# Development of A Voice-Based Virtual Assistant For Windows

**Navya S[1], Pavana N[2], Sanjana H Rao[3], Sanvi U[4], Rajashekar M B[5]**
[1, 2, 3, 4] Dept of CSE
[5]Associate Professor, Dept of CSE
[1, 2, 3, 4, 5] GSSSIETW, Mysuru, India

**Abstract-** *For this project, we're building a virtual assistant designed to make daily tasks easier and more interactive. The idea is that you can talk to it, just speak commands, and it'll handle things like setting reminders, opening websites, and running system functions. If it doesn't fully understand a question or command, it uses AI to figure it out and still provide a helpful answer.*

*One of the key features we focused on is making it hands-free. You simply say a wake word, and it's ready to go—no need to click or type anything. It's also built to handle mistakes smoothly, so if something goes wrong, it won't crash the whole system. We wanted to keep things simple and clean, so there's an easy-to-use interface for managing reminders and interacting with the assistant.*

*On the technical side, we've made sure it's fast and reliable. It stores data securely, can scale up if we need to add more features, and works across different platforms. We also designed it to be adaptable to future needs. We didn't want it to just work now, but to be something that can evolve as new technologies and user requirements emerge. Our goal is to create something that's easy to use, dependable, and smart enough to grow as users' needs change.*

***Keywords***- Voice interaction, Reminder management, AI-driven responses

## I. INTRODUCTION

These days, with how fast AI and voice tech are developing, it's becoming easier to create tools that can actually help with everyday tasks. For this project, we decided to build a virtual assistant that you can talk to — kind of like a smart helper for your computer. It can do things like open websites, manage reminders, carry out system commands, and even chat with you a bit.

The whole thing is made using Python, mainly because of how many useful libraries it has. We used speech_recognition so it can listen to voice commands, pyttsx3 so it can reply with speech, and CustomTkinter to give it a cleaner, more modern interface. You can either say "Hey Windows" to start talking to it, or just use the buttons on the screen if you prefer clicking around.

What makes this assistant more than just a basic command runner is that it also uses some AI. So, if you ask something it doesn't recognize directly, it can still try to figure out what you mean and give a helpful response. It also saves reminders and other user data so that nothing is lost when you close the program.

The project is built with future improvements in mind — we kept it flexible so we can add more features later or even make it work on other platforms. Overall, it's a small but solid example of how voice assistants can be used to make computer tasks feel a bit easier and more natural.

## II. LITERATURE SURVEY

Doing a proper literature survey is one of the most important early steps in any software development project. Before starting development, it's essential for us to consider key factors like how much time the project might take, the overall budget, and the available technical resources. Once those basics are clear, the next decision usually comes down to picking the right operating system and programming language based on what best suits the project's goals.

In the context of this projecta voice-based virtual assistantwe explored a range of studies and papers to better understand the current landscape of technologies and their practical applications.

[1] **A 2021 review by Kumar, Hossain, and others** provides a solid overview of how popular voice assistants like Siri, Alexa, and Google Assistant work behind the scenes. The paper goes into detail about key technologies such as Automatic Speech Recognition (ASR), Natural Language Processing (NLP), and Text-to-Speech (TTS), which all work together to allow these systems to understand human speech and respond appropriately. They also cover real-world applications, especially in areas like healthcare, education, and

smart homes. Some of the major challenges pointed out include dealing with different accents, ensuring privacy, and handling a wide range of languages. The study emphasizes the need for more context-aware systems and better energy efficiency to make these tools more effective and widely usable.

[2] **Another 2021 paper titled Voice Assistants for Education**: A Case Study takes a closer look at how assistants like Google Assistant, Alexa, and Cortana are being used in educational settings. It talks about how these tools help both teachers and students—for example, by running quizzes, offering mock tests, managing schedules, and giving quick access to study materials. They've even been helpful in areas like language and music learning. That said, the paper also points out some issues, like how voice assistants sometimes misinterpret what's being asked, and concerns about accessibility for people who aren't fluent in English or those with limited access to tech.

[3] **In The Role of Natural Language Processing in Voice Assistants, Arora and Verma (2021)** dig deeper into how NLP is used to help these assistants actually understand what users are saying. They explain how machine learning helps improve accuracy over time, with some systems reaching over 95% recognition accuracy. The paper also covers some of the newer technologies being used, like BERT and GPT, which improve understanding even further—especially when it comes to dealing with multiple languages or emotional tones in speech. Ethical concerns like data privacy and bias in algorithms are also highlighted, showing that technical progress needs to be matched with responsible use of data.

[4] **Accessibility is a big topic in tech, and a 2022 paper by Gupta and Mahajan** titled Accessibility and Inclusivity in Voice Assistant Design really highlights its importance. The authors discuss how voice assistants can help people with disabilities—like enabling hands-free control for people with mobility challenges or helping visually impaired users with information access and communication tools like speech-to-text. Despite their potential, there are still gaps—such as inconsistent speech recognition and lack of unified standards across devices. The paper calls for better AI models and stricter data protection to make these tools more inclusive and trustworthy.

Based on the findings from these studies, it's clear that building a voice-based assistant for desktops is both technically feasible and increasingly relevant, especially with how much progress has been made in NLP and speech technologies. Choosing Python as the development language made the most sense for this project, mainly because it supports a wide range of libraries related to voice and AI. Windows was selected as the platform due to its popularity and ease of integration with desktop applications.

## III. METHODOLOGY

For our group project, we've designed a modular voice assistant that combines face authentication and hot word detection, all while running efficiently using parallel processing. The system has two primary components: main.py and run.py.The main.py script handles the core functionalities of the assistant. It sets up the graphical user interface (GUI) using the eel library and launches a local web interface (index.html) for the user to interact with. When authentication is successful, the assistant greets the user and grants access, while a failure in authentication prompts a notification to the user. On the other hand, the run.py script is responsible for managing the system processes. We've used Python's multiprocessing library to run the voice assistant (startJarvis()) and the hot word detection process (listenHotword()) concurrently. This allows the system to listen for activation commands while still performing other tasks at the same time.
To ensure everything runs smoothly, we've implemented process lifecycle management. T (index.html) for user interaction. Successful authentication triggers UI feedback, greets the user, and provides access to the assistant, while authentication failure notifies the user accordingly. The run.py script manages system processes, using Python's multiprocessing library to concurrently run the of Voice-Based Virtual Assistant for Windows assistant (startJarvis()) and the hot word detection process (listenHotword()), implemented via the engine.features.hotword module. This parallel execution ensures the system can listen for activation commands while performing other tasks. The code also handles processlifecycle management, using join() and terminate() to ensure proper cleanup, especially terminating the hotword detection process if the assistant stops. This integrated approach not only automates and streamlines attendance tracking but also empowers educators and administrators to make informed decisions through comprehensive data analytics.

### Development of Voice-Based Virtual Assistant for Windows

The diagram illustrates the operational workflow of a virtual voice based assistant for windows recognition attendance system integrated with a National Language Processing.

- **Actors**

User: The primary actor who interacts with the voice assistant. The user initiates commands and requests functionalities.

• **Voice Assistant System:** This represents the entire system that processes user commands and provides responses. It encapsulates various functionalities that can be accessed by the user.

• **Use Cases:** The diagram includes several key use cases that represent specific actions or functionalities available to the user.

• **Activate Assistant:** This use case represents the action of activating the voice assistant, typically by saying a wake word Give Voice Command: After activation, the assistant can process various commands, such as opening programs,
Incorporating these key elements, we’ve created a voice assistant that’s not only reliable but also flexible enough to adapt to new features in the future. Our goal is to make it a valuable tool for streamlining tasks in an intuitive, user-friendly way.

## IV. SNAPSHOTS

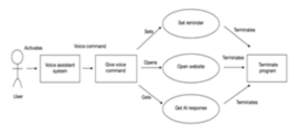
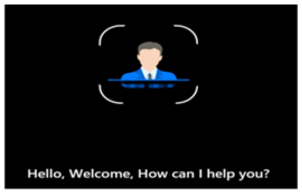
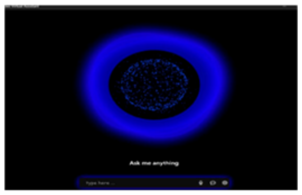


**Figure 1: Voice based virtual assistant use case diagram.**



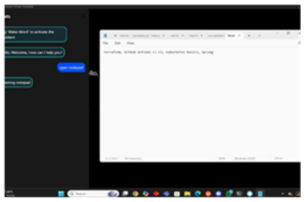
**Snapshot 3:** Authentication successful

**Snapshot 1: Initialising the voice assistant**



**Snapshot 4:** Activation of Voice assistant

**Snapshot 2:** Authentication in



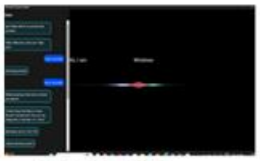
**Snapshot 5:** Command to open an application.

**Snapshot 6:** Command to open a

**Snapshot 9:** Command to play

**Snapshot 7:** Command to open a website

**Snapshot 10:** AI response and termination of voice



**Snapshot 8:** Command to set a

## V. CONCLUSION

To wrap things up, this voice assistant project, built with Python smarts and a friendly face, really hits the sweet spot for making everyday tasks a breeze. It's got that handy voice recognition, takes care of those little chores automatically, and looks good doing it with its modern design.But what truly sets it apart is how it treats your personal information. By keeping everything local, right here on your computer, it gives you a real sense of security and privacysomething that's hard to come by with those cloud-based assistants. It's more than just a tool; it's a dependable helper that can jog your memory, pull up websites without you lifting a finger, and even have a natural-sounding chat.The way it's designed is also pretty clever. It's also built to be reliable and easy to get along with, so you won't find yourself wrestling with complicated menus or errors.

## REFERENCES

[1] Girish Kumar, Hanumanta Dh, Dilshad Ahmad, and Hinge Sai Kiran explored how voice-based virtual assistants can interact with users through speech to carry out common tasks, aiming to improve accessibility and convenience.

[2] Srikanth Thirumani, Srija Reddy Korem, and Abhiram Dontha focused on building a virtual assistant specifically for Windows. Their work highlights how speech recognition can be used to make daily computer use more efficient and productive.

[3] Ashvini Khobragade, Rupali Mamale, Pranay Lohabare, and Shashank Mankar worked on designing a virtual assistant for desktop systems. Their research looks into

making these assistants smarter and more helpful by using voice commands and user-friendly interfaces.