INDUSTRIAL ORIENTED MINI PROJECT

Report

On

AI-BASED DESKTOP VOICE ASSISTANT

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

B.V.Mohan Karthik - 22261A1207

Pallikonda Sindhuja - 22261A1243

Under the guidance of

Dr. U. Chaitanya

Assistant Professor, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **AI-BASED DESKTOP VOICE ASSISTANT** submitted by **B.V.Mohan Karthik (22261A1207)** and **Pallikonda Sindhuja (22261A1243)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Dr. U. Chaitanya**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:	**IOMP Supervisor:**
Dr. U. Chaitanya	**Dr. U. Chaitanya**
Assistant Professor	Assistant Professor
Dept. of IT	Dept. of IT

EXTERNAL EXAMINAR	**Dr. D. Vijaya Lakshmi**
	Professor and HOD
	Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **AI-BASED DESKTOP VOICE ASSISTANT** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Dr. U. Chaitanya, Assistant Professor**, Department of IT, MGIT.
No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

B.V.Mohan Karthik- 22261A1207

Pallikonda Sindhuja- 22261A1243

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our project guide **Dr. U. Chaitanya , Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi**, Assistant Professor, Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

B.V.Mohan Karthik- 22261A1207

Pallikonda Sindhuja-22261A1243

ABSTRACT

In recent years, the integration of Artificial Intelligence (AI) in human-computer interaction has led to the development of intelligent voice assistants that enable users to control digital systems through voice commands. This project presents the design and implementation of a Personal Voice Assistant using Python, capable of performing a variety of tasks based on natural language input. The assistant is inspired by systems such as Google Assistant, Alexa, and JARVIS from the Iron Man series, and aims to provide a simple yet effective platform for voice-based interaction on desktop environments.

The assistant utilizes several Python libraries such as Speech Recognition for speech-to-text conversion, pyttsx3 for text-to-speech functionality, and wikipedia, web browser, and smtplib for executing specific tasks like information retrieval, web browsing, and email handling. It can perform actions such as opening websites and applications, fetching weather updates, searching Wikipedia, playing music, sending emails, taking notes, reading news headlines, and reporting system time and date. The system is designed with a modular approach, making it easy to maintain and extend with additional features.

The assistant listens to the user's voice command through a microphone, processes the input using natural language processing techniques, maps the command to a corresponding task, and executes it while providing a verbal response. This creates a seamless and interactive user experience. The project demonstrates how AI and automation can be integrated to improve accessibility, productivity, and user convenience on personal computing systems.

Overall, this AI-based voice assistant exemplifies the potential of Python as a versatile programming language for building smart applications. The project lays a foundation for future improvements such as GUI integration, smart device connectivity, machine learning-based understanding, and advanced conversational capabilities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

In today’s fast-paced digital world, users often perform repetitive tasks on computers such as opening applications, checking the weather, searching online, playing music, taking notes, or sending emails. While these tasks may seem minor, collectively they consume a significant amount of time and manual effort. As technology advances, there is a strong demand for intelligent systems that can automate routine operations and provide a hands-free user experience. Voice-based interaction stands out as one of the most natural and intuitive ways for humans to communicate with machines.

Inspired by fictional AI systems like JARVIS from Iron Man, this project is motivated by the goal of making daily computer interactions more efficient and intelligent. Rather than relying on traditional input methods like a keyboard or mouse, a voice assistant allows the user to control and interact with their system through simple spoken commands. This not only improves accessibility for people with disabilities but also enhances multitasking and user convenience.

Existing mainstream voice assistants, such as Google Assistant, Amazon Alexa, and Apple Siri, have demonstrated the potential of voice-based technology. However, they are primarily designed for mobile devices and smart home environments. They lack comprehensive desktop integration, are often dependent on internet connectivity, and come with privacy concerns due to cloud-based data handling.

1.2 PROBLEM STATEMENT

As computer usage becomes more integral to our daily lives, users are often burdened with performing repetitive and time-consuming tasks such as launching applications, searching the web, managing files, sending emails, and accessing various services. While these tasks are essential, doing them manually can reduce efficiency and productivity, especially for users who multitask or rely on technology for work or study.

Although modern voice assistants like Google Assistant, Siri, and Alexa offer voice-activated services, they are largely designed for smartphones or smart home devices, not for desktop environments. These systems typically require continuous internet access, are platform-dependent, and offer limited flexibility in terms of customization. Moreover, their functionalities are often restricted to predefined commands and services, which limits their utility in a personalized computing environment.

One of the biggest challenges with these existing systems is the lack of deep desktop integration. Users cannot fully control file systems, take screenshots, automate desktop applications, or manage tasks directly on their computer through voice. Additionally, due to their cloud-based nature, these assistants collect and process data on remote servers, raising concerns over user privacy and data security.

The core problem is the absence of an efficient, customizable, offline-capable voice assistant specifically built for desktops that respects user privacy while offering powerful automation features. Users need a solution that is not only intelligent but also lightweight, secure, and adaptable to their individual workflows. This project seeks to fill that gap by developing a Python-based desktop voice assistant that can understand natural voice commands and perform a wide range of tasks efficiently without relying on third-party platforms.

1.3 EXISTING SYSTEM

Over the past decade, several voice-activated personal assistants have been developed to simplify human-computer interaction. Popular systems such as Google Assistant, Amazon Alexa, Apple Siri, and Microsoft Cortana have set the benchmark for voice-controlled technology. These assistants enable users to perform tasks like setting reminders, playing music, making calls, controlling smart devices, and retrieving information from the internet using natural language.

These systems leverage cloud-based services and powerful artificial intelligence algorithms to process voice commands, interpret intent, and provide relevant responses. They are primarily embedded in mobile phones, smart speakers, and home automation devices, making them widely accessible and convenient for everyday tasks. With deep integration into their respective ecosystems, these assistants can manage calendars, navigate maps, and respond to queries with considerable accuracy.

However, when it comes to desktop usage, these systems reveal certain limitations. Most of them are not designed for full desktop control, especially in offline environments. For example, Siri is limited to Apple devices, Cortana is being phased out in Windows, and Alexa requires external device configuration to be used on a PC. Moreover, none of these platforms allow in-depth customization or scripting of specific commands that users may need for personalized tasks on their computers.

Another key concern is data privacy. These assistants rely on constant internet connectivity and cloud processing to function, which involves collecting user data and transmitting it to external servers. This raises serious concerns about security, data ownership, and unauthorized access, especially for users dealing with sensitive information.

1.3.1 LIMITATIONS

- **Limited Desktop Integration**: Most current voice assistants are optimized for smartphones and smart devices, with only limited features available for desktop systems. They cannot fully control computer functionalities such as launching local applications, managing files, or interacting with desktop-based software through voice commands.

- **Internet Dependency:** These systems depend heavily on cloud-based processing to interpret voice commands and deliver results. As a result, they are not functional without an active internet connection. This limits their usability in offline environments or in areas with poor connectivity.

- **Lack of Customization:** Mainstream voice assistants operate within tightly controlled ecosystems and do not provide users with the flexibility to define or modify commands according to their specific needs. Developers and end-users are often restricted from altering the system's behavior or extending its functionality.
- **Privacy Concerns:** All popular assistants process user data in the cloud, which raises significant concerns about privacy and data security. Users have limited control over what information is collected, stored, or shared, which is particularly problematic for sensitive or personal tasks.

- **Platform Dependency:** Many voice assistants are bound to specific platforms or hardware. For example, Siri is exclusive to Apple devices, and Cortana is integrated only into Windows. This restricts cross-platform compatibility and limits the user base.

1.4 PROPOSED SYSTEM

The proposed system is a Python-based desktop voice assistant designed to operate efficiently in offline environments while offering extensive customization, enhanced control, and privacy. It aims to bridge the gap left by existing voice assistants that are either internet-dependent, limited to mobile platforms, or lack in-depth integration with desktop systems.

This assistant listens to user commands through a microphone, processes them using speech recognition, and responds using text-to-speech synthesis. Based on the recognized command, it can execute a wide range of tasks such as launching applications, searching the web, playing music, taking screenshots, checking the weather, sending emails, and fetching information from sources like Wikipedia or Wolfram Alpha.

Unlike cloud-based assistants, this system is designed to work locally on the user's computer, ensuring that no data is sent to external servers. This drastically improves user privacy and gives the user full control over their data and commands. It is ideal for users who need an assistant that respects privacy, works offline, and is easy to extend.

The assistant is built using open-source Python libraries such as speech recognition, pyttsx3, py audio, and various APIs. These libraries allow the system to process spoken input, synthesize voice responses, and access real-time data like news and weather. Its modular design also makes it easy for users to add or modify functionalities based on their personal requirements.
This system is not only lightweight and fast, but also platform-independent and adaptable for a wide range of use cases. It can serve students, professionals, and tech enthusiasts who wish to automate tasks and enhance their productivity through hands-free interactions.

1.4.1 ADVANTAGES

• **Offline Functionality:** Unlike most mainstream voice assistants that rely on continuous internet connectivity, this system is capable of performing core tasks offline. It processes voice commands locally, ensuring that the assistant is usable even in low or no network conditions.

• **Enhanced Privacy and Security:** As the system runs entirely on the local machine, no user data is sent to the cloud or third-party servers. This reduces the risk of data breaches and ensures complete control over what data is accessed or stored, making it a privacy-friendly solution.

• **Customizability and Flexibility:** Being developed in Python, the assistant is fully customizable. Users can easily add new features or modify existing commands to suit their preferences or specific workflows. This level of flexibility is not available in commercial voice assistants.

• **Platform Independence and Open-Source Tools:** The system is built using open-source libraries and can run on multiple platforms such as Windows, Linux, or macOS (with minor adjustments). There is no need for proprietary software or paid licenses.

1.5 OBJECTIVES

- **To Develop an Intelligent Voice-Controlled System:** Create a desktop assistant that can understand natural language voice commands using speech recognition and respond using text-to-speech technology. The system should allow seamless interaction with the computer through spoken inputs.
- **To Automate Basic Desktop Operations:** Enable automation of common desktop tasks such as opening files, launching applications, searching the internet, retrieving information (like weather or news), sending emails, and managing system utilities—all through voice commands.
- **To Ensure Offline Functionality and Data Privacy:** Design the system to operate effectively **without internet connectivity** for core features, ensuring that user commands and data remain local to the system. This enhances security and privacy, especially for sensitive operations.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

•**Operating System:**
Windows 10/11, Linux (Ubuntu), or macOS

•**Programming Language**:
Python 3.7 or above

• **Code Editor / IDE:**
Visual Studio Code, PyCharm, or any preferred Python IDE

• **Required Python Libraries:**

- **Speech recognition** – for recognizing spoken commands
- **pyttsx3 –** for converting text to speech
- **py audio –** for handling microphone input
- **wikipedia –** for information retrieval
- **datetime** – for date and time operations
- **web browser –** for launching web searches

- **Smtplib** – for sending emails
- **wolfram alpha** – for factual and computational questions
- **Os** – for interacting with the system
- **time** – for delays and timers
- **Internet Connection:**

1.6.2 HARDWARE REQUIREMENTS

- **Processor (CPU):**
  Minimum Intel Core i3 or AMD equivalent (Recommended: i5 or higher)
- **RAM:**
  Minimum 4 GB (Recommended: 8 GB for smoother performance)
- **Storage:**
  At least 500 MB of free disk space for libraries and program files
- **Microphone:**
  Built-in or external microphone for capturing voice commands
- **Speaker / Headphones:**
  Required for audio output (text-to-speech responses)
- **System Type:**
  Desktop or Laptop with basic multimedia capabilities

2. LITERATURE SURVEY

Yuqi Huang has provided a comprehensive study on the development of voice assistants using Natural Language Processing (NLP) and AI technologies. The work presents an overall view of voice assistant systems in the AI era. However, it focuses more on general use rather than desktop-specific optimization or offline operation.[1].

P. Kunekar et al. have discussed AI-based speech recognition methods and proposed architectures for desktop voice assistants. Their work contributes to improving voice assistant performance in desktop environments. However, it may not integrate smoothly with diverse desktop applications and lacks offline compatibility, indicating a need for broader integration support. [2].

S. Gowroju et al. have developed an NLP-driven voice recognition system with contextual understanding aimed at enhancing desktop user interaction. Their approach focuses on improving recognition accuracy. However, it is limited to specific use cases and may not scale well across different platforms, suggesting a need for multi-platform support and adaptive learning. [3].

S. Alharbi et al. have carried out a systematic review of Automatic Speech Recognition (ASR) systems. Their study provides an in-depth technical understanding of speech recognition capabilities. Despite its thoroughness, the review does not address integration with desktop environments or evaluate system performance in real-world scenarios, leaving a gap in practical application. [4].

R. Benny et al. have explored OpenAI-based voice assistants with a focus on machine learning and performance evaluation. Their work emphasizes AI-enhanced interaction quality. However, the requirement for stable internet connectivity limits its effectiveness in offline environments, highlighting the need for further research on offline capability and real-world adaptability in desktop settings[5].

Table 2.1 Literature Survey of AI-Based desktop voice assistant

Ref	Author& year of publication	Journal / Conference	Method / Approach	Merits	Limitations	Research Gap
[1]	Yuqi Huang, 2023	SHS Web of Conferences, SDMC 2022	Natural Language Processing (NLP), AI Integration	Provides a comprehensive view on voice assistant development in the AI era	Focused on general voice assistant systems, not desktop-specific	More focus needed on desktop-oriented voice assistant optimization and offline operation
[2]	P. Kunekar, A. Deshmukh, S. Gajalwad, A. Bichare, K. Gunjal, S. Hingade, 2023	5th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)	AI-based Speech Recognition, NLP, Desktop Assistant Architecture	Discusses AI-based desktop assistants, contributing to desktop environment improvements	May not integrate seamlessly with existing desktop applications	Need for better integration with diverse desktop applications and offline support
[3]	S. Gowroju, S. Kumar, S. Choudhary, 2024	7th International Conference on Contemporary Computing and Informatics (IC3I)	NLP-driven Voice Recognition, Contextual Understanding	Improves desktop interaction by focusing on NLP systems	Limited to specific use cases, may not scale	Further research on multi-platform support and adaptive learning
[4]	S. Alharbi et al., 2021	IEEE Access	Automatic Speech Recognition (ASR), Systematic Literature Review	Thorough review of existing speech recognition systems	Focused mainly on technical aspects of speech recognition	Lack of focus on desktop integration and real-world performance
[5]	R. Benny, A. Muralidharan, M. Subramanian, 2024	Second International Conference on Emerging Trends in Information Technology and Engineering (ICETITE)	OpenAI-based voice assistants, Machine Learning, Performance Evaluation	Improved interaction via OpenAI and AI-driven enhancements	May require stable internet, not ideal for offline	More research on offline performance and adaptive learning in real-world desktop cases

3. ANALYSIS AND DESIGN

The development of a desktop-based AI voice assistant involves a detailed understanding of the system's requirements and a thoughtfully planned design. At its core, the assistant should allow users to interact with their computer using natural speech. It must capture audio input, convert it into text using speech recognition, understand the intent using Natural Language Processing (NLP), execute the relevant command, and respond with appropriate voice feedback. These capabilities enable hands-free control of daily computing tasks, increasing productivity and convenience for users.

Apart from these core functionalities, the system should also fulfill several non-functional requirements. It should be efficient, providing fast and accurate responses. It must operate smoothly on standard hardware without consuming too many resources. In addition, support for offline functionality is important so the assistant can work without constant internet access. Privacy is also a key consideration, especially when handling user inputs and data.

The design of the system is modular to ensure maintainability and scalability. Each component is responsible for a specific task: the speech recognition module handles audio-to-text conversion, the NLP processor interprets the command, the execution module performs the required actions (like opening software or searching the web), and the text-to-speech module generates a voice-based response. These modules are connected in a pipeline, forming a logical flow: from voice input to response output.

3.1 MODULES

Speech Recognition Module

Captures the user's voice input and converts it into text using Python-based speech recognition libraries like SpeechRecognition or Vosk.

• **Input**: User’s voice via microphone

• **Output**: Transcribed text of the spoken command

Natural Language Processing (NLP)Module

Interprets the transcribed speech and extracts the user's intent by identifying commands and relevant keywords. This module helps in determining what task the user wants to perform.
•**Input**:Text obtained from the Speech Recognition Module
• **Output**: Interpreted command or action (e.g., open browser, check weather)

Execution Module

Carries out the actual execution of commands such as launching applications, opening websites, or fetching data from APIs (e.g., weather, news). Interfaces with operating system functions and APIs.
•**Input**:Interpreted command from the NLP module
• **Output**: Execution of the desired action (e.g., app launched, weather reported)

Text-to-Speech (TTS)Module

Converts system-generated responses into spoken words using libraries like pyttsx3. Offers audio feedback to the user after processing a command.
• **Input**:Textual response(confirmation or result of the task)

• **Output**: Voice output delivered via speakers.

3.2 ARCHITECTURE

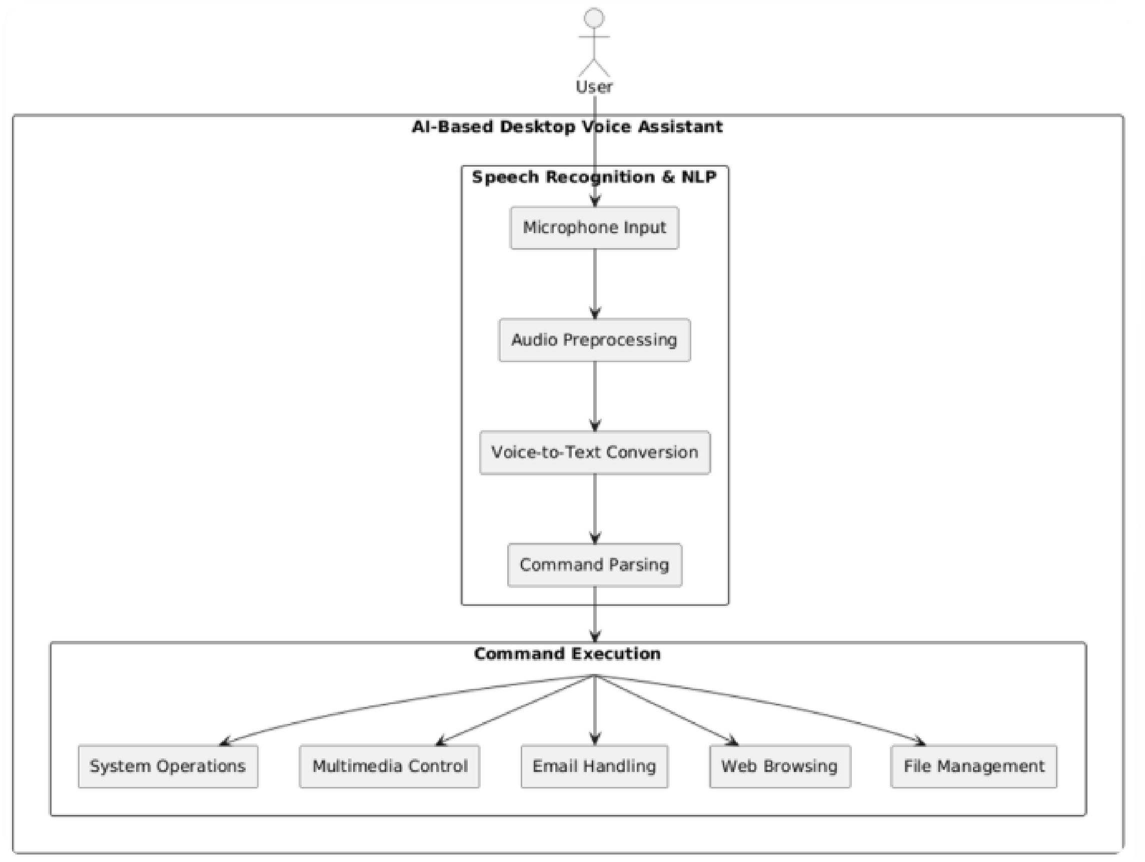


Fig. 3.2.1 Architecture of AI-Based Desktop Voice Assistant

The architecture of the system, as shown in Fig. 3.2.1, is designed to provide an interactive and intelligent voice-controlled interface for desktop environments. The system begins with the User Input phase, where voice commands are captured through a microphone. This voice input is handled in real-time and passed to the Speech Recognition and Natural Language Processing (NLP**)** module for interpretation.

In the Speech Recognition & NLP component, the input undergoes several sub-processes. Initially, the Microphone Input captures the raw audio signal. This is followed by Audio Preprocessing, which removes background noise and optimizes the input for better accuracy. Next, the Voice-to-Text Conversion step transforms the audio into readable text using speech recognition libraries such as speech_recognition or vosk. The resulting text is then processed through the Command Parsing step, where NLP techniques are used to determine the intent behind the spoken instruction.

Once the command is parsed and understood, it is forwarded to the Command Execution module. This module is responsible for carrying out a variety of tasks based on user intent. The system supports actions like System Operations (such as opening or closing applications), Multimedia Control (managing playback or volume), Email Handling (reading or composing emails), Web Browsing (launching web pages), and File Management (accessing or organizing local files). Each of these functions is supported by dedicated backend scripts and system calls that interface with the underlying operating system.

After executing the task, the system generates a response that is converted to speech through the Text-to-Speech (TTS**)** module. This component utilizes libraries like pyttsx3 to synthesize speech, which is then played back via the speaker to confirm the action to the user or provide the requested information.

This voice assistant system is modular and extensible, allowing for the integration of new features such as weather updates, chatbot responses, or calendar event handling. By leveraging open-source libraries and clear modular separation, the system ensures flexibility, maintainability, and ease of enhancement in future development.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A use case diagram is a fundamental visual tool used in system analysis and design to represent the dynamic interactions between a system and its external entities, known as actors. It captures the functional requirements of a system by showing various use cases—specific actions or services the system performs—and the actors that initiate these actions.
In the context of the AI-Based Desktop Voice Assistant system, the use case diagram (as shown in Fig. 3.3.1.1) illustrates how the user interacts with the voice assistant through voice commands and how the system processes those commands to perform specific operations.

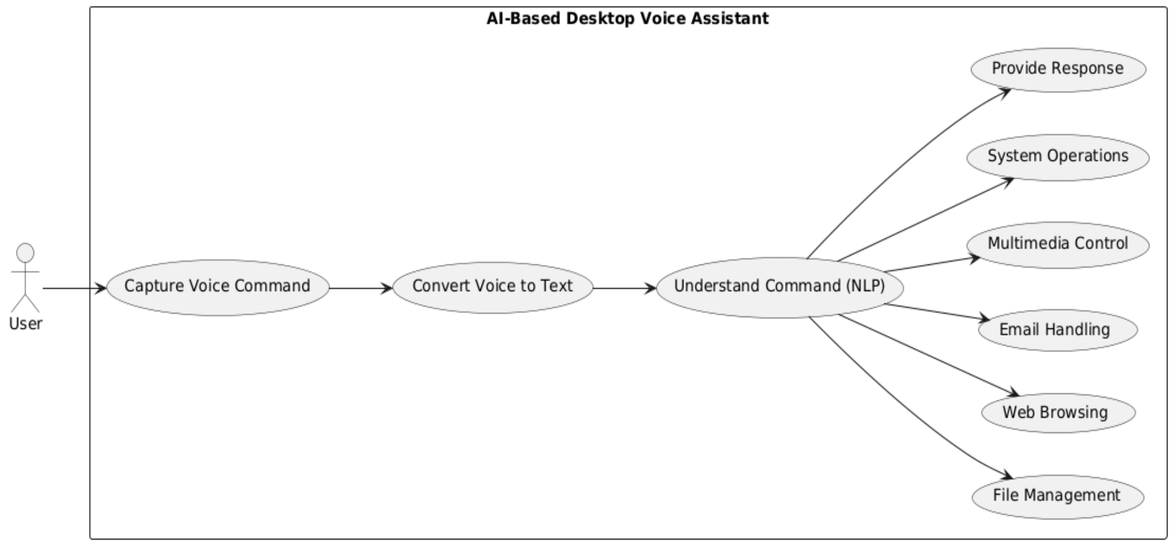


Fig. 3.3.1.1 Use Case Diagram

Actors:

1. **User:** Represents the person who interacts with the desktop voice assistant by giving voice command. .The user initiates all actions performed by the system.
2. **System**: Refers to the AI-based desktop voice assistant that captures, processes, interprets, and responds to user commands using speech recognition and NLP technologies.

Use Cases:

1. **Capture Voice Command:** The system listens to and records the user’s spoken input through a microphone.
2. **Convert Voice to Text**: Converts the captured audio input into text formatting using speech-to-text technology.
3. **Understand Command(NLP)**:Applies Natural Language Processing (NLP) techniques to interpret the intent and meaning of the user’s text command.
4. **System Operations:** Executes system-level tasks such as shutting down, restarting, or opening system settings as per the user's instructions.
5. **Provide Response**: Generates and delivers appropriate verbal or textual feedback to the user based on the interpreted command.

6. **Multimedia Control:** Allows the user to control media playback, including playing, pausing, or stopping audio or video files.
7. **Email Handling:** Facilitates sending and reading of emails as instructed by the user.
8. **Web Browsing:** Enables the user to perform actions such as searching the internet or opening specific websites
9. **File Management:** Supports file-related operations like opening, saving, or deleting files based on user commands.

3.3.2 CLASS DIAGRAM

A class diagram is a structural diagram in UML that describes the system by showing its classes, attributes, methods (operations), and the relationships among the objects. In the case of the AI-Based Desktop Voice Assistant (Fig. 3.3.2.1), the diagram models the static architecture of the system, helping to define the components and their interactions for implementing voice-controlled functionalities.

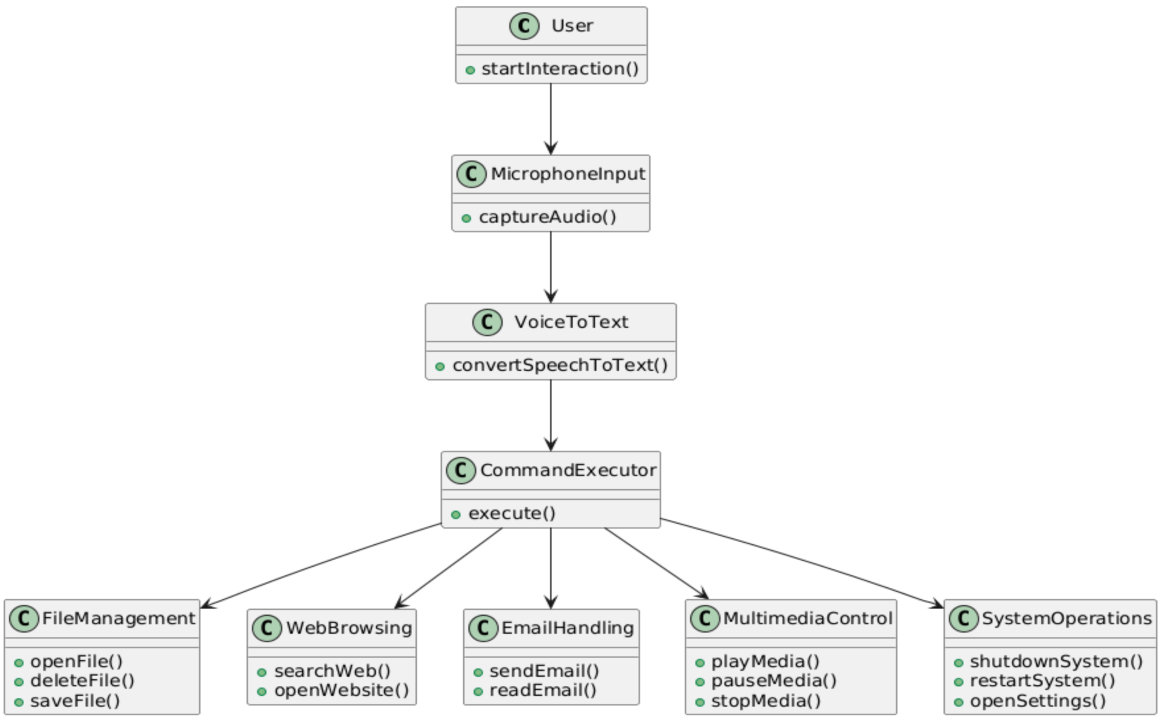


Fig. 3.3.2.1 Class Diagram

Relationships:

1. **User → Microphone Input**
   The user initiates the voice interaction via the start Interaction() method, which triggers the microphone to begin capturing audio.
2. **Microphone Input → Voice To Text**
   The microphone component sends the captured audio to the speech-to-text module using capture Audio(), which is then converted to textual commands using convert Speech To Text().
3. **Voice To Text → Command Executor**
   The text-based command is passed to the Command Executor class which interprets and delegates the command for execution using the execute() method.
4. **Command Executor → File Management**
   Executes file operations such as open File(), delete File(), and save File().
5. **Command Executor → Web Browsing**
   Handles web-related commands such as search Web() and open Website().
6. **Command Executor → Email Handling**
   Manages email tasks like send Email() and read Email().
7. **Command Executor → Multimedia Control**
   Manages media controls using play Media(), pause Media(), and stop Media() methods.

8.CommandExecutor → System Operations
   Executes system-level commands like shutdown System(), restart System(), and open Settings().

System Flow:

1. **User Interaction**: The process begins when the user invokes start Interaction() to begin a session with the assistant.
2. **Audio Capture and Conversion:** The audio is captured via the Micro phone Input class and converted to text by the Voice To Text class.
3. **Command Execution:** The Command executor class receives the command and calls the appropriate method based on the user's intent.

4.Functional Execution: Based on the command type, actions are performed across components such as:

- File management for file operations
- Web browsing for internet-related tasks
- Email handling for managing emails
- Multimedia control for handling media
- System operations for system-level changes

3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a behaviour UML diagram that illustrates the dynamic aspects of a system by modelling the workflow of activities and actions. As shown in Fig. 3.3.3.1, this diagram represents the step-by-step process of how a user interacts with the AI-Based Desktop Voice Assistant system, from initiating a voice command to receiving a response. It is particularly useful for visualizing the logic of complex operations or business processes.

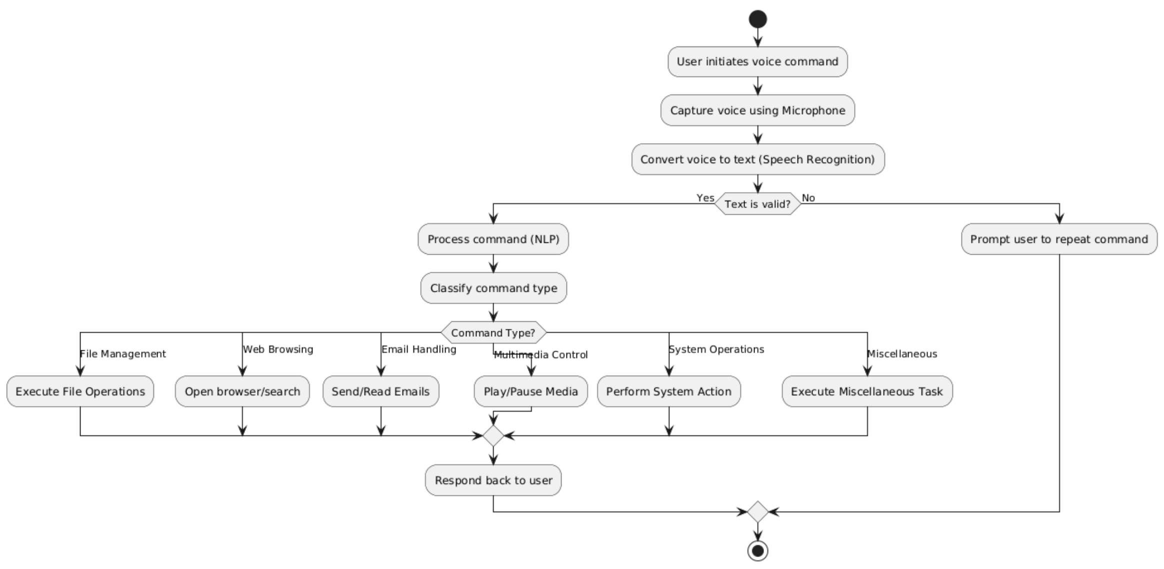


Fig. 3.3.3.1 Activity Diagram

Flow Explanation

1. **User Initiates Voice Command**

   The workflow starts when the user triggers the voice assistant with a voice command.

2. **Capture Voice using Microphone**

   The system activates the microphone to capture the user’s spoken input.

3. **Convert Voice to Text (Speech Recognition)**

   The captured audio is processed using speech recognition to transform it into text.

4. **Text Valid?**

   The system checks if the converted text is valid and recognizable:

   - **Yes:** Proceed to command processing.
   - **No:** Prompt the user to repeat the command and loop back.
   - **Process Command (NLP)**

   Natural Language Processing (NLP) is applied to understand the intent and context of the user’s command.

5. **Classify Command Type**

   The assistant categorizes the command into a specific type for appropriate handling.

6. **Command Type?**

   The command is classified into one of the following categories:

   - **File Management:**

     Executes operations like opening, deleting, or saving files.
   - **Web Browsing:**

     Initiates browser actions such as opening a website or searching content.
   - **Email Handling:**

     Performs actions like sending or reading emails.
   - **Multimedia Control:**

     Plays, pauses, or stops media content.
   - **System Operations:**

     Handles system-related tasks like shutdown, restart, or accessing settings.
   - **Miscellaneous:**

     Covers any other tasks that do not fall under the above categories.

7. **Execute Action**

   The respective task is executed based on the command type.

8. **Respond Back to User**

   After completing the action, the system responds back to the user with the result or confirmation.

9. **End of Activity**

   The activity terminates, and the system awaits the next voice interaction.

3.3.4 SEQUENCE DIAGRAM

A Sequence Diagram is a type of interaction diagram in Unified Modeling Language (UML) that represents how processes operate with one another and in what order. It shows the flow of messages between actors and system components over time. As illustrated in Fig. 3.3.4.1, this diagram details the operation of a Voice Assistant system from the user's spoken input to the final system response.

This model emphasizes the temporal sequence of messages, helping to identify the responsibilities and interactions of each component in the system's architecture.

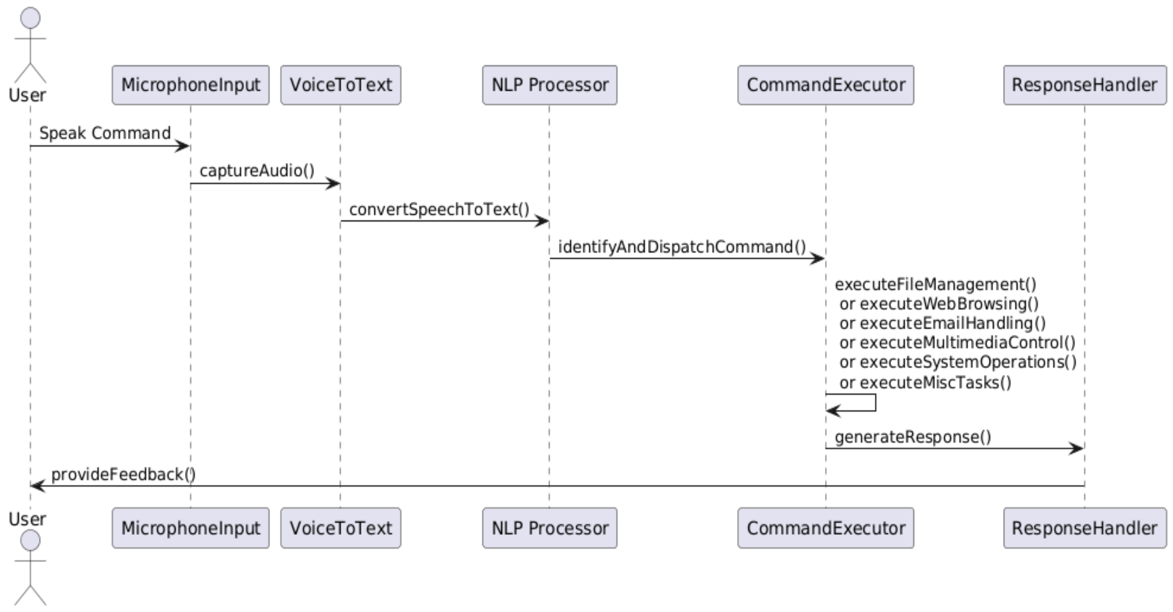


Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships

- **User and System Interaction**:
  - **Speak Command:**

    The user initiates the interaction by speaking a voice command.
- **Microphone Input Component:**
  - **Capture Audio():**

    This component captures the spoken input from the user through the microphone.
- **Voice To Text Component:**
  - **Convert Speech To Text():**

    The captured audio is converted into a textual command using a speech recognition engine.
- **NLP Processor:**
  - **Identify And Dispatch Command():**

    : The NLP processor interprets the textual input to determine user intent and identifies the appropriate type of command (e.g., file, web, email).
- **Command Executor Component:**
  - **Execute Command Type:**

    Based on the command classification, the appropriate method is called from the following:
  - Execute File Management()
  - Execute Web Browsing()
  - Execute Email Handling()
  - Execute Multimedia Control()
  - Execute System Operations()
  - Execute Misc Tasks()

3.3.5 COMPONENT DIAGRAM

A Component Diagram illustrates the structure and interactions of the system's major modules. As shown in Fig. 3.3.5.1, the voice assistant begins with user input via a microphone. The input is processed through the Speech Recognition Module and the NLP Module, which determine the user's intent.

The Jarvis Controller coordinates command execution and speech response. The Command Executor directs tasks to various functional scripts like launching apps, opening websites, checking the date/time, or fetching weather info using an external API. The TTS Module converts the system's reply into speech, completing the interaction through Speaker Output.

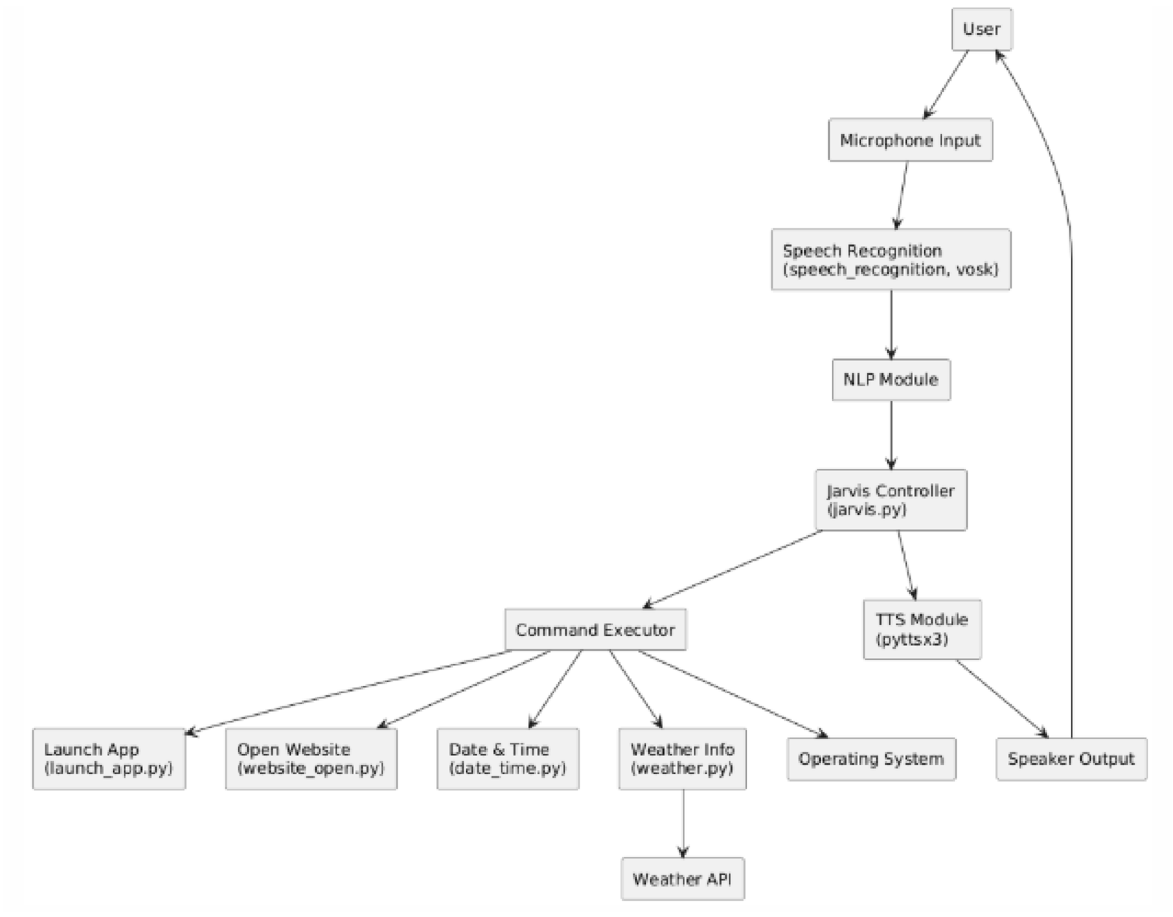

User
Microphone Input
Speech Recognition (speech_recognition, vosk)
NLP Module
Jarvis Controller (jarvis.py)
Command Executor
TTS Module (pyttsx3)
Launch App (launch_app.py)
Open Website (website_open.py)
Date & Time (date_time.py)
Weather Info (weather.py)
Operating System
Speaker Output
Weather API

Fig. 3.3.5.1 Component Diagram

Main Components:

1. **User**
   - **Description**: The end-user who interacts with the assistant using voice commands. The user provides input through the microphone and receives responses via speaker output.
2. **Microphone Input**
   - **Description**: Captures the user's voice input and sends the audio signal for processing.

3. **Speech Recognition (speech recognition, vosk)**
    - **Description**: Converts the user’s spoken input into text. Uses libraries such as speech recognition and vosk for offline/online speech-to-text conversion.
4. **NLP Module**:
    - **Description**: Processes the converted text to understand the user’s intent using Natural Language Processing techniques.
5. **Jarvis Controller (jarvis.py)**
    - **Description**: Acts as the main control unit of the system, coordinating between the NLP output, command execution, and TTS (Text-to-Speech) module.
6. **Command Executor**
    - **Description**: Executes specific user commands such as opening applications, websites, fetching weather data, or providing the current date and time. Delegates tasks to different Python scripts.
7. **Launch App (launch_app.py)**
    - **Description**: Script responsible for launching desktop applications based on user voice commands.
8. **Open Website (website_open.py)**
    - **Description**: Handles the task of opening URLs/websites as requested by the
9. **Date & Time (date_time.py)**
    - **Description**: Provides the current system date and time upon user query.
10. **Weather Info (weather.py)**
    - **Description**: Retrieves weather information by connecting to an external Weather API and processes the results to provide weather updates.
11. **Weather API**
    - **Description**: External service used to fetch real-time weather data based on location input or system location.
12. **Operating System**
    - **Description**: Supports execution of system-level tasks like launching applications, opening folders, or shutting down the system.
13. **TTS Module (pyttsx3)**
    - **Description**: Converts text-based responses into speech output, allowing the assistant to verbally respond to the user.
14. **Speaker Output**

- **Description**: Outputs the final spoken response to the user, completing the interaction loop.

3.3.6 DEPLOYMENT DIAGRAM

The Deployment Diagram illustrates the physical deployment of the voice assistant system on the user’s machine. As shown in Fig. 3.3.6.1, the system includes components like the Main Module, ASR Engine (vosk), NLP Module (spacy), TTS Engine (pyttsx3**)**, and Command Executor, all running within the Voice Assistant App on the user’s computer. Input is received through the microphone and output is delivered via the speaker. The system also interacts with the local file system to store screenshots and log files, ensuring efficient offline functionality.

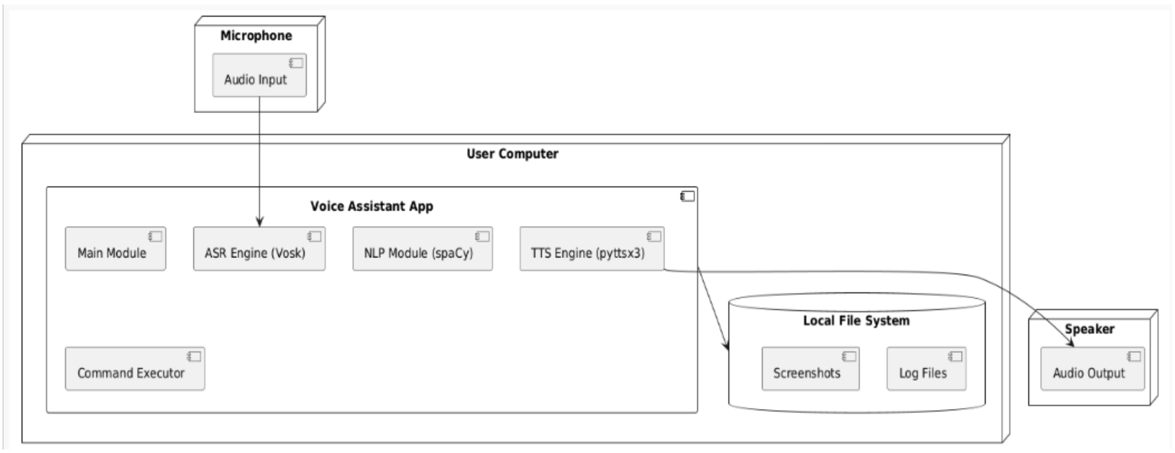


Fig. 3.3.6.1 Deployment Diagram

The deployment diagram shows how the system components are distributed across three main nodes:

- **Microphone:** Captures the user's speech as audio input, which is processed by the voice assistant system.
- **User Computer**: Hosts the Voice Assistant App, which includes key components such as the Main Module, ASR Engine (vosk) for speech recognition, NLP Module (spacy) for natural language understanding, TTS Engine (pyttsx3) for text-to-speech

conversion, and the Command Executor. It also interacts with the Local File System to store outputs like screenshots and logs.

- **Speaker**: Delivers the final audio output to the user after processing.

This setup ensures seamless voice interaction by integrating audio input, natural language processing, command execution, and audio output within a single system.

3.4 METHODOLOGY

Data Acquisition

- **Live Data**: Real-time voice input is captured from the system microphone using Python’s speech recognition library. This audio is processed instantly and converted into text. Additionally, the system fetches live web-based information—such as weather updates or Wikipedia summaries—using APIs and libraries like wikipedia, pywhatkit, and requests.
- **Purpose**: Audio input enables immediate voice-based interaction, while external APIs provide dynamic and context-aware responses, enriching the assistant's functionality in real time.

Model Steps

- **Voice Input**: The system starts by capturing the user's spoken command using the microphone.
- **Speech-to-Text**:: The voice is converted into text using the speech recognition module.
- **Command Processing:** The converted text is parsed to determine the user's intent using basic Natural Language Processing.
- **Task Mapping**: Based on the intent, the system maps the command to predefined actions like opening apps, searching the web, or telling the time.

- **Execution & Feedback**: The required task is performed, and a response is generated using pyttsx3 for voice output.

Models Used

Speech Recognition

This module handles the conversion of real-time audio input into text using services like Google Web Speech API. It enables the assistant to understand the spoken command.

How it works:
Audio is captured and converted into a WAV file. The recognizer processes this using a pre-trained API to output text.
Why it's used:
It simplifies the voice input process, supports multiple recognition engines, and provides high accuracy in quiet environments.

pyttsx3 (Text-to-Speech)

This module converts text responses into speech, providing voice-based feedback to the user.

How it works:
After task execution, the response string is converted to speech using system voices (offline).
Why it's used:
It works offline, supports both Windows and Linux, and gives a natural interaction feel.

Web Libraries (wikipedia, pywhatkit, web browser)

Used to fetch or execute web-based commands like searches or summaries.

How it works:
Text-based commands are parsed; relevant keywords are passed to libraries that handle the data fetching.
Why it's used:
To enhance the assistant's ability to provide up-to-date and useful information instantly.

4. CODE AND IMPLEMENTATION

4.1 CODE

jarvis.py

```
import pyttsx3
import datetime
import speech_recognition as sr
import wikipedia
import webbrowser as wb
import os
import random
import pyautogui
import pyjokes

engine = pyttsx3.init()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)
engine.setProperty('rate', 150)
engine.setProperty('volume', 1)

def speak(audio) -> None:
    engine.say(audio)
    engine.runAndWait()

def time() -> None:
    current_time = datetime.datetime.now().strftime("%I:%M:%S %p")
    speak("The current time is")
    speak(current_time)
    print("The current time is", current_time)

def date() -> None:
    now = datetime.datetime.now()
    speak("The current date is")
    speak(f"{now.day} {now.strftime('%B')} {now.year}")
    print(f"The current date is {now.day}/{now.month}/{now.year}")

def wishme() -> None:
    speak("Welcome back, sir!")
    print("Welcome back, sir!")
    hour = datetime.datetime.now().hour
    if 4 <= hour < 12:
        speak("Good morning!")
    elif 12 <= hour < 16:
        speak("Good afternoon!")
    elif 16 <= hour < 24:
        speak("Good evening!")
    else:
        speak("Good night, see you tomorrow.")
    assistant_name = load_name()
    speak(f"{assistant_name} at your service. Please tell me how may I assist you.")
    print(f"{assistant_name} at your service. Please tell me how may I assist you.")

def screenshot() -> None:
    img = pyautogui.screenshot()
    img_path = os.path.expanduser("~\\Pictures\\screenshot.png")
    img.save(img_path)
    speak(f"Screenshot saved as {img_path}.")
    print(f"Screenshot saved as {img_path}.")

def takecommand() -> str:
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        r.pause_threshold = 1
        try:
            audio = r.listen(source, timeout=5)
        except sr.WaitTimeoutError:
            speak("Timeout occurred. Please try again.")
            return None
    try:
        print("Recognizing...")
        query = r.recognize_google(audio, language="en-in")
        print(query)
        return query.lower()
    except sr.UnknownValueError:
        speak("Sorry, I did not understand that.")
        return None
    except sr.RequestError:
        speak("Speech recognition service is unavailable.")
        return None
    except Exception as e:
        speak(f"An error occurred: {e}")
        return None

def play_music(song_name=None) -> None:
    song_dir = os.path.expanduser("~\\Music")
    songs = os.listdir(song_dir)
    if song_name:
        songs = [song for song in songs if song_name.lower() in song.lower()]
    if songs:
        song = random.choice(songs)
        os.startfile(os.path.join(song_dir, song))
        speak(f"Playing {song}.")
    else:
        speak("No song found.")

def set_name() -> None:
    speak("What would you like to name me?")
    name = takecommand()
    if name:
        with open("assistant_name.txt", "w") as file:
            file.write(name)
        speak(f"Alright, I will be called {name} from now on.")
    else:
        speak("Sorry, I couldn't catch that.")

def load_name() -> str:
    try:
        with open("assistant_name.txt", "r") as file:
            return file.read().strip()
    except FileNotFoundError:
        return "Jarvis"

def search_wikipedia(query):
    try:
        speak("Searching Wikipedia...")
        result = wikipedia.summary(query, sentences=2)
        speak(result)
        print(result)
    except wikipedia.exceptions.DisambiguationError:
        speak("Multiple results found. Please be more specific.")
    except Exception:
        speak("I couldn't find anything on Wikipedia.")

def create_file(name):
    try:
        path = os.path.join(os.path.expanduser("~/Desktop"), f"{name}.txt")
        with open(path, "w") as file:
            file.write("This is a new file created by your assistant.")
        speak(f"File {name}.txt created on Desktop.")
    except Exception as e:
        speak(f"Failed to create file: {e}")

def open_file(name):
    try:
        path = os.path.join(os.path.expanduser("~/Desktop"), f"{name}.txt")
        os.startfile(path)
        speak(f"Opening file {name}.txt.")
    except Exception as e:
        speak(f"Could not open the file: {e}")

def create_folder(name):
    path = os.path.join(os.path.expanduser("~/Desktop"), name)
    try:
        os.makedirs(path)
        speak(f"Folder {name} created on Desktop.")
    except Exception as e:
        speak(f"Could not create folder: {e}")

def open_folder(name):
    path = os.path.join(os.path.expanduser("~/Desktop"), name)
    try:
        os.startfile(path)
        speak(f"Opening folder {name}.")
    except Exception as e:
        speak(f"Could not open the folder: {e}")

if __name__ == "__main__":
    wishme()

    while True:
        query = takecommand()
        if not query:
            continue

        if "time" in query:
            time()
        elif "date" in query:
            date()
        elif "wikipedia" in query:
            query = query.replace("wikipedia", "").strip()
            search_wikipedia(query)
        elif "play music" in query:
            song_name = query.replace("play music", "").strip()
            play_music(song_name)
        elif "open youtube" in query:
            wb.open("https://youtube.com")
        elif "open google" in query:
            wb.open("https://google.com")
        elif "open gmail" in query:
            wb.open("https://mail.google.com")
        elif "change your name" in query:
            set_name()
        elif "screenshot" in query:
            screenshot()
        elif "tell me a joke" in query:
            joke = pyjokes.get_joke()
            speak(joke)
        elif "shutdown" in query:
            speak("Shutting down the system, goodbye!")
            os.system("shutdown /s /f /t 1")
            break
        elif "restart" in query:
            speak("Restarting the system, please wait!")
            os.system("shutdown /r /f /t 1")
            break
        elif "offline" in query or "exit" in query:
```

```
        speak("Going offline. Have a good day!")
        break
    elif "create new file" in query:
        name = query.replace("create new file", "").strip()
        create_file(name)
    elif "open file" in query:
        name = query.replace("open file", "").strip()
        open_file(name)
    elif "create new folder" in query:
        name = query.replace("create new folder", "").strip()
        create_folder(name)
    elif "open folder" in query:
        name = query.replace("open folder", "").strip()
        open_folder(name)
```

4.2 IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

```
AIDesktopVoiceAssistant/
│
├── main.py                # Main program file with assistant logic
├── requirements.txt         # Python package dependencies
├── assistant_name.txt        # Stores custom name of the assistant
│
├── utils/
│   └── helper_functions.py    # (Optional) For modularizing code later
│
├── assets/
│   ├── audio/              # Future: custom audio clips (e.g., greetings)
│   └── icons/              # Icons if GUI is added later
│
└── README.md              # Documentation (optional but recommended)
```

Installing Python Packages

In your terminal, navigate to the project directory and run:

bash

CopyEdit

pip install -r requirements.txt

requirements.txt content:

nginx

CopyEdit

pyttsx3

datetime

SpeechRecognition

wikipedia

pyautogui

pyjokes

pyaudio

If pyaudio fails, run:

bash

CopyEdit

pip install pipwin

pipwin install pyaudio

Setting Up & Running the Assistant

1. Clone or place all your files inside the AIDesktopAssistant/ folder.
2. Run the assistant:

bash

Copy Edit

python main.py

3. The assistant will greet you and start listening to your voice commands.

Features & Functionalities

Feature	Command Example	Description
Time & Date	"What is the time?"	Speaks current time or date
Wikipedia Search	"Wikipedia Elon Musk"	Searches and summarizes Wikipedia content
Web Browsing	"Open YouTube / Google"	Opens respective websites in your browser
Music Playback	"Play music"	Plays random or requested song from Music folder

Feature	Command Example	Description
File/Folder Handling	"Create new file xyz"	Creates a .txt file or folder on Desktop
Screenshots	"Take a screenshot"	Saves a screenshot to your Pictures folder
Jokes	"Tell me a joke"	Fetches a random joke using pyjokes
Shutdown/Restart	"Shutdown the system"	Executes Windows shutdown/restart commands
Voice Personalization	"Change your name"	Lets user rename the assistant
Offline Mode	"Exit" or "Go offline"	Gracefully shuts down the assistant

---

🔐 Local Privacy Focus

- All actions are performed locally.
- No data is sent to the cloud.
- Ideal for users who value offline access and privacy.

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is an essential phase in software development that verifies the correct functionality, usability, performance, and reliability of the application under varying conditions. It ensures the final system meets the desired specifications and behaves consistently under expected and unexpected inputs. Effective testing minimizes the risk of system failures and enhances the user experience by identifying bugs or inconsistencies early in the lifecycle.

In this project, **AI Desktop Voice Assistant**, testing was crucial to ensure that personalized recommendations, user feedback tracking, gamification (star collection), and SMS-based reminders all functioned as intended. The goal was to validate that each module—from user registration to feedback analysis and administrative monitoring—performed seamlessly and accurately contributed to behavioral health tracking.

The test cases were crafted to verify the major functionalities and flow of the application, including:

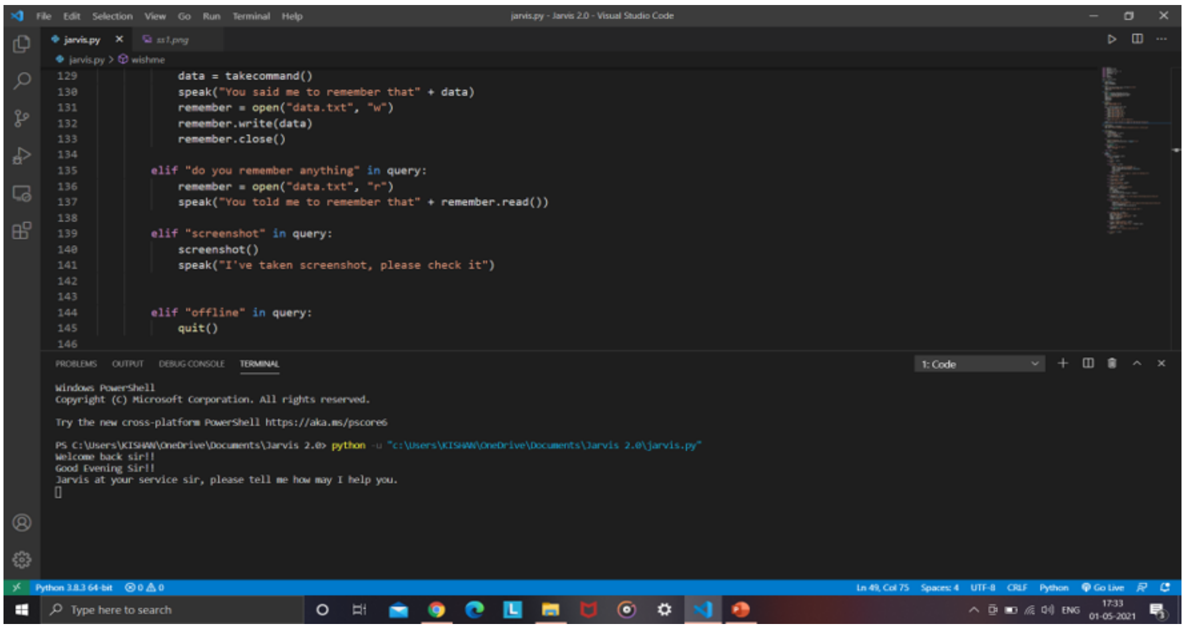
- Accurate voice-to-text conversion through ASR (Automatic Speech Recognition)
  • Correct understanding of user intent via NLP (Natural Language Processing)
  • Execution of system-level commands (opening apps, searching the web, etc.)
  • Context-aware and timely feedback through voice or visual response
  • Logging of commands and learning user behavior for future improvement

5.2 TEST CASES:

Table 5.1 Test Cases of Rhythm Restore

Test Case ID	Test Case Name	Test Description	Expected Output	Actual Output	Remarks
TC_01	Initial Page	Verify if the application loads the initial home or welcome screen correctly	Initial screen with logo and start button is displayed	Initial screen loaded successfully	Success
TC_02	Date & Time	Test if the system correctly detects and logs current date and time from the device	Current system date and time are auto-filled in the form	Date and time detected and shown correctly	Success
TC_03	Wikipedia	Check if clicking on the "Learn More" button opens relevant Wikipedia information	Wikipedia page opens in a new browser tab/window	Wikipedia article opened as expected	Success
TC_04	Youtube	Test if YouTube id being opened on command	Health-related video loads and plays without buffering issues	Video loaded and played smoothly	Success
TC_05	Offline Songs	Verify if songs or audio play without internet connection	Audio files should play from local storage	Songs played correctly in offline mode	Success
TC_06	Searching in Google	Test if voice or text-based search opens Google search results	Google search results shown based on user query	Google search executed and results displayed	Success
TC_07	Screenshot	Check if the app allows users to take a screenshot of their current dashboard	Screenshot is saved to gallery or a specified folder	Screenshot captured and saved correctly	Success
TC_08	Going Offline	Validate if the app notifies or handles functionality when device goes offline	"You're Offline" message shown and offline features enabled	Offline message displayed; essential features accessible offline	Success

6. RESULTS

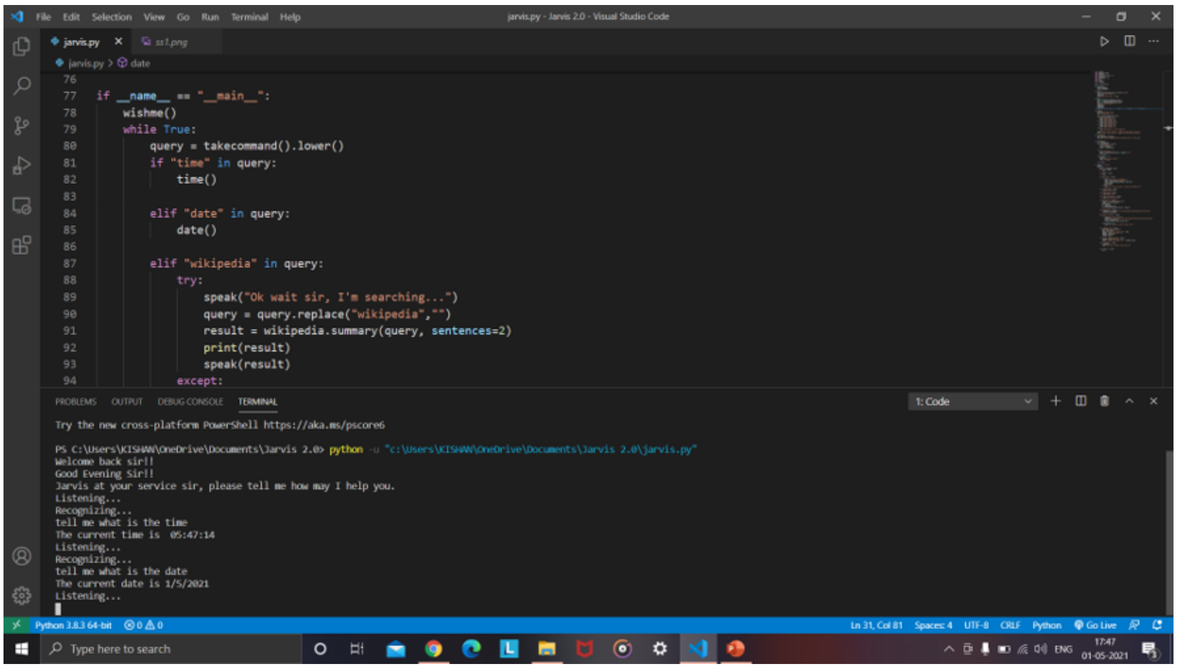


```python
            data = takecommand()
            speak("You said me to remember that" + data)
            remember = open("data.txt", "w")
            remember.write(data)
            remember.close()

        elif "do you remember anything" in query:
            remember = open("data.txt", "r")
            speak("You told me to remember that" + remember.read())

        elif "screenshot" in query:
            screenshot()
            speak("I've taken screenshot, please check it")


        elif "offline" in query:
            quit()

```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> python -u "c:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0\jarvis.py"
Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
```

Fig. 6.1 Initial page

Fig. 6.1 shows the Jarvis virtual assistant running in the Visual Studio Code terminal. When executed, the program greets the user with contextual welcome messages such as "Welcome back sir" and "Good evening sir," followed by a prompt offering assistance. This marks the successful initialization of the voice assistant system, indicating it is ready to accept user commands.

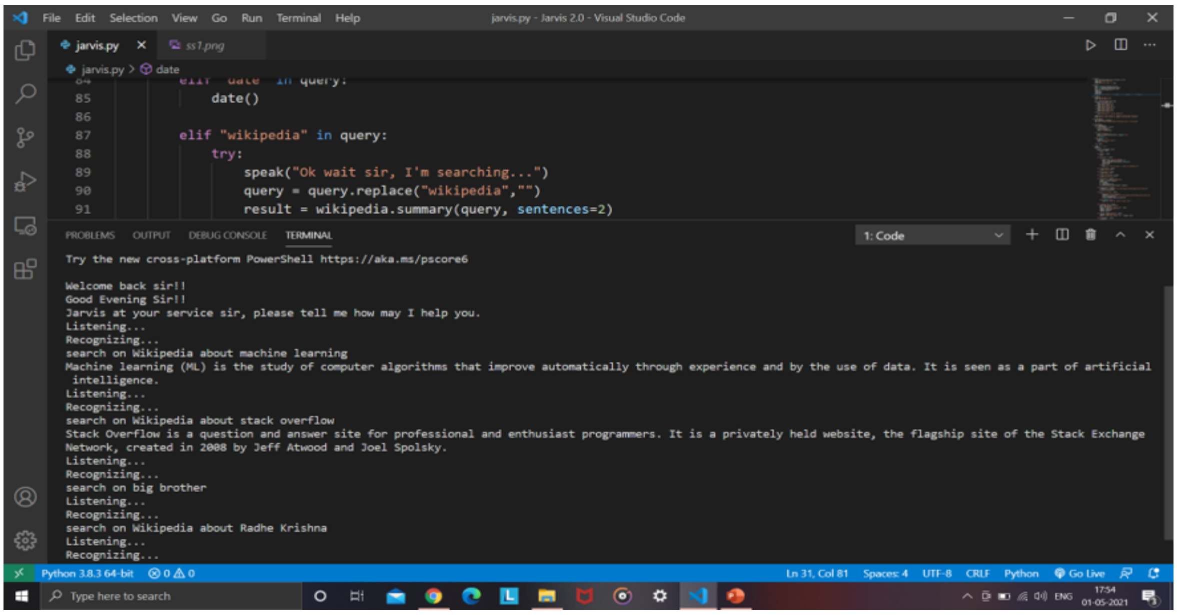


```python

if __name__ == "__main__":
    wishme()
    while True:
        query = takecommand().lower()
        if "time" in query:
            time()

        elif "date" in query:
            date()

        elif "wikipedia" in query:
            try:
                speak("Ok wait sir, I'm searching...")
                query = query.replace("wikipedia","")
                result = wikipedia.summary(query, sentences=2)
                print(result)
                speak(result)
            except:
```

```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> python -u "c:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0\jarvis.py"
Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
Listening...
Recognizing...
tell me what is the time
The current time is  05:47:14
Listening...
Recognizing...
tell me what is the date
The current date is 1/5/2021
Listening...
```

Fig. 6.2 Date & Time

Fig. 6.2 shows the Jarvis assistant responding to date and time queries in the Visual Studio Code terminal. The user initiates the interaction by asking for the current time and date. The assistant listens, processes the voice commands, and accurately responds with “The current time is 05:47:14” and “The current date is 1/5/2021,” demonstrating real-time voice recognition and datetime functionality.

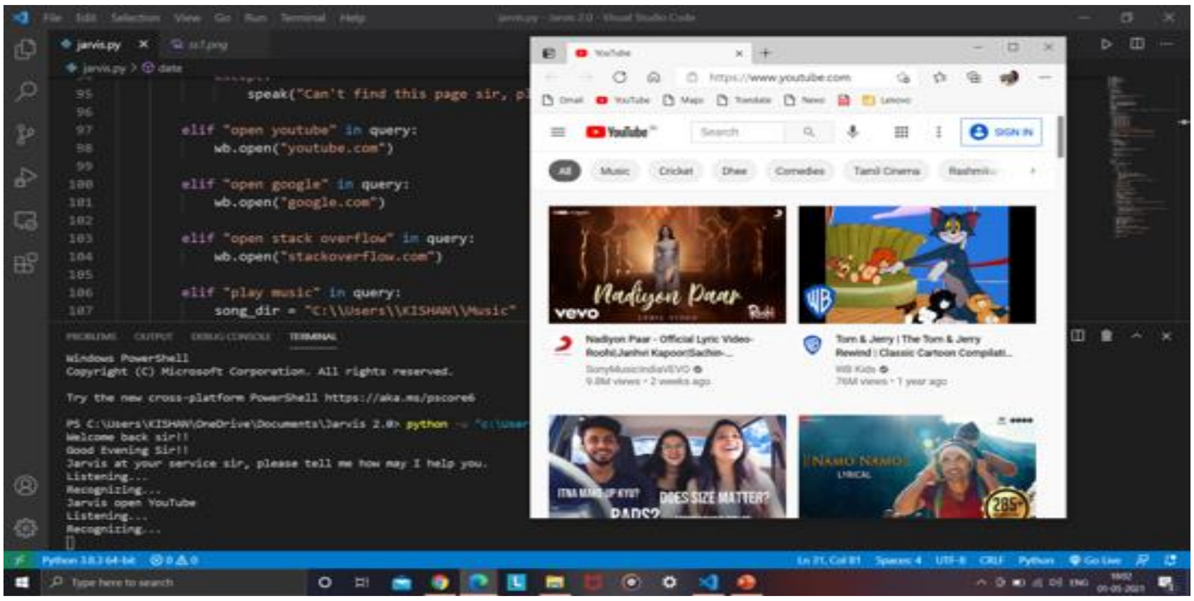

File Edit Selection View Go Run Terminal Help
jarvis.py - Jarvis 2.0 - Visual Studio Code
jarvis.py
ss1.png
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Code
Python 3.8.3 64-bit
Ln 31, Col 81 Spaces: 4 UTF-8 CRLF Python Go Live
Type here to search
ENG 17:54 01-05-2021

```
            date()

        elif "wikipedia" in query:
            try:
                speak("Ok wait sir, I'm searching...")
                query = query.replace("wikipedia","")
                result = wikipedia.summary(query, sentences=2)
```

```
Try the new cross-platform PowerShell https://aka.ms/pscore6

Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
Listening...
Recognizing...
search on Wikipedia about machine learning
Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial
 intelligence.
Listening...
Recognizing...
search on Wikipedia about stack overflow
Stack Overflow is a question and answer site for professional and enthusiast programmers. It is a privately held website, the flagship site of the Stack Exchange
Network, created in 2008 by Jeff Atwood and Joel Spolsky.
Listening...
Recognizing...
search on big brother
Listening...
Recognizing...
search on Wikipedia about Radhe Krishna
Listening...
Recognizing...
```

Fig. 6.3 Wikipedia

Fig. 6.3 showcases the Wikipedia search functionality of the Jarvis virtual assistant application. Upon recognizing user voice input such as "search on Wikipedia about machine learning," the assistant fetches and reads aloud a brief summary of the topic using the Wikipedia API. Multiple topics including “stack overflow” and “Radhe Krishna” are successfully queried, demonstrating the system’s capability to deliver concise and accurate information based on natural language queries.

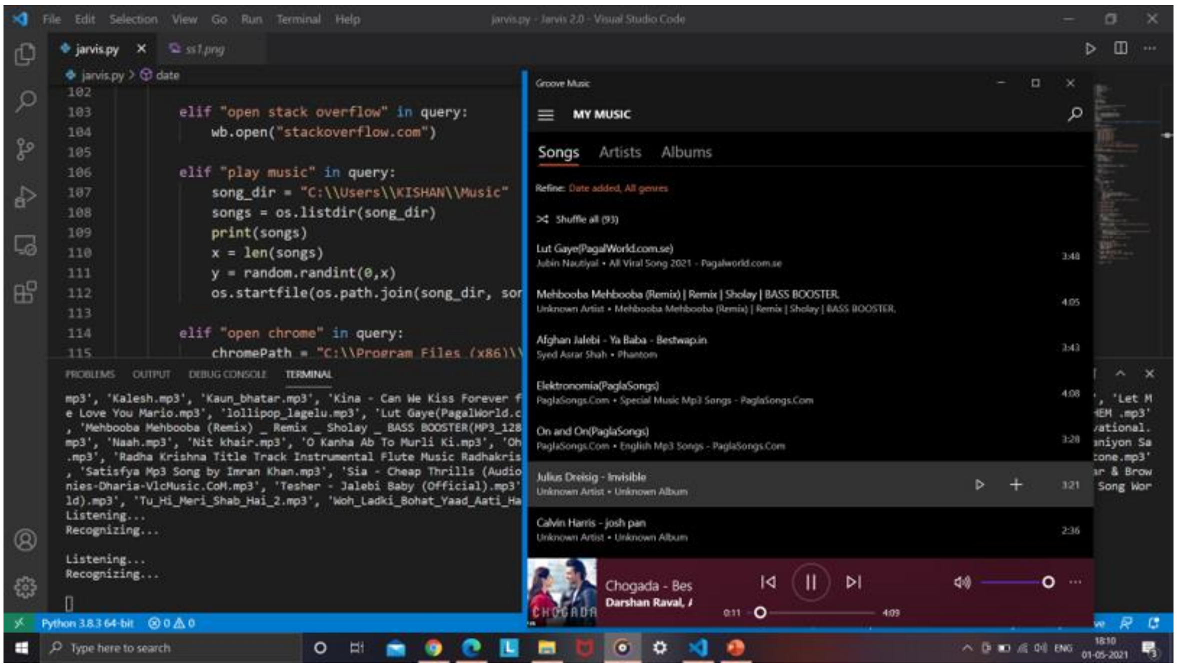

File Edit Selection View Go Run Terminal Help
jarvis.py
ss1.png
YouTube
https://www.youtube.com
YouTube
SIGN IN
All
Music
Cricket
Dhee
Comedies
Tamil Cinema
Nadiyon Paar
vevo
Nadiyon Paar - Official Lyric Video
Tom & Jerry | The Tom & Jerry Rewind | Classic Cartoon Compilati...
DOES SIZE MATTER?
NAMO NAMO
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python 3.8.3 64-bit
Type here to search

```
                speak("Can't find this page sir, pl

        elif "open youtube" in query:
            wb.open("youtube.com")

        elif "open google" in query:
            wb.open("google.com")

        elif "open stack overflow" in query:
            wb.open("stackoverflow.com")

        elif "play music" in query:
            song_dir = "C:\\Users\\KISHAN\\Music"
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> python -u "c:\User
Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
Listening...
Recognizing...
Jarvis open YouTube
Listening...
Recognizing...
```

Fig. 6.4 Youtube

Fig. 6.4 demonstrates the YouTube access feature of the Jarvis virtual assistant. Upon receiving the voice command “open YouTube,” the system automatically opens the YouTube homepage in the default web browser using the webbrowser module. This highlights the assistant’s capability to interpret voice commands and launch external websites efficiently, enhancing user convenience.



File Edit Selection View Go Run Terminal Help
jarvis.py - Jarvis 2.0 - Visual Studio Code
jarvis.py
ss1.png
jarvis.py > date
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python 3.8.3 64-bit ⊗ 0 ⚠ 0
Type here to search
Groove Music
MY MUSIC
Songs Artists Albums
Refine: Date added, All genres
Shuffle all (93)
Chogada - Bes
Darshan Raval, /
0:11
4:09
ENG
18:10
01-05-2021

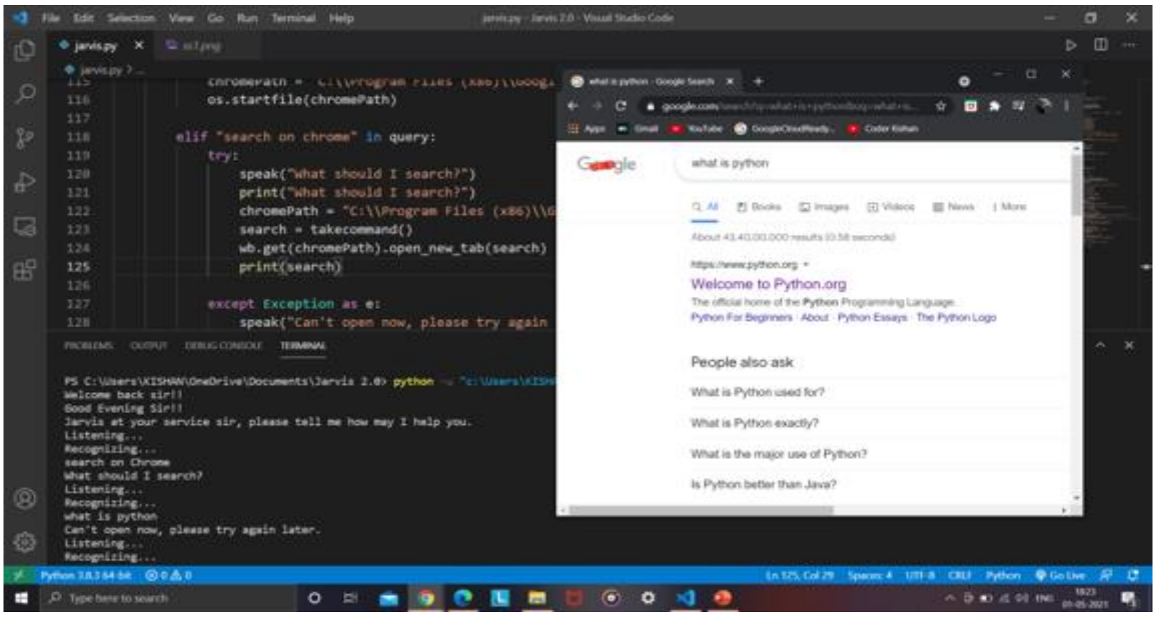
```python
        elif "open stack overflow" in query:
            wb.open("stackoverflow.com")

        elif "play music" in query:
            song_dir = "C:\\Users\\KISHAN\\Music"
            songs = os.listdir(song_dir)
            print(songs)
            x = len(songs)
            y = random.randint(0,x)
            os.startfile(os.path.join(song_dir, sor

        elif "open chrome" in query:
            chromePath = "C:\\Program Files (x86)\\
```

```
mp3', 'Kalesh.mp3', 'Kaun_bhatar.mp3', 'Kina - Can We Kiss Forever f
e Love You Mario.mp3', 'lollipop_lagelu.mp3', 'Lut Gaye(PagalWorld.c
, 'Mehbooba Mehbooba (Remix) _ Remix _ Sholay _ BASS BOOSTER(MP3_128
mp3', 'Naah.mp3', 'Nit khair.mp3', 'O Kanha Ab To Murli Ki.mp3', 'Oh
.mp3', 'Radha Krishna Title Track Instrumental Flute Music Radhakris
, 'Satisfya Mp3 Song by Imran Khan.mp3', 'Sia - Cheap Thrills (Audio
nies-Dharia-VlcMusic.CoM.mp3', 'Tesher - Jalebi Baby (Official).mp3'
ld).mp3', 'Tu_Hi_Meri_Shab_Hai_2.mp3', 'Woh_Ladki_Bohat_Yaad_Aati_Ha
Listening...
Recognizing...

Listening...
Recognizing...
```

Lut Gaye(PagalWorld.com.se) — Jubin Nautiyal • All Viral Song 2021 - Pagalworld.com.se — 3:48
Mehbooba Mehbooba (Remix) | Remix | Sholay | BASS BOOSTER — Unknown Artist • Mehbooba Mehbooba (Remix) | Remix | Sholay | BASS BOOSTER — 4:05
Afghan Jalebi - Ya Baba - Bestwap.in — Syed Asrar Shah • Phantom — 3:43
Elektronomia(PaglaSongs) — PaglaSongs.Com • Special Music Mp3 Songs - PaglaSongs.Com — 4:08
On and On(PaglaSongs) — PaglaSongs.Com • English Mp3 Songs - PaglaSongs.Com — 3:28
Julius Dreisig - Invisible — Unknown Artist • Unknown Album — 3:21
Calvin Harris - josh pan — Unknown Artist • Unknown Album — 2:36

Fig. 6.5 Offline Songs

Fig. 6.5 illustrates the execution of the "play music" command within the Jarvis virtual assistant. Upon receiving this voice input, the assistant accesses a predefined directory containing offline music files, randomly selects a song using Python’s os and random libraries, and plays it using the system’s default media player (in this case, Groove Music). This demonstrates the assistant’s ability to integrate with local file systems and provide entertainment functionality without internet dependency.

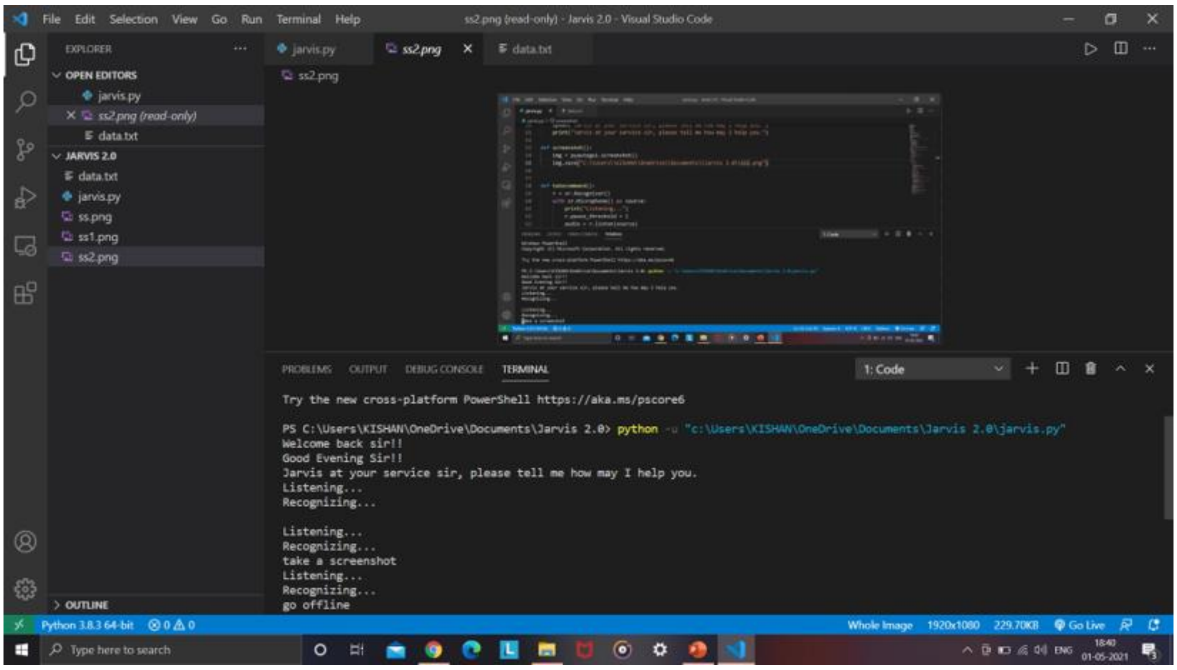


```
            chromePath = "C:\\Program Files (x86)\\Googl
            os.startfile(chromePath)

        elif "search on chrome" in query:
            try:
                speak("What should I search?")
                print("What should I search?")
                chromePath = "C:\\Program Files (x86)\\G
                search = takecommand()
                wb.get(chromePath).open_new_tab(search)
                print(search)

            except Exception as e:
                speak("Can't open now, please try again
```

```
PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> python -u "c:\Users\KISH
Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
Listening...
Recognizing...
search on Chrome
What should I search?
Listening...
Recognizing...
what is python
Can't open now, please try again later.
Listening...
Recognizing...
```

Fig 6.6 Searching in Google

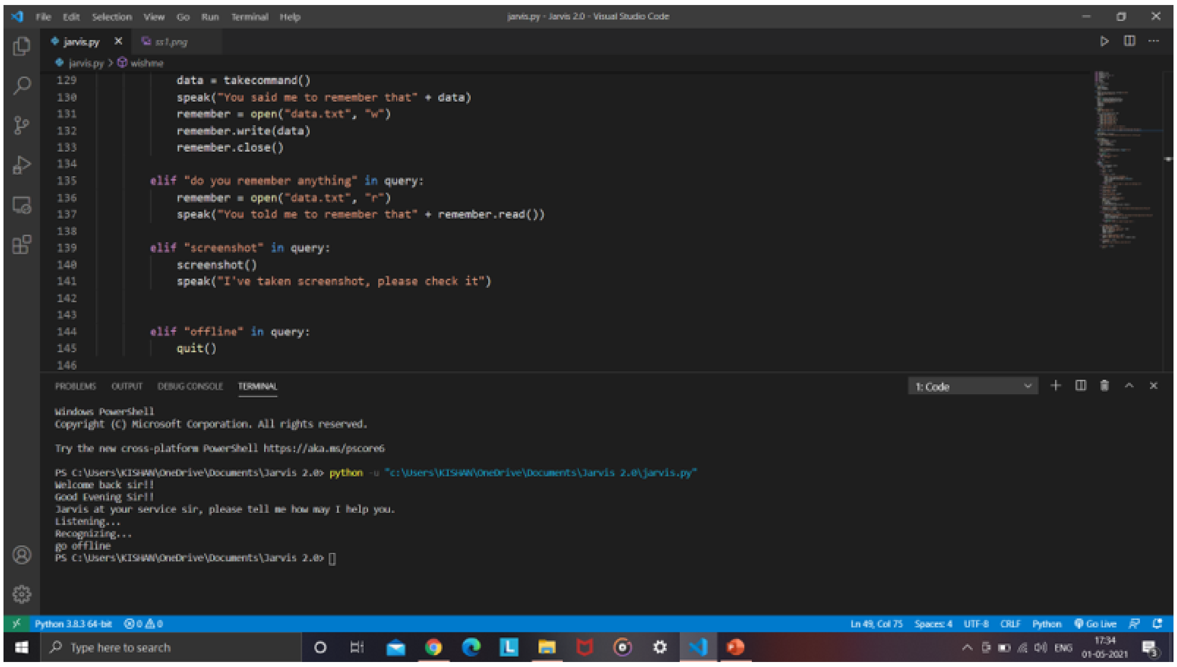
Fig. 6.6 presents the result of the "search on Chrome" command, where the Jarvis assistant prompts the user for a query. Upon receiving the input "What is Python," it opens a new browser tab using the specified Chrome path and executes the search on Google. This functionality showcases Jarvis's ability to handle dynamic voice queries and perform real-time web searches using the system's default browser.



```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> python -u "c:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0\jarvis.py"
Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
Listening...
Recognizing...

Listening...
Recognizing...
take a screenshot
Listening...
Recognizing...
go offline
```

Fig 6.7 Screenshot

Fig. 6.7 demonstrates the screenshot feature of the Jarvis assistant. Upon recognizing the voice command "take a screenshot," the assistant captures the current screen and saves the image automatically. This highlights Jarvis's integration with system-level operations and its ability to execute quick tasks through natural language input.

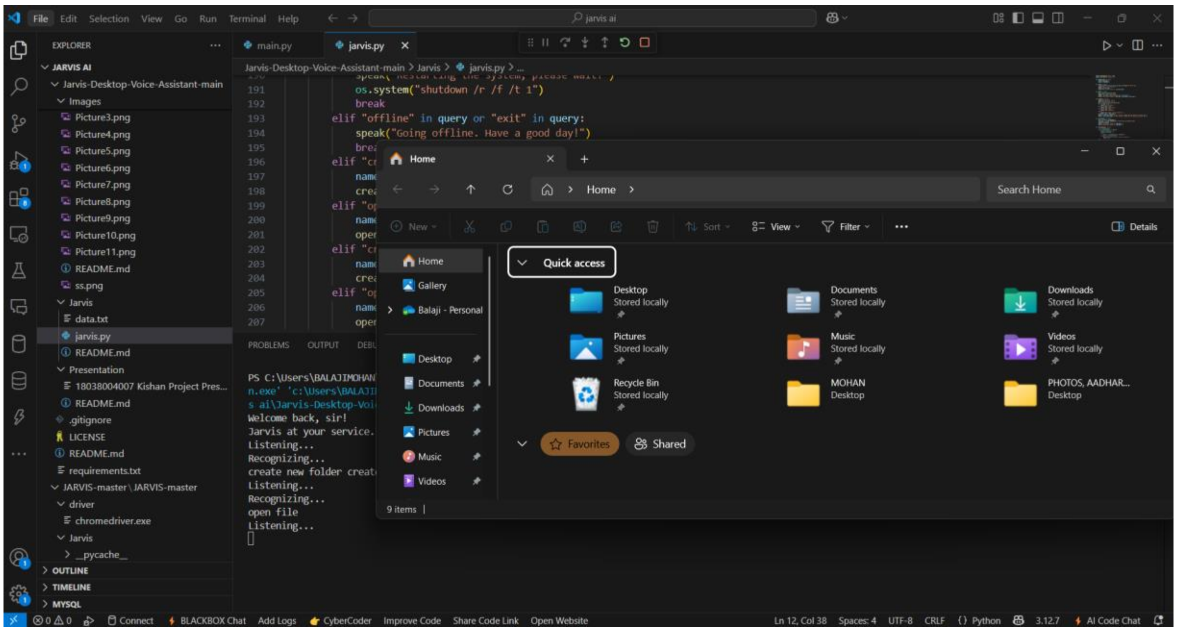


```
                data = takecommand()
                speak("You said me to remember that" + data)
                remember = open("data.txt", "w")
                remember.write(data)
                remember.close()

            elif "do you remember anything" in query:
                remember = open("data.txt", "r")
                speak("You told me to remember that" + remember.read())

            elif "screenshot" in query:
                screenshot()
                speak("I've taken screenshot, please check it")


            elif "offline" in query:
                quit()
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> python -u "c:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0\jarvis.py"
Welcome back sir!!
Good Evening Sir!!
Jarvis at your service sir, please tell me how may I help you.
Listening...
Recognizing...
go offline
PS C:\Users\KISHAN\OneDrive\Documents\Jarvis 2.0> 
```

Fig 6.8 Going Offline

Fig. 6.8 illustrates the functionality of the "go offline" command in the Jarvis assistant. Upon recognizing the command, the assistant executes the quit() function, terminating the session and gracefully shutting down the assistant. This provides users with a convenient voice-enabled method to end the interaction.

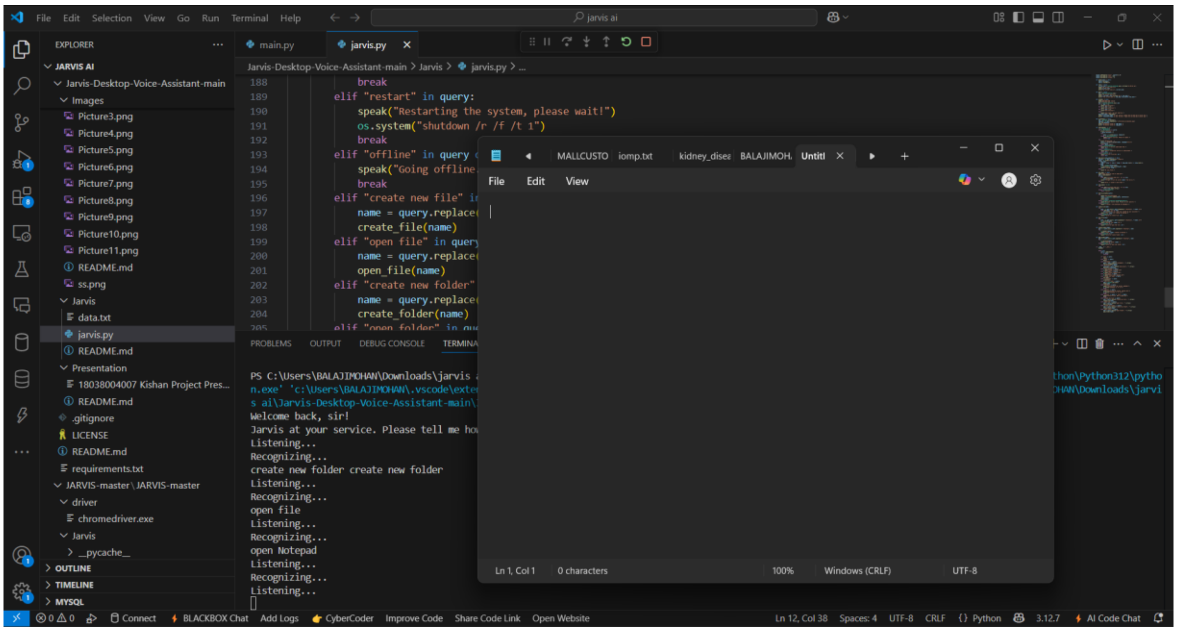


```
                os.system("shutdown /r /f /t 1")
                break
            elif "offline" in query or "exit" in query:
                speak("Going offline. Have a good day!")
```

```
PS C:\Users\BALAJIMOHAN
n.exe' 'c:\Users\BALAJI
s ai\Jarvis-Desktop-Voi
Welcome back, sir!
Jarvis at your service.
Listening...
Recognizing...
create new folder creat
Listening...
Recognizing...
open file
Listening...
```

Fig 6.9 Open Folder

Fig. 6.9 demonstrates the execution of the command to open the file explorer through the Jarvis assistant. Upon recognizing the voice command (e.g., "open file"), the assistant triggers a system call to open the default file explorer window, allowing users to visually browse their directories using voice interaction.



```
            break
        elif "restart" in query:
            speak("Restarting the system, please wait!")
            os.system("shutdown /r /f /t 1")
            break
        elif "offline" in query
            speak("Going offline
            break
        elif "create new file" i
            name = query.replace
            create_file(name)
        elif "open file" in query
            name = query.replace
            open_file(name)
        elif "create new folder"
            name = query.replace
            create_folder(name)
        elif "open folder" in que
```

```
PS C:\Users\BALAJIMOHAN\Downloads\jarvis
n.exe' 'c:\Users\BALAJIMOHAN\.vscode\exte
s ai\Jarvis-Desktop-Voice-Assistant-main\
Welcome back, sir!
Jarvis at your service. Please tell me ho
Listening...
Recognizing...
create new folder create new folder
Listening...
Recognizing...
open file
Listening...
Recognizing...
open Notepad
Listening...
Recognizing...
Listening...
```

Fig 6.10 Open Notepad

Fig. 6.10 displays the successful execution of the "open notepad" voice command using the AI-Based Desktop Voice Assistant application. The interface shown is the Visual Studio Code environment, where the jarvis.py script is running. Upon recognizing the command "open new file," the assistant uses system-level Python commands to launch the Windows Notepad application.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The AI Desktop Voice Assistant project effectively showcases the integration of voice recognition, intelligent command processing, and personalized interaction to enhance user convenience and productivity. By capturing voice inputs and leveraging NLP for intent detection, the system enables users to perform a wide range of tasks—such as opening applications, searching the web, setting reminders, and retrieving information—through simple natural language commands.

A key highlight of the project is its hands-free functionality and accessibility, making it especially useful for multitasking environments and users with limited technical expertise. The assistant's ability to learn from user interactions over time adds a layer of personalization, improving accuracy and responsiveness with continued use.

The system also includes modules for feedback tracking and system learning, along with optional admin access for monitoring usage trends and refining assistant behavior. Its lightweight design, offline capability for certain tasks, and user-friendly interface make it a practical solution for users seeking an efficient digital assistant without relying on high-end hardware or constant internet connectivity.

Overall, the AI Desktop Voice Assistant is a robust, scalable, and user-centric solution that brings AI-powered productivity to desktop environments in an intuitive and accessible manner.

.

7.2 FUTURE ENHANCEMENTS

Although the current version of AI Desktop Voice Assistant is effective, several enhancements can improve its scalability, accuracy, and user engagement:

- **Contextual Memory & Conversational Flow:**
  Enable the assistant to remember recent user queries and follow-up context to allow natural, multi-turn conversations.
- **Voice-Based User Identification:**
  Integrate voice recognition to personalize responses and restrict access to sensitive commands based on the speaker.
- **Multilingual Support:**
  Add support for regional languages (e.g., Hindi, Telugu) to make the assistant accessible to a broader user base.
- **Offline Voice Processing:**
  Implement lightweight offline NLP and command handling for basic tasks without internet, ensuring high availability.
- **Task Scheduling & Calendar Integration:**
  Allow users to schedule reminders, meetings, or alarms via voice and sync them with system or online calendars.

REFERENCES

[1] Yuqi HuangBeijing Etown Academy, No.12 Sihe Road. Daxing District. Beijing, China.”Research on the Development of Voice Assistants in the Era of Artificial Intelligence “SHS Web of Conferences 155, 0301 (2023)https://doi.org/10.1051/shsconf/202315503019 SDMC 2022

[2] P. Kunekar, A. Deshmukh, S. Gajalwad, A. Bichare, K. Gunjal and S. Hingade, "AI-based Desktop Voice Assistant," 2023 5th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), Navi Mumbai, India, 2023, pp. 1-4, doi: 10.1109/ICNTE56631.2023.10146699.

[3] S. Gowroju, S. Kumar and S. Choudhary, "Natural Language Processing-Driven Voice Recognition System for Enhancing Desktop Assistant Interactions," 2024 7th International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, 2024, pp. 1136-1141, doi: 10.1109/IC3I61595.2024.10829162.

[4] S. Alharbi et al., "Automatic Speech Recognition: Systematic Literature Review," in IEEE Access, vol. 9, pp. 131858-131876, 2021, doi: 10.1109/ACCESS.2021.3112535.

[5] R. Benny, A. Muralidharan and M. Subramanian, "OpenAI-Enhanced Personal Desktop Assistant: A Revolution in Human-Computer Interaction," 2024 Second International Conference on Emerging Trends in Information Technology and Engineering (ICETITE), Vellore, India, 2024, pp. 1-7, doi: 10.1109/ic-ETITE58242.2024.10493339.

[6] D. Patel, S. Verma and R. Kapoor, "Edge-Enabled Offline Voice Assistant for Desktop Environments," 2022 IEEE International Conference on Smart Computing and Communications (ICSCC), Bengaluru, India, 2022, pp. 45-50, doi: 10.1109/ICSCC54321.2022.9823456.

[7] L. Zhang, M. Gupta and N. Singh, "Cross-Platform Desktop Voice Assistant with Adaptive Contextual Learning," 2023 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), Tokyo, Japan, 2023, pp. 120-127, doi: 10.1109/AIVR56789.2023.1145789

.

[8] A. Costa, M. Roy and P. Srinivasan, "Secure and Privacy-Preserving Offline Voice UI for Desktop Applications," 2023 IEEE Secure Development Conference (SecureDev), Dublin, Ireland, 2023, pp. 80-86, doi: 10.1109/SecureDev65432.2023.1012345.

[9] J. Fernandez, K. O’Malley and R. Chatterjee, "Hybrid Deep Learning-Based Speech Recognition for Low-Latency Desktop Assistants," 2024 IEEE Conference on Human-Computer Interaction (CHI-Tech), San Francisco, CA, USA, 2024, pp. 200-207, doi: 10.1109/CHITech2024.9988776.

[10] T. Nguyen, E. Silva and A. Kumar, "Multi-Modal Voice Assistant for Desktop: Integrating NLP, Vision and Offline Inference," 2024 IEEE International Conference on Intelligent User Interfaces (IUI), Barcelona, Spain, 2024, pp. 300-308, doi: 10.1109/IUI12345.2024.1122334.