

# **PYTHON PROJECT DOCUMENTATION**

## **CHATBOT ASSISTANT AND VOICE ASSISTANT**

### **PROBLEM STATEMENT:**

The goal of this project is to develop an integrated voice and chat bot assistant that enables users to interact through both spoken and written commands. The assistant should provide a seamless conversational experience for tasks like fetching weather updates, news, jokes, translations, diary management, and more. It must support dynamic command integration from external programs, allowing for versatile applications. A user-friendly GUI with an animated avatar will enhance engagement, with smooth transitions between voice and text modes. The assistant should ensure natural and prompt responses while addressing common issues, such as model generation warnings or animation disruptions, without compromising performance. The project aims to create an intuitive and efficient digital assistant that makes technology more accessible and user-friendly.

### **FUNCTIONALITIES:**

1. **Diary Mode:** Allows users to create, read, and manage personal diary entries.

- Users can write, edit, and review diary entries, making it a convenient personal note-keeping tool for capturing thoughts and experiences.

BY- PYTHON FILES

2. **Reader Mode:** Reads out text content aloud for users.

- This mode provides audio output for text content, such as articles or diary entries, making information accessible to those who prefer listening.

BY- PYTHON FILES

3. **Joke Delivery:** Shares jokes for entertainment.

- Offers random or user-requested jokes to add a touch of humor, lightening the mood during interactions.

BY- USING PYTHON LIB ( `get_jokes` )

4. **Quotes:** Delivers motivational or inspirational quotes.

- Provides users with motivational or thought-provoking quotes, offering daily inspiration or encouragement.

BY- USING PYTHON API(<https://zenquotes.io/api/random>)

5. **Weather Forecast:** Provides current weather conditions and forecasts.

- Retrieves weather updates for specified locations, including temperature and conditions, keeping users informed about the weather.

BY-

API([http://api.weatherapi.com/v1/current.json?key={api\\_key}&q={city}](http://api.weatherapi.com/v1/current.json?key={api_key}&q={city}))

6. **News Briefing:** Offers the latest news updates or headlines.

- Summarizes recent news headlines or specific topics, helping users stay up-to-date with current events.

**BY-**

**API**([https://newsapi.org/v2/everything?q={category}&apiKey={api\\_key}](https://newsapi.org/v2/everything?q={category}&apiKey={api_key}))

**7. Flashcards:** Assists with learning by presenting educational flashcards.

- Displays flashcards to aid in learning and memorizing information, supporting interactive education on various topics.

## **BY-USING PYTHON COLLECTIONS**

**8. Recipe Finder:** Finds and shares recipes based on user input.

- Searches for recipes according to user preferences or available ingredients, making meal planning easier.

**BY-**

**API**([https://api.spoonacular.com/recipes/findByIngredients?ingredients={ingredients}&apiKey={api\\_key}](https://api.spoonacular.com/recipes/findByIngredients?ingredients={ingredients}&apiKey={api_key}))

**9. Calendar:** Helps manage events,

- Allows, view events aiding in schedule and time management.

## **USING PYTHON-LIB (datetime)**

**10. Calculator:** Performs basic arithmetic and complex calculations.

- Facilitates quick mathematical operations, from simple arithmetic to advanced calculations.

**11. Expense Tracker:** Allows users to manage and track expenses.

- Users can log and categorize expenses, helping monitor spending and manage personal finances.

## **USING-PYTHON COLLECTIONS**

**12. Natural Responses:** Provides conversational and human-like replies.

- Ensures interactions feel natural and engaging, making the assistant's replies fluid and contextually appropriate.

## **USING CHATTERBOT IN CHAT ASSISTANT (DIALOGPT-MEDIUM)**

## **USING TORCH IN VOICE ASSISTANT(DIALOGPT-MEDIUM)**

**13. Voice and Text Interaction:** Supports both voice and text-based inputs for a seamless experience.

- The assistant can process commands given via voice or text, providing flexible interaction options for users.

## **USING TKINTER, SCATTERFLOW, TK**

**14. Switch Between Two Modes:** Allows seamless switching between voice and text modes.

- Users can easily transition between voice and text-based interactions without disrupting the assistant's functionality.

## **USING THREADS, DESTRUCTION OF FILE**

**15. Dynamic Changing:** Adapts the conversation flow based on user inputs and context.

- The assistant dynamically adjusts responses and behavior according to user preferences and ongoing conversations, offering a personalized experience.

These functionalities combine to create a versatile and user-friendly digital assistant capable of handling diverse tasks and

adapting to various user needs through both voice and text interactions.

## **SOURCE CODE:**

### **FOR VOICE ASSISTANT:**

```
import pyttsx3 as p
import speech_recognition as sr
from googletrans import Translator
import pyjokes
import pywhatkit
import wikipedia
import webbrowser
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
import requests
import random
import webbrowser
import datetime
import csv
import re
from datetime import datetime
import warnings
import tkinter as tk
from tkinter import scrolledtext, ttk
import threading
import os
import subprocess

while text.lower() != "exit":
    text = listen()
    text=text.lower()

    if "say" in text and "hi" in text:
        speak("Hi sir ")
    # Repeat mode
    elif "activate" in text and "repeat" in text and "mode" in text:
        speak("Repeat mode activated. What do you want to say?")
```

```

while True:
    text = listen()
    if text.lower() == "exit":
        speak("Exiting repeat mode.")
        break
    if text:
        speak(text)

# Diary mode
elif "activate" in text and "diary" in text and "mode" in text:
    speak("Activating diary mode.")
    speak("Enter the pass to confirm...")
    pass1 = listen()

    if pass1.strip() == "123": # Strips whitespace from input
        # Get today's date
        today_date = datetime.today().strftime('%Y-%m-%d')
        path = "D:\\2nd year\\python_assistant\\new1.txt"

        # Ensure the directory exists
        directory = os.path.dirname(path)
        if not os.path.exists(directory):
            os.makedirs(directory)

        # Write today's date to the file
        try:
            with open(path, 'w') as file:
                file.write(f"{today_date} ") # Add a newline for clarity
                print(f"Date written to file: {today_date}") # Debugging output

            speak("What would you like to record for today?")
            content = listen()

            if content: # Only write if content is not empty
                with open(path, 'a') as file:
                    file.write(f"{content}\n")
                    print(f"Content written to file: {content}") # Debugging output
                speak("Your entry has been saved.")
            else:

```

```

        speak("No content recorded for today.")
    except Exception as e:
        print(f"Error writing to file: {e}")
        speak("An error occurred while saving your entry.")
    else:
        speak("Wrong password... Exiting from diary mode.")

# Reader mode
elif "activate" in text and "reader" in text and "mode" in text:
    speak("Activating reader mode.")
    speak("Say the date you want to read the record for.")
    path = "D:\\2nd year\\python_assistant\\new1.txt"
    date_input = listen() # Listen for the date input from the user

    try:
        # Convert spoken date to the desired format (YYYY-MM-DD)
        date_obj = datetime.strptime(date_input, '%d %m %Y')
        formatted_date = date_obj.strftime('%Y-%m-%d')

        # Open the file to read
        with open(path, 'r') as file:
            found = False
            for line in file:
                # Check if the current line matches the formatted date
                if formatted_date in line:
                    speak("Date matched.")
                    speak(line)
                    print(line) # Read the line corresponding to the date
                    found = True
                    read_record = True # Start reading subsequent lines
                    continue # M

            if not found:
                speak("No records found for the specified date.")

    except FileNotFoundError:
        speak("The diary file does not exist. Please check the file path.")
    except ValueError:

```

```
        speak("The date format you provided is incorrect. Please use the format  
'DD MM YYYY'.")
```

```
    except Exception as e:
```

```
        print(f"An error occurred: {e}")
```

```
        speak("An error occurred while reading the file.")
```

```
elif "activate" in text and "translator" in text:
```

```
    speak("Activating Translator")
```

```
    speak("which language you need to translate")
```

```
    language=listen()
```

```
    speak("You can translate now")
```

```
    text=listen()
```

```
    translation = translate_text(text.strip(),dest_language=language.strip())
```

```
    print(f"Translated to {language.strip()}: {translation}")
```

```
    speak(translation)
```

```
elif "tell" in text and "joke" in text:
```

```
    joke = pyjokes.get_joke()
```

```
    print("Here's a joke for you:",joke)
```

```
    speak(joke)
```

```
elif "tell" in text and "quotes" in text or "quote" in text:
```

```
    response = requests.get("https://zenquotes.io/api/random")
```

```
    if response.status_code == 200:
```

```
        data = response.json()
```

```
        quote = f"{data[0]['q']} - {data[0]['a']}"
```

```
        print("The Quote is :",quote)
```

```
        speak( quote)
```

```
    else:
```

```
        print("Failed to fetch a quote.")
```

```
elif "tell" in text and "weather" in text and "forecast" in text:
```

```
    speak("Tell the city name to find the forecast: ")
```

```
    city=listen()
```

```
    city=city.lower()
```

```
    api_key = "35df73c218d549d6aa3165644241810"
```

```
    base_url =
```

```
f"http://api.weatherapi.com/v1/current.json?key={api_key}&q={city}"
```

```
    response = requests.get(base_url)
```

```
    if response.status_code == 200:
```

```
        data = response.json()
```

```
        temperature = data['current']['temp_c']
```

```

weather_description = data['current']['condition']['text']
humidity = data['current']['humidity']
weather_info = (
    f"The current temperature in {city} is {temperature} degrees Celsius. "
    f"The weather is {weather_description} with a humidity of {humidity}
percent."
)
print(weather_info)
speak(weather_info)
else:
    error_message = "Failed to fetch weather information. Please check the
city name or try again later."
    speak(error_message)
elif "tell" in text and "news" in text:
    api_key = "165011e733ac47439f2e7696b0f1d379"
    speak("Enter the news category (business, entertainment, general, health,
science, sports, technology,education): ")
    category = listen()
    url = f'https://newsapi.org/v2/everything?q={category}&apiKey={api_key}'
    response = requests.get(url)
    if response.status_code == 200:
        articles = response.json()['articles']
        news_brief = ""
        for article in articles[:3]: # Get top 5 articles
            title = article['title']
            description = article['description']
            news_brief += f"Title: {title}. Description: {description}\n"
        print(news_brief)
        speak(news_brief)
    else:
        error_message = "Failed to fetch news articles."
        print(error_message)
        speak(error_message)
elif "activate" in text and "flash card" in text and "mode" in text:
    calibrate_microphone() # Call the calibration function
    engine.say("Welcome to the flashcard assistant. You can add flashcards or
start the quiz. Say 'add' to add a flashcard, or 'quiz' to start the quiz.")
    engine.runAndWait()

```

```

while True:
    command = process_voice_input("What would you like to do? Say 'add'
to add a flashcard or 'quiz' to start the quiz.")

    if command in ["add", "add flashcards", "adding flashcards", "add flash
card", "add flash cards"]:
        print("Adding flashcards...")
        add_flashcard()
    elif command in ["quiz", "start quiz"]:
        print("Starting quiz...")
        quiz_mode()
    elif command in ["exit", "quit"]:
        engine.say("Goodbye!")
        engine.runAndWait()
        break
    else:
        print("Unrecognized command. Please try again.")
        engine.say("I did not understand that. Please say 'add' or 'quiz'.")
        engine.runAndWait()
elif "tell" in text and "recipe" in text :
    speak("Welcome to the recipe assistant.")
    recipe()
    speak("Goodbye!")
elif "activate" in text and "calendar" in text:
    voice_activated_calendar()
elif "activate" in text and "calculator" in text:
    engine.say("Welcome to the voice-activated calculator. Say 'exit' to quit.")
    engine.runAndWait()

while True:
    # Get voice input
    input_text = process_voice_input_calculator()

    # Check if the user wants to exit
    if "exit" in input_text:
        engine.say("Goodbye!")
        engine.runAndWait()
        break

```

```

# Calculate the result based on voice input
response = evaluate_expression(input_text)

# Output the result
engine.say(response)
engine.runAndWait()
print(response)
elif "activate" in text and "tracker" in text:
    try:
        with open(CSV_FILE, mode='x', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(["Date", "Amount", "Currency", "Category",
"Description"])
    except FileExistsError:
        pass

# Voice assistant for adding expenses
engine.say("Welcome to your voice-activated expense tracker. Say 'exit' to
quit or 'show expenses' to view your expenses.")
engine.runAndWait()

while True:
    # Get voice input
    input_text = process_expense_input()

    # Check if the user wants to exit
    if "exit" in input_text:
        engine.say("Goodbye!")
        engine.runAndWait()
        break

    # Check if the user wants to show expenses
    elif "show expenses" in input_text:
        response = show_expenses()
        engine.say(response)
        engine.runAndWait()

    # Parse and add the expense
    elif "add expenses" in input_text:

```

```

    print("Into Adding Expenses")
    details = parse_expense_details(input_text)
    if details:
        amount, currency, category, description = details
        add_expense(amount, currency, category, description)
        response = f"Added {amount} {currency} for {category}. Description:
{description}."
    else:
        response = "I could not understand your expense. Please try again."

    # Speak the response
    engine.say(response)
    engine.runAndWait()

elif "change" in text and "voice" in text:
    speak("Sure... I will change my voice")
    print("Process undergoing....")
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[1].id)
    speak("Voice successfully changed")
elif "what" in text and "your" in text and "boss name" in text:
    speak("my boss name is kavin")
elif "what" in text and "your" in text and "name" in text:
    speak("My name is Nova....A voice Assistant")
elif "google" in text:
    speak("searching in Google")
    searchGoogle(text)
elif "youtube" in text:
    speak("Searching in youtube")
    searchYoutube(text)
elif "wikipedia" in text:
    speak("Searching in Wikipedia")
    searchWikipedia(text)
else:
    if text=="":
        print("Try again")
    else:
        last_response = chat_with_bot(text) # Start the chat with the provided
initial message

```

```
#print("Last response from chatbot:", last_response)
```

### **SOURCE CODE FOR CHATBOT ASSISTANT:**

```
import tkinter as tk
from tkinter import scrolledtext, messagebox, ttk
import requests
import json
import schedule
import time, datetime
import threading
from chatterbot import ChatBot
from chatterbot.trainers import ChatterBotCorpusTrainer
from chatterbot.logic import BestMatch
import webbrowser
import subprocess

chatbot = ChatBot("Assistant",
    logic_adapters=[
        {
            'import_path': 'chatterbot.logic.BestMatch'
        }
    ]
)

trainer = ChatterBotCorpusTrainer(chatbot)
trainer.train(
    "chatterbot.corpus.english.greetings",
    "chatterbot.corpus.english.conversations",
    "chatterbot.corpus.english.emotion",
    "chatterbot.corpus.english.humor",
    "chatterbot.corpus.english.psychology"
)

# Command functions for each feature
def get_weather(city, output):
    api_key = 'bd5e378503939ddaee76f12ad7a97608'
    url =
    f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}
    &units=metric"
```

```
response = requests.get(url)
weather_data = response.json()

if weather_data["cod"] == 200:
    temp = weather_data["main"]["temp"]
    description = weather_data["weather"][0]["description"]
    result = f"Bot: The temperature in {city} is {temp}°C with {description}."
    output.insert(tk.END, result + "\n")
else:
    output.insert(tk.END, "City not found.\n")
```

```
def get_joke(output):
    # url =
    "https://v2.jokeapi.dev/joke/Any?blacklistFlags=nsfw,religious,political,racist,explicit"
    url = "https://v2.jokeapi.dev/joke/Any"
    response = requests.get(url).json()
    if response["type"] == "single":
        joke = response['joke']
    else:
        joke = f"{response['setup']} - {response['delivery']}"
    output.insert(tk.END, joke + "\n")
```

```
def get_quote(output):
    url = "https://zenquotes.io/api/quotes"
    response = requests.get(url).json()
    response = response[0]
    quote = f"{response['q']} — {response['a']}"
    output.insert(tk.END, quote + "\n")
```

```
def get_news_briefing(output):
    api_key = 'dac4cafc6b514145b0cbd760ef8b0a77'
    url = f"https://newsapi.org/v2/top-headlines?country=us&apiKey={api_key}"
    response = requests.get(url)
    news_data = response.json()

    if news_data["status"] == "ok":
        headlines = [article["title"] for article in news_data["articles"][:5]] # Limit
        to top 5 headlines
```

```

news_brief = "\n".join(headlines)
output.insert(tk.END, "Here are the top news headlines:\n" + news_brief +
"\n")
else:
    output.insert(tk.END, "Could not fetch news at this time.\n")

def get_today_info(output):
    # Get today's date and day
    today = datetime.datetime.now()
    date_str = today.strftime("%B %d, %Y")
    day_name = today.strftime("%A")
    current_time = today.strftime("%l:%M %p")

    # Display basic date and time
    output.insert(tk.END, f"Today is {day_name}, {date_str}. The current time is
{current_time}.\n\n")

    # Fun Fact: Special events and historical facts about today
    # 1. Get a list of holidays or events (Calendarific API)
    calendarific_api_key = 'l1IJKNGaR9E2HBgHnIJ4ctGK1qly4gm'
    holiday_url =
f"https://calendarific.com/api/v2/holidays?&api_key={calendarific_api_key}&c
ountry=US&year={today.year}&month={today.month}&day={today.day}"

    holiday_response = requests.get(holiday_url).json()
    if isinstance(holiday_response, dict) and holiday_response.get("response",
{}).get("holidays"):
        holidays = [holiday["name"] for holiday in
holiday_response["response"]["holidays"]]
        output.insert(tk.END, f"Special Holidays Today:\n- " + "\n- ".join(holidays)
+ "\n\n")
    else:
        output.insert(tk.END, "No major holidays today or couldn't retrieve holiday
data.\n\n")

    # 2. Historical Events (Today in History API)
    history_url =
f"https://history.muffinlabs.com/date/{today.month}/{today.day}"
    history_response = requests.get(history_url).json()

```

```
if "data" in history_response:
    events = [event["text"] for event in history_response["data"]["Events"][:3]]
    output.insert(tk.END, "On this day in history:\n- " + "\n- ".join(events) +
"\n\n")
```

```
# 3. Fun Fact (Numbers API)
```

```
number_url = f"http://numbersapi.com/{today.month}/{today.day}/date"
```

```
number_response = requests.get(number_url)
```

```
if number_response.status_code == 200:
```

```
    output.insert(tk.END, f"Fun Fact: {number_response.text}\n\n")
```

```
else:
```

```
    output.insert(tk.END, "Couldn't fetch a fun fact about today.\n")
```

```
# Ending statement
```

```
output.insert(tk.END, "Enjoy your day!\n")
```

```
def handle_special_search(command):
```

```
    if command.lower().startswith("google "):
```

```
        query = command[7:]
```

```
        webbrowser.open(f"https://www.google.com/search?q={query}")
```

```
    elif command.lower().startswith("wikipedia "):
```

```
        query = command[10:]
```

```
        webbrowser.open(f"https://en.wikipedia.org/wiki/{query.replace(' ', '_')}")
```

```
    elif command.lower().startswith("youtube "):
```

```
        query = command[8:]
```

```
        handle_special_search(command,
```

```
"https://www.youtube.com/results?search_query={query}")
```

```
def load_reminders():
```

```
    try:
```

```
        with open("reminders.json", "r") as f:
```

```
            # Check if the file is empty
```

```
            content = f.read().strip()
```

```
            if content: # Only load if there's content
```

```
                return json.loads(content)
```

```
            return [] # Return an empty list if the file is empty
```

```

except FileNotFoundError:
    return [] # Return an empty list if the file doesn't exist
except json.JSONDecodeError: # Handle JSON errors
    messagebox.showerror("Error", "Could not decode reminders.json. It may
be corrupted.")
    return [] # Return an empty list on error


def save_reminders(reminders):
    with open("reminders.json", "w") as f:
        json.dump(reminders, f)


def run_scheduler(output):
    while True:
        schedule.run_pending()
        time.sleep(1)


class ChatbotApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Text-Based Chatbot with To-Do List")
        self.root.geometry("525x770")
        self.root.configure(bg="#1E1E2E") # Dark blue background


        # Frame for Output and Input
        frame_chat = ttk.Frame(root, padding="10", relief=tk.RAISED,
borderwidth=2)
        frame_chat.grid(row=0, column=0, sticky=tk.W + tk.E)
        frame_chat.configure(style="ChatFrame.TFrame")


        # Output area
        self.output = scrolledtext.ScrolledText(frame_chat, width=50, height=15,
wrap=tk.WORD,
                                font=("Arial", 12), bg="#1E1E2E", fg="#000000")
        self.output.grid(row=0, column=0, columnspan=2, padx=10, pady=10)


        # Input field for commands
        self.input_field = ttk.Entry(frame_chat, width=40, font=("Arial", 12),
                                background="#1E1E2E", foreground="#000000")

```

```
self.input_field.grid(row=1, column=0, padx=10, pady=10)

    ttk.Button(frame_chat, text="Send", command=self.handle_input,
style="Accent.TButton").grid(row=1, column=1, padx=5)

    # Frame for To-Do List
    frame_todo = ttk.Frame(root, padding="10", relief=tk.RAISED,
borderwidth=2)
    frame_todo.grid(row=1, column=0, sticky=tk.W + tk.E)
    frame_todo.configure(style="TodoFrame.TFrame")

    tk.Label(frame_todo, text="Add Task:", font=("Arial", 12),
background="#1E1E2E", fg="white").grid(row=0, column=0, padx=10,
pady=(10, 0))

    self.task_entry = ttk.Entry(frame_todo, width=40, font=("Arial", 12),
background="#1E1E2E", foreground="white")
    self.task_entry.grid(row=1, column=0, padx=10, pady=5)

    ttk.Button(frame_todo, text="Add Task", command=self.add_task,
style="Accent.TButton").grid(row=1, column=1, padx=5)

    ttk.Button(frame_chat, text="Switch To Voice",
command=self.switch_to_voice, style="Accent.TButton").grid(row=2,
column=0, padx=10, pady=10)

    tk.Label(frame_todo, text="To-Do List:", font=("Arial", 12),
background="#1E1E2E", fg="white").grid(row=2, column=0, padx=10,
pady=(10, 0))

    self.tasks_listbox = tk.Listbox(frame_todo, width=40, height=10,
font=("Arial", 12), bg="#1E1E2E", fg="white")
    self.tasks_listbox.grid(row=3, column=0, padx=10, pady=5)

    ttk.Button(frame_todo, text="Delete Task", command=self.delete_task,
style="Accent.TButton").grid(row=3, column=1, padx=5, pady=5)

    # Style Configuration for Buttons
    style = ttk.Style()
```

```
style.configure("Accent.TButton", background="#4E6E8E",
foreground="#000000", padding=10, font=("Arial", 10, "bold"))
style.map("Accent.TButton", background=[("active", "#4E6E8E")],
foreground=[("active", "#000000")])
```

```
# Frame Style Configuration
```

```
style.configure("ChatFrame.TFrame", background="#2E2E3E")
```

```
style.configure("TodoFrame.TFrame", background="#2E2E3E")
```

```
# Start the scheduler in a separate thread
```

```
threading.Thread(target=run_scheduler, args=(self.output,),
daemon=True).start()
```

```
def process_command(self, command):
```

```
    self.output.insert(tk.END, f"You: {command}\n")
```

```
# Handle weather command
```

```
if 'weather' in command:
```

```
    city = command.replace("weather", "").strip()
```

```
    if 'in' in city:
```

```
        city = city.replace("in", "").strip()
```

```
    if city:
```

```
        get_weather(city, self.output)
```

```
    else:
```

```
        self.output.insert(tk.END, "Please specify the city.\n")
```

```
elif "joke" in command:
```

```
    get_joke(self.output)
```

```
elif "quote" in command:
```

```
    get_quote(self.output)
```

```
elif "news" in command:
```

```
    get_news_briefing(self.output)
```

```
elif "today" in command:
```

```
    get_today_info(self.output)
```

```

        elif command.lower().startswith("google ") or
command.lower().startswith("wikipedia ") or
command.lower().startswith("youtube "):
            handle_special_search(command)

# Handle reminder command
elif command.lower().startswith("remind me to"):
    command=command.lower()
    parts = command.split(" at ")
    if len(parts) == 2:
        message = parts[0].replace("remind me to ", "").strip()
        time_str = parts[1].strip()
        self.set_reminder(message, time_str)
        self.output.insert(tk.END, "Bot: Reminder Set\n")

    else:
        self.output.insert(tk.END, "Bot: Please use the format: 'Remind me to
<task> at <HH:MM>'.\n")

    else:
        # Use chatterbot to handle general conversations
        response = chatbot.get_response(command)
        self.output.insert(tk.END, f"Bot: {response}\n")

def switch_to_voice(self):
    # Call another Python file (replace 'voice_app.py' with your actual file)
    subprocess.Popen(['python', 'voice_assistant.py']) # Use 'python3' if
needed for Python 3.x
    self.root.destroy() # Close the current window

def handle_input(self):
    command = self.input_field.get()
    self.process_command(command)
    self.input_field.delete(0, tk.END) # Clear the input field

# To-Do List Functions
def add_task(self):
    task = self.task_entry.get()

```

```

if task:
    self.tasks_listbox.insert(tk.END, task)
    self.task_entry.delete(0, tk.END) # Clear the entry box
else:
    messagebox.showwarning("Warning", "Please enter a task.")

def delete_task(self):
    try:
        selected_task_index = self.tasks_listbox.curselection()[0]
        self.tasks_listbox.delete(selected_task_index)
    except IndexError:
        messagebox.showwarning("Warning", "Please select a task to delete.")

def set_reminder(self, message, time_str):
    reminders = load_reminders()

    # Define the job function to display the reminder and remove it
    def job():
        messagebox.showinfo("Reminder", message)
        self.remove_reminder(time_str)

    # Schedule the job
    schedule.every().day.at(time_str).do(job)

    reminders.append({"time": time_str, "message": message})
    save_reminders(reminders)
    messagebox.showinfo("Reminder Set", f"Reminder set for {time_str}: {message}")

def remove_reminder(self, time_str):
    reminders = load_reminders()
    reminders = [reminder for reminder in reminders if reminder["time"] != time_str]
    save_reminders(reminders)

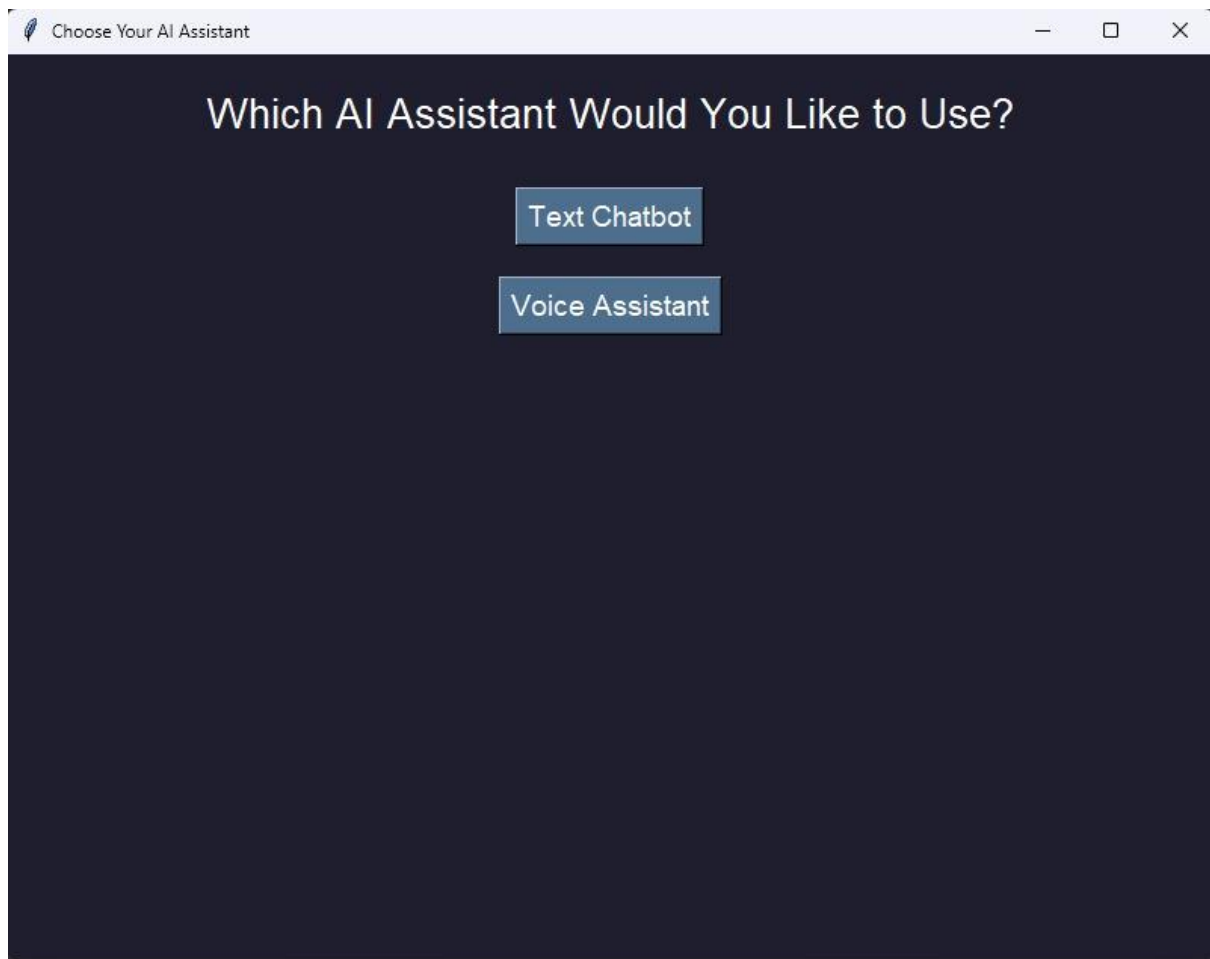
# GUI setup
def main():
    root = tk.Tk()
    app = ChatbotApp(root)

```

```
root.mainloop()
```

```
if __name__ == "__main__":  
    main()
```

## **OUTPUT:-**



Text-Based Chatbot with To-Do List

You: tell me a joke  
What is the least spoken language in the world? - Sign language.  
You: hello  
Bot: Greetings!  
You: How are you?  
Bot: I am doing well.  
You: weather in coimbatore  
Bot: The temperature in coimbatore is 30.88°C with haze.

Enter text

Send

Switch To Voice

Add Task:

Add Task

To-Do List:

Task1  
Task2  
Task4

Delete Task

