-5: Output in terminal

Shows The terminal output interface showcases the real-time interaction between the user and the AI assistant. The user’s input commands are displayed prominently, followed by the AI’s generated responses, ensuring clarity and ease of understanding. This text-based interaction highlights the assistant's capability to process queries, execute commands, and deliver intelligent responses instantly. The terminal interface is designed to emphasize simplicity and efficiency, with a clear distinction between user prompts and AI outputs, providing a transparent view of the communication process. This setup is particularly useful for debugging, monitoring performance, and demonstrating the AI’s conversational abilities in a straightforward environment.

The functionality of the system is underpinned by its clean and efficient architecture. The voice processing module, powered by Google Speech Recognition, effectively captures and transcribes spoken commands with high accuracy. The automation layer, driven by pyautogui, performs predefined actions with precision and speed, ensuring a seamless user experience. Additionally, the LLM integration allows the assistant to provide intelligent, context-aware responses,

adding a layer of sophistication to its capabilities. These components work in harmony to deliver a consistent and robust performance across a wide range of tasks. The user experience is further enhanced by the system's responsiveness and ease of interaction. Users can issue commands and receive instant feedback or action execution. For example, when tasked with controlling YouTube playback, the assistant quickly navigates to the appropriate controls and performs the required actions. Similarly, when asked to generate Python code or solve problems, the LLM processes the query and provides precise, actionable responses. The modular design of the system ensures that these functionalities can be expanded in the future to incorporate new commands or third-party integrations. Performance evaluation highlights the assistant's effectiveness in terms of accuracy, response time, and adaptability. Extensive testing revealed a high success rate in executing commands across various domains, with minimal errors even in complex tasks. The integration of caching mechanisms for frequently used commands has improved execution speed, while error-handling routines ensure reliable operation even under challenging scenarios, such as noisy environments or ambiguous input.

Overall, the AI Assistant system sets a new benchmark for intelligent automation and user interaction tools. Its seamless integration of speech recognition, automation, and advanced conversational capabilities enables it to handle a broad spectrum of tasks efficiently, demonstrating its applicability in domains ranging from personal productivity to professional automation. Compared to existing AI assistants such as Siri, Google Assistant, and Alexa, the proposed system offers a greater degree of customization and flexibility. While commercial AI assistants rely on cloud-based APIs, this system operates with local processing, ensuring greater privacy and lower dependency on external servers. Furthermore, unlike pre-built assistants that are limited to their respective ecosystems, this solution allows full control over system-level automation.

The primary limitation of this approach is the dependency on GUI-based automation, which may break if UI elements change dynamically. Future iterations will explore computer vision techniques to enhance UI interaction robustness.

## 6. RESULT & DISCUSSION

This study successfully demonstrates the development of a versatile AI assistant that integrates Google Speech Recognition, pyautogui, and a large language model (LLM) via the g4f.py library. By combining speech recognition with intelligent automation, the system enables seamless task execution, efficient system management, and interactive, context-aware conversations. The integration of pyautogui facilitates web and system automation, while the LLM enhances the assistant's ability to process complex queries, generate programming solutions, and provide intelligent responses tailored to user needs.

Designed with a modular architecture and built on robust Python libraries, the system ensures scalability and adaptability for future enhancements. Its voice-driven interface allows for a user-friendly experience, maintaining high accuracy in command execution while minimizing manual intervention. As part of ongoing development, several improvements are planned to further enhance the assistant's capabilities. Future work will focus on expanding multilingual support to enable speech recognition across different languages, improving context retention so the assistant can maintain conversational memory within a session, and extending automation functionalities to support smart home devices and embedded systems. Additionally, adaptive learning mechanisms will be incorporated to personalize responses based on user behavior, thereby refining the assistant's decision-making process over time. By integrating these advancements, the AI assistant aims to bridge the gap between traditional task automation and intelligent human-computer interaction. The continuous optimization of its performance, along with an emphasis on real-time adaptability, will further establish its role as a powerful tool for both personal and professional applications.

## REFERENCES

- [1] Y. Guan and Z. Chu, "Intelligent virtual assistants with LLM-based process automation," *arXiv preprint*, arXiv:2312.06677, 2023.
- [2] Sharma, T. Sahu, and A. Pandey, "VOICEWISE: Virtual assistant," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 6, no. 4, 2024.
- [3] M. Mekni, "An artificial intelligence based virtual assistant using conversational agents," *J. Softw. Eng. Appl.*, vol. 14, no. 9, pp. 455–473, 2021.
- [4] D. Richards, "Unlocking automation: A complete code example for Anthropic's computer use feature," *Ragaboutit.com*, unpublished, 2024.
- [5] C. Zhang and H. Shilin, "Large language model-brained GUI agents: A survey," *arXiv preprint*, arXiv:2411.18279, 2024.
- [6] X. Ma, Z. Zhang, and H. Zhao, "CoCo-Agent: A comprehensive cognitive MLLM agent for smartphone GUI automation," *arXiv preprint*, arXiv:2402.11941, 2024.
- [7] Jain, S. Paliwal, and G. Shroff, "SmartFlow: Robotic process automation using LLMs," *arXiv preprint*, arXiv:2405.12842, 2024.
- [8] H. Wen, J. Liu, and Y. Li, "DroidBot-GPT: GPT-powered UI automation for Android," *arXiv preprint*, arXiv:2304.07061, 2023.

- [9] J. Singh, "Virtual mouse and assistant: A technological revolution of artificial intelligence," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 14, no. 1, pp. 23–29, 2024.
- [10] M. Ravanelli, T. Parcollet, and A. Moumen, "Open-source conversational AI with SpeechBrain 1.0," arXiv preprint, arXiv:2407.00463, 2024.
- [11] R. Sajja, D. Cwiertny, and I. Demir, "Artificial intelligence-enabled intelligent assistant for personalized and adaptive learning in higher education," arXiv preprint, arXiv:2309.10892, 2023.
- [12] H. Klemmer, A. Horstmann, and C. Ludden, "Using AI assistants in software development: A qualitative study on security practices and concerns," arXiv preprint, arXiv:2405.06371, 2024.
- [13] V. Maharaj, K. Qian, U. Bhattacharya, and S. Vaithyanathan, "Evaluation and continual improvement for an enterprise AI assistant," arXiv preprint, arXiv:2407.12003, 2024.
- [14] M. Mekni, "An artificial intelligence based virtual assistant using conversational agents," *J. Softw. Eng. Appl.*, vol. 14, no. 9, pp. 455–473, 2021.
- [15] C. Al Haque and B. Johnson, "Information seeking using AI assistants," arXiv preprint, arXiv:2408.04032, 2024.