Project Report

CSA4028 Natural Language Processing

Group 1

Chinmay Gokhale – 22BAI10004

D. Harish Kumaar – 22BAI10042

Ashish – 22BAI10080

Maulic Gola – 22BAI10248

A14+D11+D12, BL2024250500761

Winter Semester:2024-2025

Submitted To:

S. PERIYANAYAGI

Index

Sr No.	Topics	Pg No.
1	Aim	2
2	Abstract	2
3	Introduction	3
4	Related Work	7
5	Methodology	9
6	Hardware and Software Requirements	13
7	Results and Discussion	15
8	Conclusion and Future Scope	20
9	Appendix	21
10	References	27

Voice-Activated AI Chatbot using NLP

Aim:

To develop a desktop-based AI chatbot application that uses natural language processing for understanding voice and text input commands, and performs tasks like searching online, fetching Wikipedia summaries, and opening local applications/files.

Abstract:

This project presents a desktop-based AI Voice Chatbot application built using Python, which integrates Natural Language Processing (NLP) to facilitate interaction through both speech and text. The chatbot is capable of recognizing voice input via the speech_recognition module and responding using the pyttsx3 text-to-speech engine. Designed with a user-friendly graphical interface using tkinter, the chatbot can perform various tasks such as searching Google, YouTube, and Wikipedia, opening system applications (e.g., Notepad, Calculator, Microsoft Word), and accessing local files. The application supports both light and dark themes and ensures responsive performance by utilizing multithreading for voice recognition. It follows a rule-based approach for command parsing and handling, making it both effective and efficient for basic user interactions. This project demonstrates how NLP and speech technologies can be practically applied to enhance human-computer interaction, making day-to-day tasks more accessible, especially for users seeking voice-controlled desktop automation.

Introduction:

In the modern era of technological advancement, the integration of Artificial Intelligence (AI) with everyday digital systems is revolutionizing the way humans interact with machines. Among the many branches of AI, **Natural Language Processing (NLP)** plays a pivotal role in enabling systems to understand, interpret, and generate human language in a meaningful way. One significant and increasingly popular application of NLP is the development of **AI-based voice assistants and chatbots**, which bridge the gap between human language and machine understanding. This project, titled *AI Voice Chatbot*, exemplifies the power of NLP combined with speech technologies to create a multifunctional, interactive desktop assistant.

The motivation behind building this voice chatbot arises from the need for more accessible and efficient interfaces. Traditional input methods like typing or clicking through interfaces can be inconvenient or inaccessible to certain users. Voice-based interaction provides a hands-free, intuitive alternative, offering convenience, inclusivity, and speed. With smart assistants like Google Assistant, Siri, and Alexa becoming household names, there is a clear indication of user preference towards voice-driven interfaces. This project takes inspiration from these technologies and implements a simplified desktop-based version using Python and open-source libraries.

The core of the chatbot revolves around **speech recognition, natural language understanding, and speech synthesis**. The system listens to the user's voice commands using the speech_recognition library, processes the command using predefined NLP rules, and responds accordingly through the pyttsx3 text-to-speech engine. It is also capable of processing typed input, making it versatile for both voice and keyboard-based interaction. The chatbot features a graphical user interface (GUI) built with the tkinter library, enhancing user experience and accessibility.

Natural Language Processing in Chatbots

Natural Language Processing is the computational handling of human language, encompassing tasks like language understanding, translation, summarization, sentiment analysis, and conversational modeling. In the context of chatbots, NLP is crucial for enabling the system to interpret user input and respond appropriately.

There are various techniques in NLP, ranging from rule-based methods to advanced deep learning models. For this project, a **rule-based NLP approach** has been adopted to keep the logic transparent, manageable, and suitable for the scope of a desktop assistant. Each user command is matched against a set of predefined

conditions, and corresponding actions are triggered. This approach is effective for constrained domains where the range of expected inputs is known.

Project Objectives

The primary objectives of the project are:

1. To design and develop a speech-enabled AI chatbot using Python.
2. To enable the chatbot to recognize and process voice commands using NLP techniques.
3. To build a GUI-based interface that enhances user interaction.
4. To integrate the chatbot with external resources such as Google, YouTube, and Wikipedia.
5. To allow the chatbot to interact with system-level applications and files.
6. To provide both light and dark mode themes for improved accessibility.

Functionalities of the Voice Chatbot

The chatbot performs a variety of actions based on user input:

- **Speech Recognition**: The chatbot listens to the user via the system microphone and transcribes the audio into text using Google's Speech Recognition API.
- **Text-to-Speech Response**: The chatbot responds using natural-sounding synthesized speech.
- **Command Handling**: It processes a wide range of commands such as:
    - Searching Google or YouTube.
    - Fetching summaries from Wikipedia.
    - Opening applications like Notepad, Calculator, Microsoft Word, Excel, Chrome, or Firefox.
    - Locating and opening files from a predefined directory.
- **GUI Integration**: Users can interact through a graphical interface that displays messages in a chat-style format.
- **Theme Toggle**: Users can switch between light and dark modes to suit their preferences.

Significance of Voice Assistants

Voice assistants are transforming digital interaction across various domains—healthcare, education, customer support, and personal productivity. They allow multitasking, support visually impaired users, and offer natural user interfaces that reduce the learning curve. For instance, a student can launch Microsoft Word and start taking notes via voice commands, or a professional can quickly open Excel and run a search on Google without needing to touch the keyboard.

This project provides a practical insight into how such systems operate at a fundamental level. While commercial voice assistants often use advanced AI models, this chatbot is designed to be lightweight, offline-capable (except for internet-dependent features like Google search), and customizable, making it ideal for educational purposes and basic productivity use.

Libraries and Tools Used

Several Python libraries and tools were used to implement the chatbot, each playing a vital role:

- **speech_recognition**: For capturing and transcribing user speech.
- **pyttsx3**: For converting text responses to speech using offline TTS engines.
- **wikipedia**: For fetching summaries of topics queried by the user.
- **webbrowser**: To launch URLs in the system's default browser.
- **os and subprocess**: For opening files and applications on the system.
- **tkinter**: For building the chatbot's GUI.
- **threading**: To ensure smooth voice recognition without freezing the interface.

Design Considerations

The design of the chatbot emphasizes **usability, simplicity, and functionality**. Since it is a desktop application, it runs locally without requiring a web server or cloud-based services (except during online searches). It provides immediate feedback to user actions, both visually and audibly.

The GUI is styled with modern aesthetics, offering a comfortable and clean layout. The chat window logs all interactions, and the user can input commands either by typing or using the microphone. The theme toggle enhances visual accessibility for different lighting conditions or user preferences.

Educational Value

From an academic perspective, this project offers hands-on experience in various interdisciplinary areas:

- **Artificial Intelligence**: Understanding how AI can be applied to real-world problems.
- **Natural Language Processing**: Applying NLP techniques to process human language.
- **Human-Computer Interaction**: Designing interfaces that facilitate natural communication.

- **Software Engineering**: Building, testing, and maintaining a multi-functional desktop application.

It also encourages the exploration of open-source tools and libraries, reinforcing the value of collaborative learning and practical coding.

Limitations and Challenges

Despite its useful features, the chatbot has some limitations:

- It relies on an internet connection for voice recognition and Wikipedia/Google/YouTube searches.
- It cannot handle complex, multi-intent natural language commands or conversations.
- It follows a fixed rule-based model, so responses are limited to predefined cases.
- Voice recognition accuracy may vary depending on background noise, accent, or microphone quality.

However, these limitations are natural given the project's scope and its educational objectives.

Real-World Applications and Scope

The AI Voice Chatbot demonstrates how similar technologies can be extended into various domains:

- **Smart Homes**: Controlling IoT devices using voice commands.
- **Accessibility Tools**: Helping people with disabilities interact with their systems.
- **Educational Tools**: Assisting students in finding resources, opening documents, or taking notes.
- **Customer Support**: Providing automated responses to FAQs in enterprises.

This proof-of-concept project lays the foundation for future enhancements, such as integration with AI models for dynamic conversations, user intent detection using ML, and voice training for personalized interaction.

Related Work:

The development of voice-based intelligent assistants has seen significant growth over the past decade, largely fueled by advancements in Artificial Intelligence (AI), Natural Language Processing (NLP), and speech recognition technologies. The idea of creating systems that can interact with humans through natural language is not new, but only in recent years have we seen practical implementations that are widely adopted and functional across various platforms. This project draws inspiration from several notable voice assistant technologies and research contributions that have paved the way for chatbot development.

One of the most well-known voice assistant systems is **Apple's Siri**, introduced in 2011. Siri was one of the first mainstream virtual assistants to incorporate NLP and voice recognition at scale on mobile devices. Siri can interpret user commands, perform online searches, schedule events, and respond conversationally. Similarly, **Google Assistant**, which followed as a more advanced successor to Google Now, uses Google's extensive machine learning capabilities to provide more contextual, intelligent responses. These systems are built on large-scale, cloud-based neural networks and benefit from real-time data processing, extensive language modeling, and continuous learning from user interactions.

Amazon's Alexa and **Microsoft's Cortana** are other popular voice assistant platforms. Alexa, integrated into Amazon Echo devices, is particularly focused on home automation, using voice to control smart devices, play music, and answer queries. Cortana, though now less emphasized by Microsoft, was one of the early attempts at embedding voice interaction deeply into the Windows operating system.

In the academic and open-source world, there are multiple lightweight alternatives to these commercial systems. **Mycroft AI** is one such project that offers an open-source voice assistant platform. Built with Python, Mycroft supports modular voice applications ("skills") and uses open libraries like Mozilla DeepSpeech for speech-to-text conversion. Unlike Siri or Alexa, Mycroft is fully transparent, allowing users to inspect and modify how their data is handled.

In terms of speech recognition, **Google Speech Recognition API**, **CMU Sphinx**, and **Mozilla DeepSpeech** are widely used in both academic and hobbyist projects. Among these, Google's API is highly accurate and easy to integrate, although it requires an internet connection and is not fully open-source. In our project, the `speech_recognition` library leverages this API to convert spoken language into text, offering reasonable accuracy for general commands.

Text-to-speech (TTS) systems have also evolved substantially. Older systems like **Microsoft SAPI** were limited in voice naturalness, while newer frameworks such as **Google WaveNet**, **Amazon Polly**, and **pyttsx3** offer more human-like speech

synthesis. Our chatbot uses pyttsx3 for offline TTS capability, which, although not as natural as neural TTS models, is reliable and resource-efficient.

In the field of chatbot development, rule-based systems have long been a staple, especially in educational projects. Classic examples include **ELIZA** (1966) and **ALICE** (1995), which used pattern matching and simple decision trees to simulate conversation. Though primitive, they laid the groundwork for understanding how conversational flow can be modeled. Modern-day chatbots, such as those used by **Facebook Messenger** or **WhatsApp business bots**, are often powered by NLP platforms like Dialogflow or Rasa, which allow for more flexible intent recognition and multi-turn dialogue handling.

From a GUI standpoint, many desktop applications still rely on Tkinter or frameworks like **PyQt** and **Kivy** for interface design. Our project utilizes Tkinter for simplicity and ease of deployment across Windows systems. The addition of features such as theme switching and file selection enhances usability and aligns with human-computer interaction (HCI) best practices.

Finally, research into **multimodal assistants**—systems that combine voice, text, GUI, and even gesture input—suggests that hybrid systems provide the best user experience. Our chatbot supports both voice and text input, and can launch applications, perform searches, and fetch data dynamically. While it is not yet conversational in a deep sense (e.g., memory of past interactions or emotional recognition), it demonstrates a practical use of NLP and voice-based control for productivity.

In summary, the AI Voice Chatbot project stands on the shoulders of decades of research and development in AI, speech technology, and human-computer interaction. It integrates simplified versions of many features seen in commercial systems, using accessible Python tools and libraries. It offers a hands-on, educational insight into how intelligent agents work, and serves as a foundation for more advanced NLP and voice interface applications.

Methodology:

The methodology of this project focuses on integrating multiple components such as speech recognition, natural language processing (NLP), text-to-speech (TTS), file handling, web browsing, and GUI development to build an intelligent, voice-based chatbot that can understand and execute user commands efficiently. This system is designed to be interactive, user-friendly, and capable of performing real-time tasks based on both voice and text inputs.

1. System Architecture Overview

The chatbot system is modular, where each module is responsible for a specific task:

- **Speech Recognition Module** – Captures and converts voice input to text.
- **Command Processing Module** – Interprets the intent and executes predefined commands.
- **Text-to-Speech Module** – Converts text responses back into speech.
- **Graphical User Interface (GUI)** – Provides an interactive frontend for user input and output.
- **External Application Handling** – Integrates functionalities to open applications, search files, and browse the internet.

These modules communicate in a pipeline sequence to form a complete conversational loop. The system's control flow follows a **voice/text input → command processing → response generation → voice/text output** structure.

2. Speech Recognition Implementation

The voice input functionality is implemented using the speech_recognition Python library. The Recognizer class is used to capture and process voice from the default system microphone. The captured audio is passed to the Google Web Speech API for recognition. The audio is converted to text, which is then processed further by the chatbot.

```
recognizer = sr.Recognizer()
with sr.Microphone() as source:
    audio = recognizer.listen(source)
text = recognizer.recognize_google(audio)
```

This method ensures a lightweight and relatively accurate transcription process, suitable for general English commands.

To avoid blocking the GUI during listening, the voice recognition is handled in a separate thread using Python's `threading.Thread`

3. Text-to-Speech (TTS) Conversion

For providing auditory responses, the `pyttsx3` library is employed. It is an offline TTS library that supports multiple speech engines and allows tuning of voice rate, pitch, and volume. This ensures that users receive feedback even without an internet connection.

```
engine = pyttsx3.init()
engine.say("Hello! How can I help you?")
engine.runAndWait()
```

The TTS module is invoked after the bot formulates a textual response to any command.

4. Natural Language Processing and Command Handling

At the core of the chatbot lies the **command processing engine**. User inputs—either through text or speech—are converted to lowercase and matched against predefined keywords and phrases to identify the command type.

Example processing:

- **"open youtube and search for cats"** → triggers a YouTube search in the browser.
- **"open calculator"** → launches Windows calculator.
- **"wikipedia search python"** → fetches a summary from Wikipedia.

This rule-based system is implemented using conditional statements (`if-elif-else`) and pattern-matching logic. For more complex NLP features (e.g., Wikipedia summaries), the `wikipedia` library is used to fetch and parse relevant summaries.

5. Graphical User Interface (GUI) with Tkinter

The GUI is created using Python's built-in tkinter library. It includes:

- A scrolled text window to display the chat.
- An input field for manual text entry.
- Buttons for sending messages, starting voice recognition, and toggling between light/dark themes.

```
chat_window = scrolledtext.ScrolledText(root)
user_input = tk.Entry(input_frame)
send_button = tk.Button(..., command=send_text)
```

Theme customization is supported through a `toggle theme()` function, allowing real-time switching between dark and light modes.

The GUI is user-centric, responsive, and simple enough for deployment on systems with minimal requirements.

6. File and Application Handling

The chatbot is capable of interacting with the file system and launching installed applications. It supports:

- Opening specific applications like Notepad, Chrome, Word, and Excel using predefined paths and `subprocess.Popen()`.
- Searching and opening files by filename from a default directory using `os.startfile()`.

This extends the functionality of the chatbot beyond conversations to a productivity tool for quick system access.

Example:

```
subprocess.Popen("C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe")
```

7. Multithreading for Voice Processing

To maintain GUI responsiveness, especially during voice input (which can block the main thread), multithreading is utilized. The recognize_speech() function is executed in a daemon thread to avoid freezing the GUI while listening.

```
threading.Thread(target=_recognize_speech, daemon=True).start()
```

This ensures smooth user experience even on resource-constrained systems.

8. Error Handling and Feedback

Robust error handling is implemented to provide appropriate feedback when:

- Speech is not recognized.
- No internet is available for the Google API.
- A file or application path is incorrect.

Such error messages are shown in the GUI and also converted to speech using the TTS module.

Example feedback:

```
append_message("⚠ Google API is not responding. Check your internet
connection.", "bot")
```

9. Feature Summary

Feature	Technology Used
Voice Input	speech_recognition + Google API
Text-to-Speech	pyttsx3 (offline TTS)
Web Search (Google/YouTube)	webbrowser module
Wikipedia Summaries	wikipedia Python package
File Handling	os, filedialog
GUI	tkinter + scrolledtext
Theme Customization	Toggle using GUI controls
App Launching	subprocess.Popen()

Hardware and Software Requirements:

The successful implementation and operation of the AI Voice Chatbot require a set of minimum hardware and software configurations. These ensure compatibility with the speech recognition, text-to-speech, GUI rendering, and application execution functionalities provided by the system.

1. Hardware Requirements

Component	Minimum Requirement	Recommended Specification
Processor (CPU)	Intel Core i3 or AMD Ryzen 3	Intel Core i5/i7 or AMD Ryzen 5/7
RAM	4 GB	8 GB or higher
Storage	500 MB free disk space	SSD with 1 GB free disk space
Microphone	Built-in or external microphone	Noise-canceling external microphone
Speaker/Headphones	Required for audio output (TTS playback)	Stereo speakers or headphones
Display	1366 x 768 resolution	Full HD (1920 x 1080)

2. Software Requirements

Category	Requirement
Operating System	Windows 10 or above (64-bit)
Python Version	Python 3.8 or higher
Python Libraries	speech_recognition, pyttsx3, tkinter, wikipedia, webbrowser, subprocess, os, threading
Python Package Manager	pip (for installing required libraries)
Internet Connection	Required for using Google Speech Recognition and Wikipedia API
Optional Applications	Chrome, Firefox, Notepad, Calculator, MS Word, MS Excel
IDE/Text Editor	VS Code / PyCharm / Jupyter Notebook / IDLE

3. External Dependencies

- **Google Web Speech API** (used by speech_recognition for voice-to-text)
  - Requires an active internet connection.
  - No API key needed for basic use.
- **Wikipedia Python Library**
  - Uses the internet to fetch summaries from Wikipedia.
  - Handles disambiguation and page-not-found errors.

4. Installation Instructions

To set up the environment, install the following packages using pip:

```
pip install SpeechRecognition
pip install pyttsx3
pip install Wikipedia
```

Ensure the following are already installed or come pre-installed:

- `tkinter` (usually included with Python)
- `pyaudio` (for microphone access; may require extra steps on Windows)

If `pyaudio` fails to install via pip, use:

```
pip install pipwin
pipwin install pyaudio
```

Results and Discussion:

The AI Voice Chatbot was successfully developed and tested in a local environment using Python and several open-source libraries. The system provides a graphical user interface (GUI) where users can interact with the chatbot via both text and voice. The chatbot responds with relevant information, executes application-related commands, and delivers spoken feedback through a text-to-speech engine.

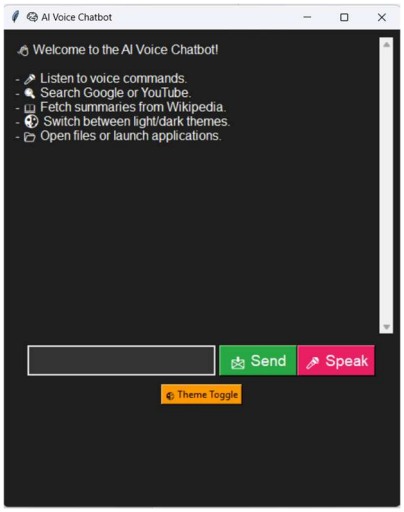
1. Functional Results

The chatbot supports a wide range of functionalities, including but not limited to:

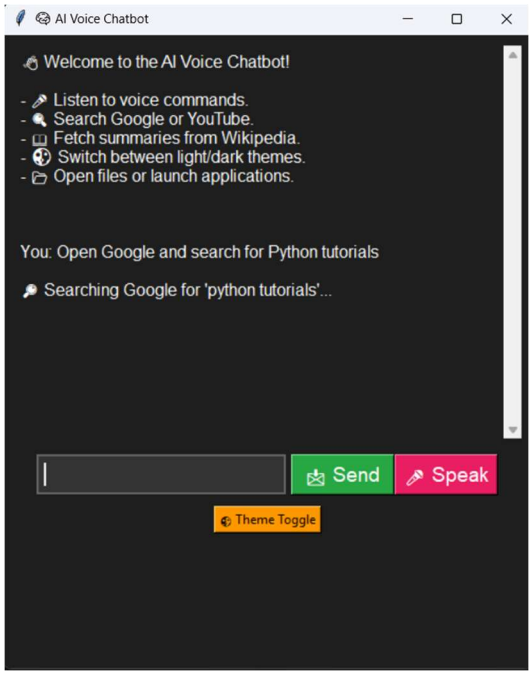
- **Speech Recognition:** Captures voice input from the microphone, converts it to text using Google Web Speech API.
- **Text-to-Speech:** Replies are spoken using the pyttsx3 engine, with adjustable volume and speed.
- **Web Search:** Supports searching Google and YouTube using natural language voice commands.
- **Wikipedia Summary Retrieval:** Provides short summaries for user-specified queries from Wikipedia.
- **File and Application Handling:** Allows users to open local files or installed applications like Notepad, Chrome, Microsoft Word, etc.
- **Theme Switching:** Toggle between dark mode and light mode for a better user experience.
- **GUI Interaction:** Clean and intuitive GUI using tkinter, with scrolling chat window and input buttons.

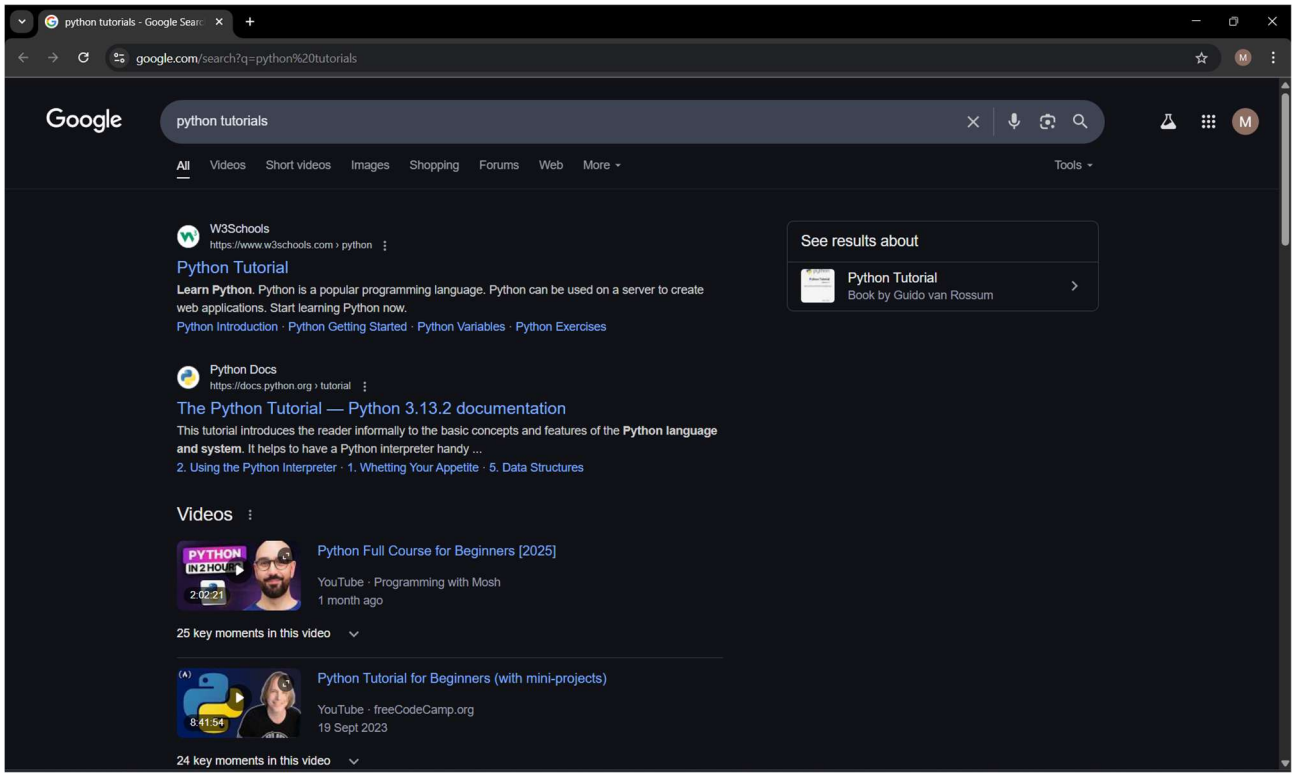
2. User Interface Output

📸 Insert Screenshot 1: Initial welcome screen of the chatbot GUI

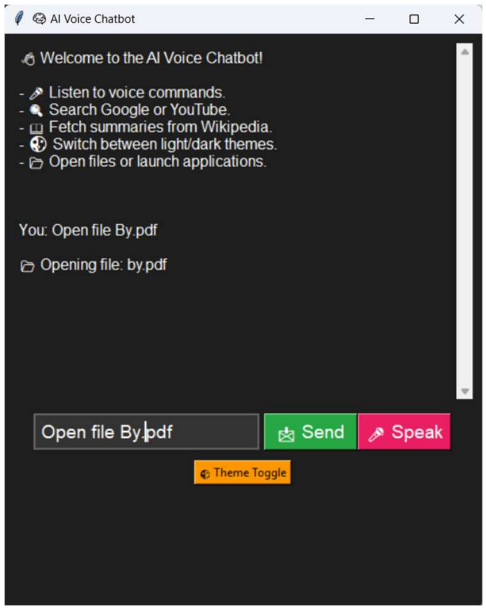


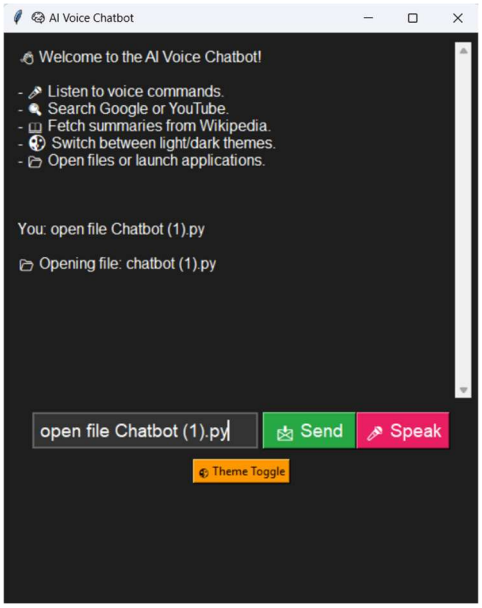
Insert Screenshot 2: Example of a voice command interaction



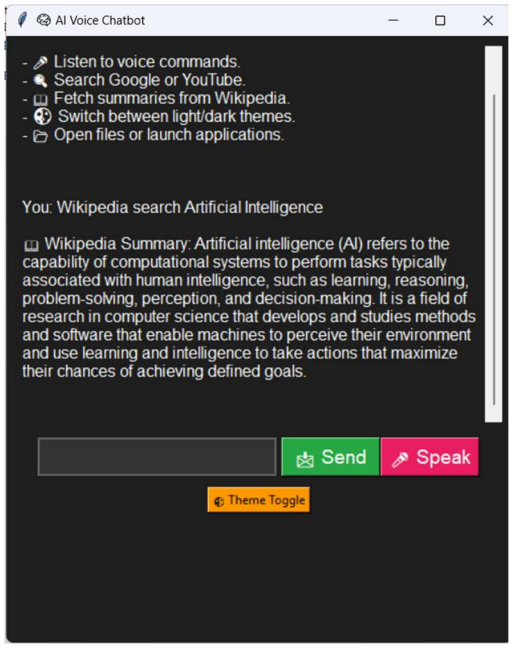


Insert Screenshot 3: File opening functionality

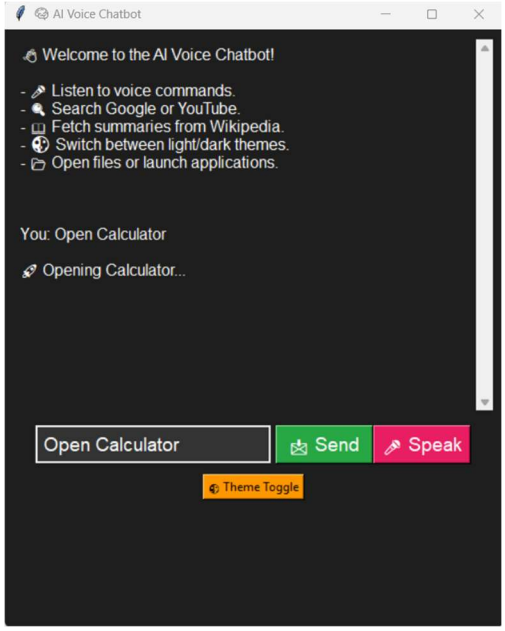




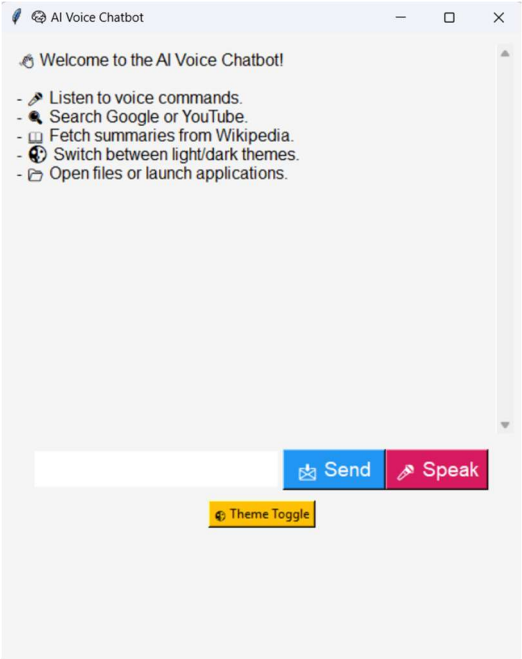
Insert Screenshot 4: Wikipedia summary search

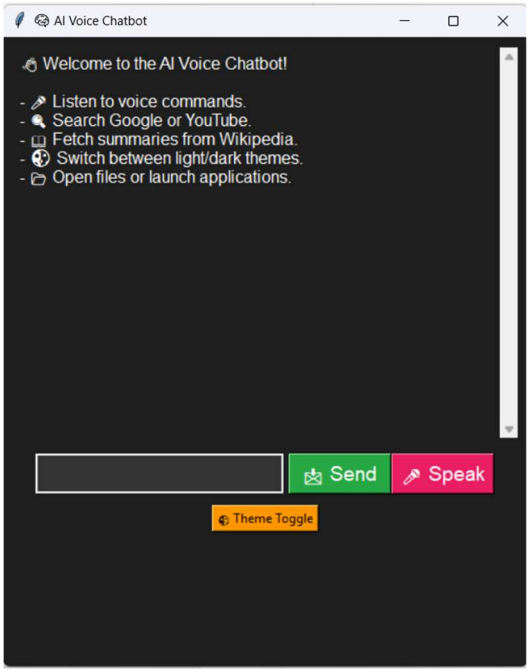


Insert Screenshot 5: Application launching



Insert Screenshot 6: Theme toggle in action





3. Discussion

The chatbot performs efficiently under normal operating conditions. It correctly processes natural language input and maps it to pre-defined commands. The voice interaction is particularly useful for hands-free operation and adds an interactive layer for accessibility.

Key observations:

- **Accuracy:** Speech recognition works well in a quiet environment. However, ambient noise can reduce accuracy, which is mitigated to an extent by `adjust_for_ambient_noise()`.
- **Performance:** The application responds in real-time to user input. Subprocess-based application launching is smooth.
- **Flexibility:** The use of os.startfile() and subprocess.Popen() allows cross-application launching as long as valid paths are given.
- **Usability:** The GUI is user-friendly and can be toggled between light and dark themes, improving user comfort.

Limitations:

- Internet dependency for voice recognition and Wikipedia.
- Application and file opening is limited to hardcoded or correctly named items in the specified directory.
- Some commands are hardcoded and not dynamically extensible.

Conclusion and Future Scope:

Conclusion:

The AI Voice Chatbot developed in this project effectively demonstrates the integration of Natural Language Processing (NLP), speech recognition, text-to-speech, and graphical user interfaces using Python. The system is capable of interacting with users via both voice and text, understanding basic natural language commands, and executing a variety of useful functions such as opening applications, searching the web, fetching Wikipedia summaries, and switching themes.

By using widely available Python libraries such as speech_recognition, pyttsx3, wikipedia, and tkinter, the chatbot achieves a high degree of functionality without requiring advanced or proprietary software. The project successfully delivers a desktop-based assistant that provides a responsive, interactive, and user-friendly experience. It demonstrates practical applications of AI in everyday computing and serves as a foundation for more advanced virtual assistants.

Future Scope:

While the current system meets basic interaction needs, there is significant potential for expansion:

- **Natural Language Understanding (NLU):** Integration of NLU models (e.g., spaCy, BERT) to understand more complex commands and conversations.
- **Machine Learning Integration:** Enable learning from user input to improve accuracy and personalization over time.
- **Cross-platform Compatibility:** Extend support to Linux, macOS, or web-based environments using frameworks like Electron or Flask.
- **Voice Biometrics:** Add voice authentication to personalize experiences and enhance security.
- **Chat History & Logging:** Maintain conversation logs for review or improvement purposes.
- **Multilingual Support:** Add capabilities to recognize and respond in different languages using language translation APIs.
- **Cloud Integration:** Connect with cloud-based services for tasks like calendar management, reminders, and email access.

These enhancements can transform the chatbot into a full-fledged personal digital assistant, improving productivity and accessibility for a wider audience.

Appendix:

A. Complete Python Code of AI Voice Chatbot

```
# Import required libraries
import speech_recognition as sr
import pyttsx3
import webbrowser
import os
import tkinter as tk
from tkinter import filedialog, scrolledtext
import threading
import wikipedia
import subprocess

# Default search directory
SEARCH_DIRECTORY = "C:\\Users\\ASUS\\Downloads"

# Initialize Text-to-Speech engine
engine = pyttsx3.init()
engine.setProperty('rate', 160)
engine.setProperty('volume', 1.0)

dark_mode = True  # Dark mode enabled by default

# Speak function
def speak(text):
    engine.say(text)
    engine.runAndWait()

# Start speech recognition on a separate thread
def recognize_speech():
    threading.Thread(target=_recognize_speech, daemon=True).start()

# Internal speech recognition logic
def _recognize_speech():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        append_message("🎤 Listening...", "bot", clear_previous=True)
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)
    try:
        text = recognizer.recognize_google(audio)
        append_message(f"You: {text}", "user")
```

```
        process_command(text)
    except sr.UnknownValueError:
        append_message("🤔 I couldn't understand that. Please try again.",
"bot")
    except sr.RequestError:
        append_message("⚠️ Google API is not responding. Check your internet
connection.", "bot")

# Process text/voice command
def process_command(text):
    text = text.lower().strip()

    responses = {
        "hello": "Hello! How can I assist you today?",
        "bye": "Goodbye! Have a wonderful day! 😊",
        "thank you": "You're welcome! 😊",
        "your name": "I'm your friendly AI voice assistant!",
    }

    for key, response in responses.items():
        if key in text:
            append_message(response, "bot")
            speak(response)
            return

    if "open youtube and search for" in text:
        query = text.replace("open youtube and search for", "").strip()
        webbrowser.open(f"https://www.youtube.com/results?search_query={query}")
        reply = f"🔍 Searching YouTube for '{query}'..."

    elif "open google and search for" in text:
        query = text.replace("open google and search for", "").strip()
        webbrowser.open(f"https://www.google.com/search?q={query}")
        reply = f"🔎 Searching Google for '{query}'..."

    elif "wikipedia search" in text:
        query = text.replace("wikipedia search", "").strip()
        try:
            summary = wikipedia.summary(query, sentences=2)
            reply = f"📖 Wikipedia Summary: {summary}"
        except wikipedia.exceptions.DisambiguationError as e:
            reply = f"❌ Multiple results found. Be more specific: {',
'.join(e.options[:5])}."
        except wikipedia.exceptions.PageError:
            reply = f"❌ No results found for '{query}'."
```

```
        except Exception as e:
            reply = f"⚠️ An error occurred while searching Wikipedia: {str(e)}"

    elif "open calculator" in text:
        open_application("calc", "Calculator")

    elif text.startswith("open file"):
        filename = text.replace("open file", "").strip()
        open_file_by_name(filename)

    elif "open a file" in text:
        append_message("📂 Please select a file to open...", "bot")
        open_file()

    elif "open notepad" in text:
        open_application("C:\\Windows\\System32\\notepad.exe", "Notepad")

    elif "open chrome" in text:
        open_application("C:\\Program
Files\\Google\\Chrome\\Application\\chrome.exe", "Google Chrome")

    elif "open firefox" in text:
        open_application("C:\\Program Files\\Mozilla Firefox\\firefox.exe",
"Mozilla Firefox")

    elif "open word" in text or "open microsoft word" in text:
        open_application("C:\\Program Files\\Microsoft
Office\\root\\Office16\\WINWORD.EXE", "Microsoft Word")

    elif "open excel" in text or "open microsoft excel" in text:
        open_application("C:\\Program Files\\Microsoft
Office\\root\\Office16\\EXCEL.EXE", "Microsoft Excel")

    else:
        reply = f"❌ I didn't understand that command: '{text}'. Try again."

    append_message(reply, "bot")
    speak(reply)

# Handle send button press
def send_text():
    user_text = user_input.get().strip()
    if user_text:
        append_message(f"You: {user_text}", "user")
        process_command(user_text)
```

```
        user_input.delete(0, tk.END)

# Update chat display
def append_message(text, sender, clear_previous=False):
    chat_window.config(state=tk.NORMAL)
    if clear_previous:
        chat_window.delete("end-2l", tk.END)
    chat_window.insert(tk.END, text + "\n\n", sender)
    chat_window.config(state=tk.DISABLED)
    chat_window.yview(tk.END)

# Open file using file dialog
def open_file():
    file_path = filedialog.askopenfilename(title="Select a file")
    if file_path:
        try:
            os.startfile(file_path)
            append_message(f"📂 Opening file: {os.path.basename(file_path)}",
"bot")
        except Exception as e:
            append_message(f"⚠️ Unable to open the file: {str(e)}", "bot")

# Open file by name from predefined directory
def open_file_by_name(filename):
    file_path = os.path.join(SEARCH_DIRECTORY, filename)
    if os.path.exists(file_path):
        try:
            os.startfile(file_path)
            append_message(f"📂 Opening file: {filename}", "bot")
        except Exception as e:
            append_message(f"⚠️ Unable to open the file: {str(e)}", "bot")
    else:
        append_message(f"❌ File '{filename}' not found in {SEARCH_DIRECTORY}.",
"bot")

# Launch installed application
def open_application(app_name, app_display_name):
    try:
        subprocess.Popen(app_name)
        append_message(f"🚀 Opening {app_display_name}...", "bot")
    except FileNotFoundError:
        append_message(f"⚠️ {app_display_name} not found on your system.",
"bot")
```

```
# Toggle light/dark mode
def toggle_theme():
    global dark_mode
    dark_mode = not dark_mode
    bg_color = "#1e1e1e" if dark_mode else "#f5f5f5"
    fg_color = "white" if dark_mode else "black"
    input_bg = "#333" if dark_mode else "white"

    root.configure(bg=bg_color)
    chat_window.configure(bg=bg_color, fg=fg_color)
    user_input.configure(bg=input_bg, fg=fg_color, insertbackground=fg_color,
highlightthickness=0 if not dark_mode else 2, relief="flat")
    input_frame.configure(bg=bg_color)
    button_frame.configure(bg=bg_color)
    send_button.configure(bg="#28A745" if dark_mode else "#2196F3", fg="white")
    mic_button.configure(bg="#E91E63" if dark_mode else "#D81B60", fg="white")
    theme_button.configure(bg="#FF9800" if dark_mode else "#FFC107", fg="black")

# GUI Setup
root = tk.Tk()
root.title("🧠 AI Voice Chatbot")
root.geometry("500x600")
root.configure(bg="#1e1e1e" if dark_mode else "#ffffff")

chat_window = scrolledtext.ScrolledText(root, state=tk.DISABLED, wrap=tk.WORD,
height=20, font=("Arial", 12), bg="#252526", fg="white", bd=5, relief="flat")
chat_window.pack(pady=10, padx=10, fill=tk.BOTH)

# Welcome message
welcome_message = (
    "👋 Welcome to the AI Voice Chatbot!\n\n"
    "- 🎤 Listen to voice commands.\n"
    "- 🔍 Search Google or YouTube.\n"
    "- 📖 Fetch summaries from Wikipedia.\n"
    "- 🎨 Switch between light/dark themes.\n"
    "- 📂 Open files or launch applications.\n\n"
)
append_message(welcome_message, sender="bot")

# Input frame
input_frame = tk.Frame(root, bg="#1e1e1e")
input_frame.pack(pady=5)

user_input = tk.Entry(input_frame, font=("Arial", 14), bd=5, relief="solid",
bg="#333333", fg="white")
```

```
user_input.grid(row=0, column=0, padx=(0, 5), sticky="ew")
user_input.bind("<Return>", lambda event: send_text())

send_button = tk.Button(input_frame, text="📩 Send", command=send_text,
font=("Arial", 14), width=8)
send_button.grid(row=0, column=1)

mic_button = tk.Button(input_frame, text="🎤 Speak", command=recognize_speech,
font=("Arial", 14), width=8)
mic_button.grid(row=0, column=2)

button_frame = tk.Frame(root)
button_frame.pack(pady=5)

theme_button = tk.Button(button_frame, text="🎨 Theme Toggle",
command=toggle_theme)
theme_button.pack()

toggle_theme()  # Set initial theme

root.mainloop()
```

B. Additional Notes:

- **Directory Structure:** Make sure the SEARCH_DIRECTORY points to an existing directory on your machine.
- **System Compatibility:** Tested on Windows OS; some paths (e.g., for Chrome, Word, etc.) may need adjustment on different systems.
- **Microphone Access:** Ensure that microphone permissions are enabled for Python.
- **Internet Required:** Wikipedia search and speech recognition require a stable internet connection.

References:

1. Python SpeechRecognition Documentation - https://pypi.org/project/SpeechRecognition/
2. pyttsx3 Documentation - https://pypi.org/project/pyttsx3/
3. Tkinter GUI Programming - https://docs.python.org/3/library/tkinter.html
4. Wikipedia Python API - https://pypi.org/project/wikipedia/
5. Webbrowser module - https://docs.python.org/3/library/webbrowser.html
6. Subprocess module - https://docs.python.org/3/library/subprocess.html
7. os module - https://docs.python.org/3/library/os.html
8. GUI Design Patterns - https://refactoring.guru/design-patterns/gui
9. Introduction to NLP - https://www.geeksforgeeks.org/natural-language-processing-overview/
10. AI Voice Assistants Overview - https://www.ibm.com/topics/ai-voice-assistants