PROJECT ON

PERSONAL AI VOICE ASSISTANT

SUBMITTED BY

CHARAKA KAVYA

(AICTE STUDENT ID: STU663f52fe3e8981715426046)

ABSTRACT

The Personal AI Voice Assistant project focuses on building an intelligent system that allows users to interact with their computers through voice commands. It integrates key technologies such as speech recognition, natural language processing, and text-to-speech conversion. The assistant can perform various tasks like opening applications, browsing websites, telling the time, and responding to general queries. A graphical user interface (GUI) enhances accessibility by allowing both text and voice interactions. This system aims to offer a simplified, hands-free, and user-friendly way to enhance digital productivity.

This assistant is capable of performing essential tasks like opening applications, conducting web searches, playing audio tones, telling the current date and time, and providing responses for general queries. It utilizes Google's SpeechRecognition API for converting spoken language into text, while the pyttsx3 library is used to vocalize responses.

One of the key highlights of the system is its simple yet functional GUI built with Tkinter, which supports both text and voice inputs. This dual-interaction model ensures greater flexibility and inclusiveness. Furthermore, the project emphasizes modularity and customizability, allowing users to extend the assistant with new functionalities or connect it with external APIs.

During development, challenges such as handling real-time voice input, API latency, and maintaining GUI responsiveness were tackled using techniques like multithreading and exception handling. The assistant performs effectively for a wide range of use cases, proving its potential as a productivity-enhancing tool.

Overall, this project showcases how combining modern speech technologies with interactive interfaces can result in a powerful virtual assistant. It opens opportunities for future improvements like offline voice recognition, wake-word activation, smarter device integration, and expanded automation features, ultimately pushing the boundary of user–machine interaction.

INDEX

CHAPTER 1

INTRODUCTION

1.INTRODUCTION

1.1 Project Definition

The Personal AI Voice Assistant is a software application designed to facilitate hands-free interaction between the user and their computer. By using natural language voice commands, users can instruct the assistant to perform a variety of tasks such as opening applications, conducting web searches, playing audio, telling the current time and date, and responding to general queries. The system combines speech recognition, natural language processing, and text-to-speech technologies to provide an intuitive and interactive user experience. This assistant is aimed at improving user productivity and accessibility by reducing the reliance on manual input devices like keyboards and mice.

1.2 Objective of the Project

The main objectives of this project include:

- Developing a voice-controlled assistant that can accurately recognize and interpret user commands.
- Automating routine tasks such as launching applications, browsing the web, playing sounds, and providing timely information.
- Creating a graphical user interface (GUI) that supports both voice and text inputs, enhancing accessibility and usability for different types of users.
- Ensuring the system is modular and customizable, allowing for easy addition of new commands and features.
- Providing an offline-capable solution as much as possible to reduce dependency on internet connectivity and improve user privacy.
-

1.3 Software Requirements

To build and run the Personal AI Voice Assistant, the following software components are required:

- Python 3.x: The core programming language used for developing the assistant.
- Libraries and Modules:
    - pyttsx3: A text-to-speech conversion library that allows the assistant to speak responses aloud.
    - SpeechRecognition: Enables converting spoken language into text using speech recognition APIs.
    - tkinter: The built-in Python GUI library used for building a simple and interactive graphical interface.
    - webbrowser: To open URLs in the system's default web browser.
    - datetime: For retrieving and formatting current date and time.

    - os: For interacting with the operating system, such as opening applications.
    - threading: To handle simultaneous voice input and GUI responsiveness.
    - winsound (Windows-only): For playing simple audio tones.
- Operating System Compatibility: The assistant is compatible with Windows, Linux, and MacOS environments.

1.4 Hardware Requirements

The hardware requirements for deploying the Personal AI Voice Assistant include:

- Microphone: Essential for capturing the user's voice commands clearly and accurately.
- Speaker or Headphones: To output the assistant's spoken responses and audio feedback.
- Computing Device: A computer or laptop with at least 4GB of RAM to ensure smooth processing of voice commands and GUI interactions.
- Optional: A stable internet connection is recommended for APIs that may require online access, though offline features are prioritized.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 Existing System

Several popular voice assistants are currently available in the market, such as Apple's Siri, Google Assistant, Amazon Alexa, and Microsoft Cortana. These assistants are integrated into smartphones, smart speakers, and computers, providing a range of functionalities including answering questions, setting reminders, playing music, and controlling smart home devices. They primarily rely on cloud-based services for processing voice commands and generating responses. Many open-source projects also exist, aiming to provide voice assistant capabilities with varying levels of complexity.

2.2 Disadvantages

Despite their popularity, existing voice assistant systems have several limitations:

- Dependency on Internet Connectivity: Most assistants require a constant internet connection to process voice commands via cloud services, which limits offline usability.
- Privacy Concerns: Cloud-based processing raises privacy issues as voice data is transmitted to external servers.
- Limited Customization: Proprietary systems provide limited scope for users or developers to customize commands and functionality.
- High Hardware Requirements: Some assistants require modern hardware or specific devices to function optimally.
- Complexity: Many systems are complex to set up or modify for personal or educational use.
-

2.3 Proposed System

The proposed Personal AI Voice Assistant addresses these limitations by providing an offline-capable, customizable voice assistant built entirely in Python. It combines speech recognition, command processing, and text-to-speech conversion locally on the user's machine. The system offers a user-friendly graphical interface that supports both voice and text inputs, enabling greater accessibility and flexibility. Unlike commercial assistants, it emphasizes modularity, allowing users to add or modify commands easily. The assistant focuses on essential functionalities such as opening applications, browsing websites, and providing basic conversational responses without requiring continuous internet connectivity.

2.4 Advantages

- Offline Functionality: Can perform many tasks without an internet connection, enhancing privacy and usability.
- Open Source and Lightweight: Developed using Python with minimal dependencies, making it easy to run on various systems.
- Customizable: Users can extend or modify the assistant's capabilities by adding new commands or integrating external APIs.
- Simple GUI: Provides a visual interface that supports both typing and voice commands, improving user interaction.
- Privacy-Focused: Voice data is processed locally, minimizing data exposure.

CHAPTER 3

METHODOLOGY

3.METHODOLOGY

The methodology outlines the step-by-step approach taken to develop the Personal AI Voice Assistant, focusing on capturing user voice input, processing commands, generating responses, and providing an interactive interface.

Step 1: Voice Input Capture

The system begins by capturing the user's spoken commands using the device's microphone. The Python SpeechRecognition library is employed to continuously listen and convert spoken words into text. This step is crucial for enabling natural voice interaction.

Step 2: Speech-to-Text Conversion

Once voice input is captured, the speech recognition engine processes the audio and converts it into textual commands. This allows the system to interpret the user’s intent.

Step 3: Command Processing

The textual input is analyzed and matched against a predefined set of commands. These commands include tasks like opening applications, searching the web, playing audio, or providing date/time information. If the input matches a known command, the respective task handler is executed.

Step 4: Fallback Response Generation

If the command is unrecognized or requires a conversational response, a fallback module generates an appropriate reply. This ensures the assistant remains responsive even for general or unexpected queries.

Step 5: Text-to-Speech Conversion

The system converts the generated textual response back to audible speech using the pyttsx3 library. This completes the voice interaction loop, providing audible feedback to the user.

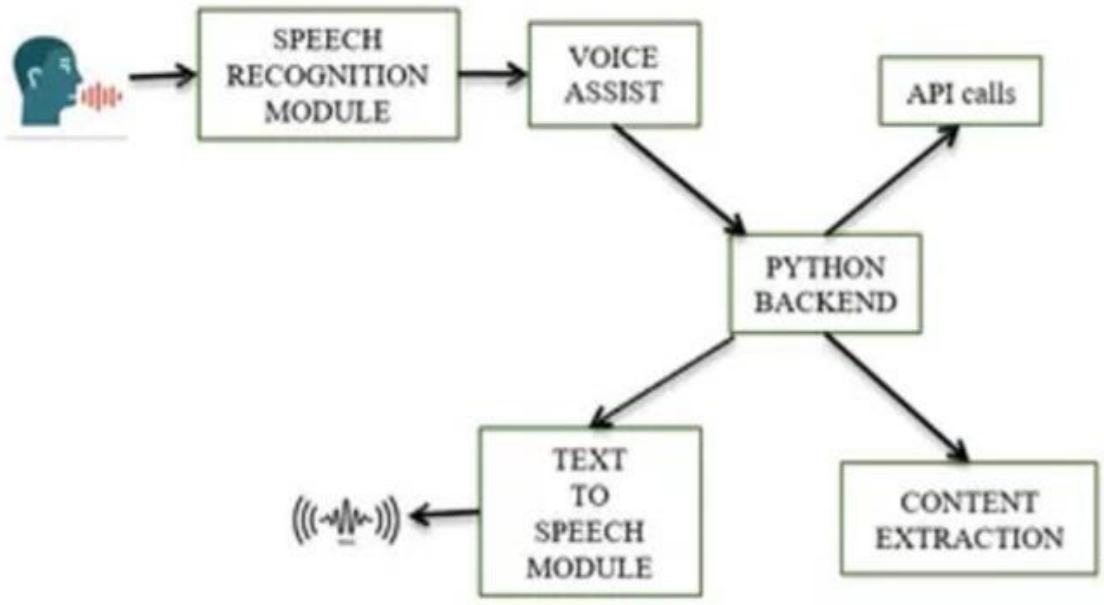
Step 6: Graphical User Interface (GUI)

A user-friendly GUI built with Tkinter displays the conversation in text form, allows manual text input, and provides buttons for example commands. This makes the assistant accessible for users who prefer typing or want visual confirmation of interactions.

Step 7: Multithreading and Exception Handling

To maintain smooth performance, the system uses multithreading to process voice input and GUI events simultaneously, preventing freezing or lag. Exception handling is implemented to gracefully manage errors like microphone issues or unrecognized inputs.

3.1 Block Diagram



CHAPTER 4

IMPLEMENTATION

4. IMPLEMENTATION

4.1 Components of the Personal AI Voice Assistant

The Personal AI Voice Assistant is composed of several key components that work together to enable seamless voice interaction and task execution. Each component plays a specific role in ensuring the system functions efficiently and responsively.

1. Voice Recognition Module

- Captures audio input from the user's microphone.
- Converts spoken words into text using speech-to-text technologies (via Python's SpeechRecognition library).
- Handles background noise and listens continuously or on-demand.

2. Command Processor

- Analyzes the converted text command.
- Matches the input to a predefined set of supported commands like opening websites, applications, or performing simple tasks (e.g., telling time).
- Routes recognized commands to the respective function for execution.

3. Fallback Response Generator

- Handles inputs that do not match predefined commands.
- Generates conversational or informative replies to maintain interaction flow.
- Ensures the assistant can answer general queries and respond meaningfully.

4. Text-to-Speech (TTS) Engine

- Converts textual responses into spoken output.
- Uses libraries like pyttsx3 to provide clear and natural-sounding voice feedback.
- Supports asynchronous audio playback to keep the interface responsive.

5. Graphical User Interface (GUI)

- Built using the Tkinter framework.
- Displays conversation history, accepts typed commands, and shows real-time responses.
- Provides buttons for example commands to assist users in interacting.
- Offers visual feedback alongside voice responses for enhanced accessibility.

6. Multithreading and Error Handling

- Ensures simultaneous processing of voice recognition, GUI events, and response generation without freezing.
- Manages errors such as microphone access issues, API failures, or unrecognized input gracefully to maintain stability.

4.1 Source Code

```
import pyttsx3
import datetime
import webbrowser
import os
import speech_recognition as sr
import tkinter as tk
from tkinter import scrolledtext
import openai
import winsound
import time
import threading

# Set your OpenAI API key here
openai.api_key = "your-real-api-key"  # Replace with your valid OpenAI API key

# Initialize text-to-speech engine
engine = pyttsx3.init()

# Store last spoken text to allow repeat
last_spoken_text = ""

# Speak text aloud
def speak(text):
    global last_spoken_text
    last_spoken_text = text
    engine.say(text)
    engine.runAndWait()

# Greeting based on time of day
def greet():
    hour = datetime.datetime.now().hour
    if hour < 12:
        text = "Good morning!"
    elif hour < 18:
        text = "Good afternoon!"
    else:
        text = "Good evening!"
    update_chat(f"Assistant: {text}")
    speak(text)
    intro = "I am your assistant. How can I help you?"
    update_chat(f"Assistant: {intro}")
```

```
    speak(intro)

# Listen using microphone
def listen():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        update_chat("Assistant: Listening...")
        r.adjust_for_ambient_noise(source)  # Helps in noisy environments
        try:
            audio = r.listen(source, timeout=5, phrase_time_limit=7)
            command = r.recognize_google(audio)
            update_chat(f"You said: {command}")
            return command.lower()
        except sr.WaitTimeoutError:
            speak("Listening timed out, please try again.")
        except sr.UnknownValueError:
            speak("Sorry, I didn't catch that.")
        except Exception as e:
            speak("Something went wrong.")
            print("Speech recognition error:", e)
        return ""

# Use OpenAI to respond to unrecognized commands
def fallback_response(command):
    try:
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": command}]
        )
        return response.choices[0].message.content
    except Exception as e:
        print("OpenAI API error:", e)
        return "I couldn't connect to OpenAI services. Please check your API key
or internet."

# Simple music using beeps
def play_simple_music():
    notes = [
        (262, 400), (262, 400), (392, 400), (392, 400),
        (440, 400), (440, 400), (392, 800),
        (349, 400), (349, 400), (330, 400), (330, 400),
        (294, 400), (294, 400), (262, 800),
    ]
    for freq, dur in notes:
        winsound.Beep(freq, dur)
```

```
        time.sleep(0.1)

# Respond to commands
def respond(command):
    if "time" in command:
        now = datetime.datetime.now()
        reply = f"Today is {now.strftime('%A, %B %d, %Y')}. The time is
{now.strftime('%I:%M %p')} IST."
    elif "youtube" in command:
        reply = "Opening YouTube"
        webbrowser.open("https://www.youtube.com")
    elif "google" in command:
        speak("What should I search on Google?")
        query = listen()
        if query:
            webbrowser.open(f"https://www.google.com/search?q={query}")
            reply = f"Searching Google for {query}"
        else:
            reply = "No search query received."
    elif "joke" in command:
        reply = "Why don't scientists trust atoms? Because they make up
everything!"
    elif "open notepad" in command:
        reply = "Opening Notepad"
        os.system("notepad")
    elif "play music" in command:
        reply = "Playing a simple tune"
        play_simple_music()
    elif "who are you" in command:
        reply = "I am your AI assistant, built with Python and OpenAI."
    elif "image" in command or "generate image" in command:
        reply = "Opening image generator for you."
        webbrowser.open("https://chat.openai.com/")
    elif "exit" in command or "quit" in command:
        reply = "Goodbye!"
        speak(reply)
        root.destroy()
        exit()
    else:
        reply = fallback_response(command)

    update_chat(f"Assistant: {reply}")
    speak(reply)

# Thread wrapper for responsiveness
```

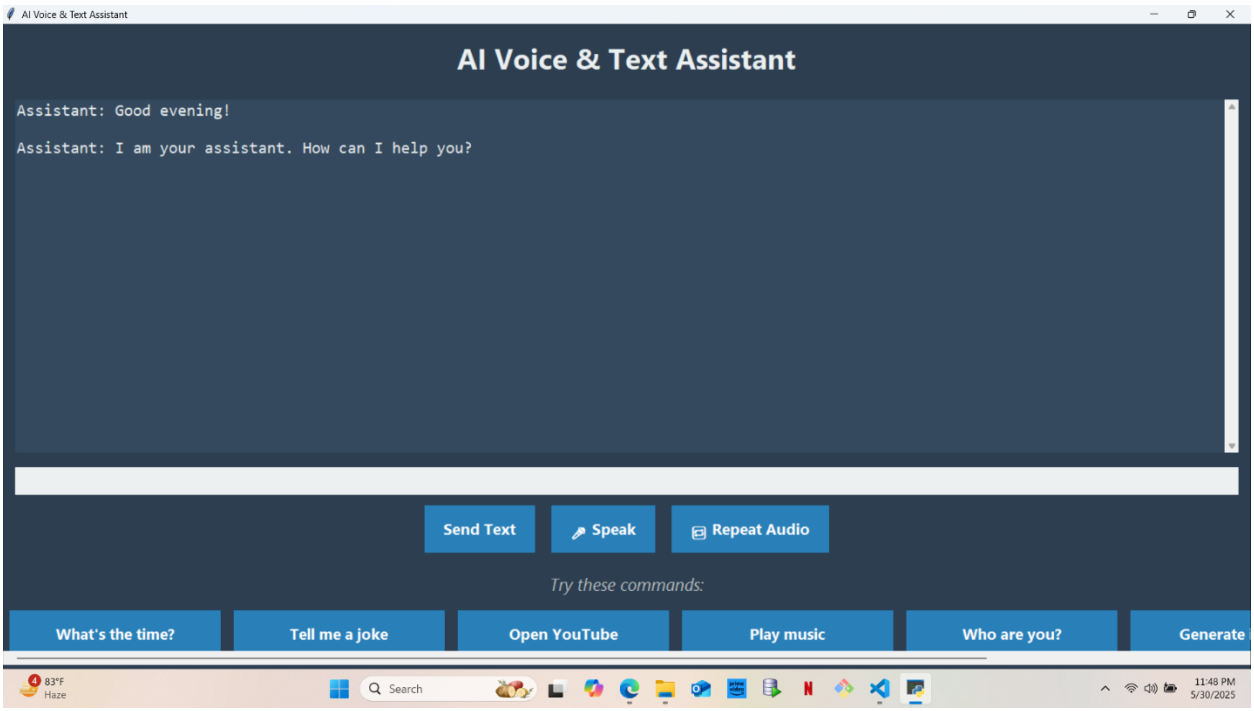
```
def threaded_respond(command):
    threading.Thread(target=respond, args=(command,), daemon=True).start()

# Handle voice input
def handle_voice():
    command = listen()
    if command.strip():
        threaded_respond(command)
    else:
        speak("Please say something.")

# Handle text input
def handle_text():
    command = entry.get()
    if command.strip():
        update_chat(f"You typed: {command}")
        threaded_respond(command)
    else:
        speak("Please type something.")
    entry.delete(0, tk.END)

# Update chat window
def update_chat(message):
    chat_box.configure(state='normal')
    chat_box.insert(tk.END, message + "\n\n")
    chat_box.configure(state='disabled')
    chat_box.see(tk.END)

# Repeat last response
def repeat_audio():
    if last_spoken_text:
        speak(last_spoken_text)
    else:
        speak("There is nothing to repeat yet.")

# --- GUI SETUP ---
root = tk.Tk()
root.title("Personal AI Voice Assistant")
root.geometry("750x550")
root.configure(bg="#2c3e50")

label = tk.Label(root, text="Personal AI Voice Assistant", font=("Segoe UI", 26,
"bold"),
                 fg="#ecf0f1", bg="#2c3e50")
label.pack(pady=15)
```

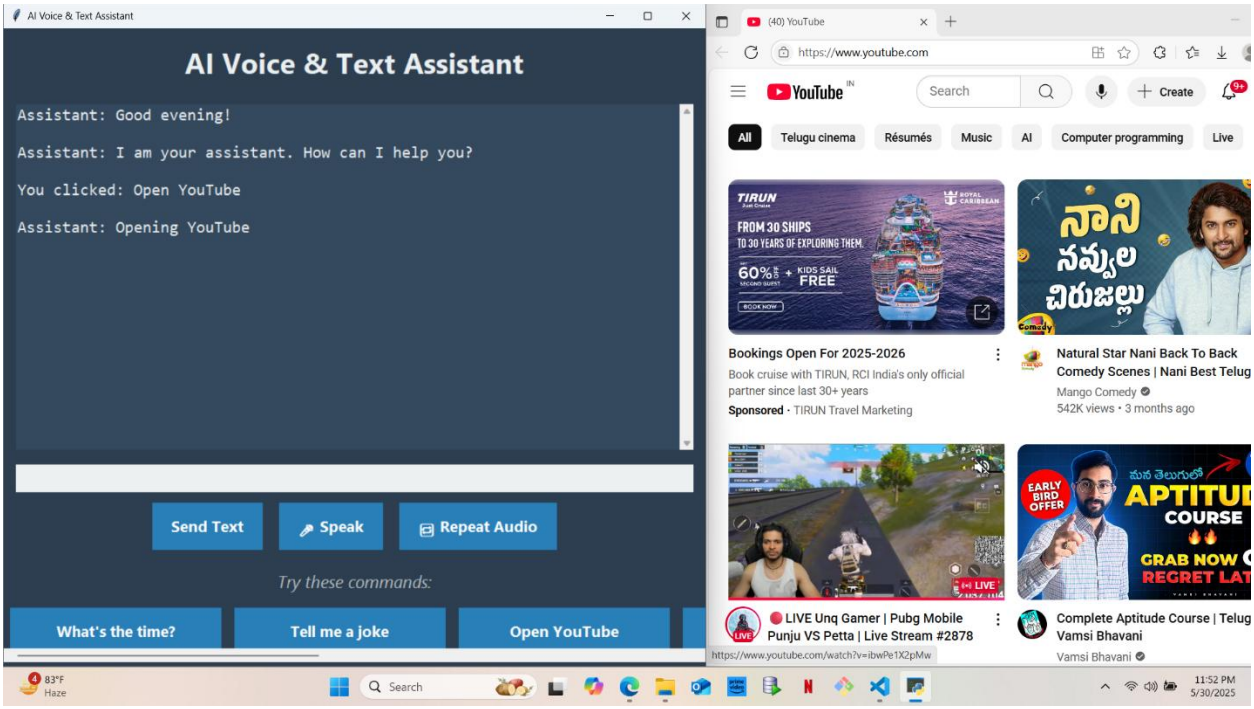
```
chat_box = scrolledtext.ScrolledText(root, wrap=tk.WORD, font=("Consolas", 15),
                                     state='disabled', height=16, bg="#34495e",
fg="#ecf0f1")
chat_box.pack(padx=15, pady=10, fill=tk.BOTH, expand=True)

entry = tk.Entry(root, font=("Segoe UI", 16), bg="#ecf0f1", fg="#2c3e50")
entry.pack(padx=15, pady=8, fill=tk.X)

btn_frame = tk.Frame(root, bg="#2c3e50")
btn_frame.pack(pady=5)

def style_button(btn):
    btn.config(font=("Segoe UI", 14, "bold"), bg="#2980b9", fg="white",
               activebackground="#3498db", relief=tk.FLAT, padx=18, pady=10,
cursor="hand2")
    btn.bind("<Enter>", lambda e: btn.config(bg="#3498db"))
    btn.bind("<Leave>", lambda e: btn.config(bg="#2980b9"))

btn_text = tk.Button(btn_frame, text="Send Text", command=handle_text)
btn_voice = tk.Button(btn_frame, text="🎤 Speak", command=handle_voice)
btn_repeat = tk.Button(btn_frame, text="🔁 Repeat Audio", command=repeat_audio)

for i, b in enumerate([btn_text, btn_voice, btn_repeat]):
    style_button(b)
    b.grid(row=0, column=i, padx=10)

# Suggested commands
suggestion_label = tk.Label(root, text="Try these commands:", font=("Segoe UI",
16, "italic"),
                            fg="#bdc3c7", bg="#2c3e50")
suggestion_label.pack(pady=(15, 5))

suggestion_frame = tk.Frame(root, bg="#2c3e50")
suggestion_frame.pack(pady=5, fill=tk.X)

canvas = tk.Canvas(suggestion_frame, height=55, bg="#2c3e50",
highlightthickness=0)
h_scroll = tk.Scrollbar(suggestion_frame, orient=tk.HORIZONTAL,
command=canvas.xview)
canvas.configure(xscrollcommand=h_scroll.set)
h_scroll.pack(side=tk.BOTTOM, fill=tk.X)
canvas.pack(side=tk.LEFT, fill=tk.X, expand=True)

btn_container = tk.Frame(canvas, bg="#2c3e50")
```

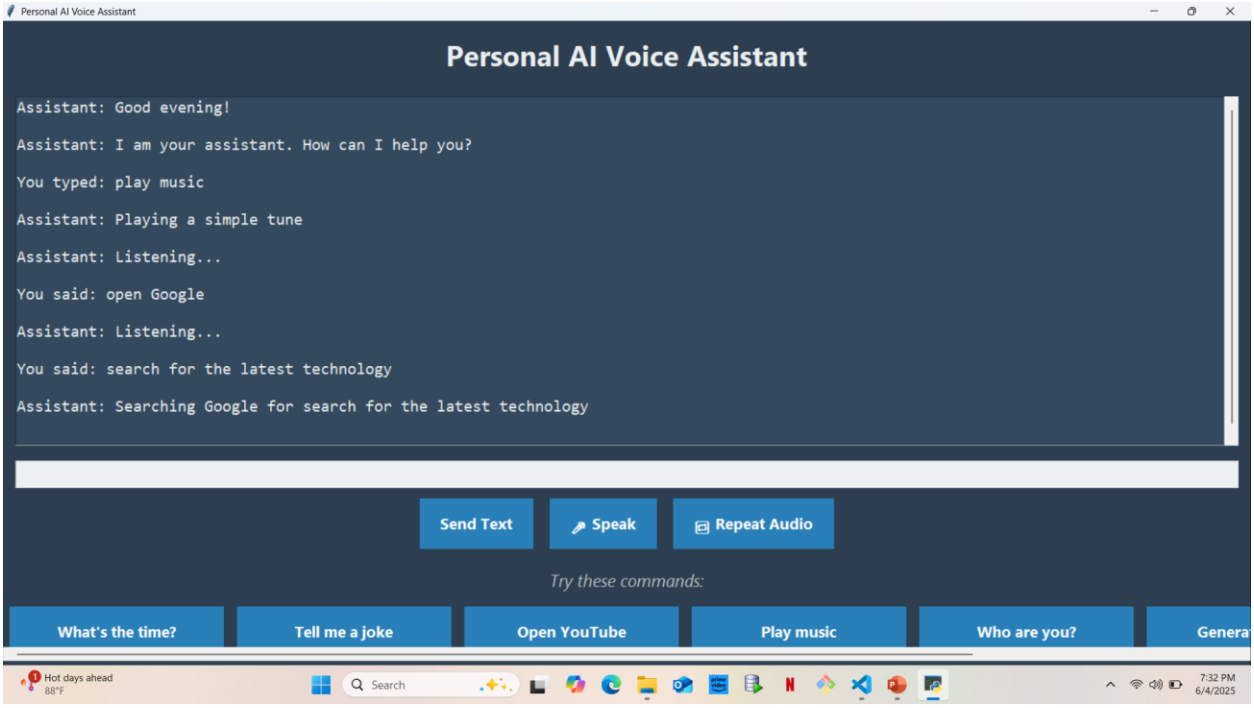
```
canvas.create_window((0, 0), window=btn_container, anchor='nw')

def on_configure(event):
    canvas.configure(scrollregion=canvas.bbox('all'))
btn_container.bind('<Configure>', on_configure)

example_commands = [
    "What's the time?",
    "Tell me a joke",
    "Open YouTube",
    "Play music",
    "Who are you?",
    "Generate image",
    "Open notepad",
]

for cmd in example_commands:
    b = tk.Button(btn_container, text=cmd, width=20,
                  command=lambda c=cmd: threaded_respond(c))
    style_button(b)
    b.pack(side=tk.LEFT, padx=8, pady=5)

# Start assistant
greet()
root.mainloop()
```

CHAPTER 5

RESULTS






Personal AI Voice Assistant
Personal AI Voice Assistant
Assistant: Good evening!
Assistant: I am your assistant. How can I help you?
You typed: play music
Assistant: Playing a simple tune
Assistant: Listening...
You said: open Google
Assistant: Listening...
You said: search for the latest technology
Assistant: Searching Google for search for the latest technology
Send Text
Speak
Repeat Audio
Try these commands:
What's the time?
Tell me a joke
Open YouTube
Play music
Who are you?
Genera
Hot days ahead
88°F
Search
7:32 PM
6/4/2025

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6. CONCLUSION & FUTURE SCOPE

6.1 Conclusion

The Personal AI Voice Assistant efficiently integrates voice recognition, text processing, and speech synthesis to provide a smooth and interactive user experience. It allows users to perform essential tasks such as opening applications, browsing the web, and receiving intelligent responses using both speech and GUI-based input. The project demonstrates how accessible and practical a lightweight virtual assistant can be using open-source technologies. Despite facing challenges like handling real-time audio and ensuring accurate responses, the assistant proved to be effective and responsive. The system paves the way for smarter personal productivity tools and highlights the growing importance of natural user interfaces.

6.2 Future Scope

- Integration of wake-word detection for hands-free activation.
- Enable offline voice recognition for privacy and speed.
- Expand functionality to include smart home device control.
- Support calendar reminders, alarms, and file system automation.
- Enhance GUI with real-time feedback and animations.

6.3 References

- https://pypi.org/project/pyttsx3/
- https://pypi.org/project/SpeechRecognition/
- https://docs.python.org/3/
- https://tkdocs.com/