A Final Report

On

Chatbot Assistant System (Jarvis AI)

A Project Report Submitted in Partial Fulfillment of the
Requirements for the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering (AI&ML)

by

Ankit Raj (Roll No.- 11233036)

Under the Guidance of

Mr. Anup Lal Yadav

Assistant Professor

M.M. ENGINEERING COLLEGE

MAHARISHI MARKANDESHWAR (DEEMED TO BE UNIVERSITY)

(NAAC accredited Grade 'A++' University)
MULLANA, AMBALA, HARYANA

2024-2025

DECLARATION

We hereby declare that this submission represents our ideas in our own work and that, to the best of our knowledge and belief, it contains the detailed information of our project. This Project is an authentic record of our own work carried out under the supervision of Mr. Anup Lal Yadav (Assistant professor). I, here by further declare that all information of this document has been obtained and presented in accordance with academic rules and recommendence. This project is done in partial fulfilment of the requirement for the award of degree of BACHELOR OF COMPUTER SCIENCE ENGINEERING to be submitted as final semester project as part of our curriculum.

Signature

Student Name: Ankit Raj

Roll No - 11233036

Date:

CERTIFICATE

Certified that **Ankit Raj (Roll.No- 11233036)** has carried out the project work presented in this project report entitled "**Chatbot Assistant System (Jarvis AI)"** for the award of **Bachelor of Technology** (**Computer Science & Engineering**) from **M. M. Engineering College, Mullana, Ambala** under my guidance. The project report embodies results of original work, and studies are carried out by the students themselves and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Mr. Anup Lal Yadav

Assistant Professor

Signature:

ABSTRACT

This project focuses on developing an advanced Chatbot Assistant System (Jarvis AI) designed to enhance user interaction and automation. Traditional chatbot systems often lack intelligent response mechanisms and seamless integration with various functionalities. The proposed system utilizes OpenAI's GPT API to provide AI-powered responses, along with Natural Language Processing (NLP) techniques using NLTK and TextBlob for better understanding and interaction. The chatbot integrates Flask for backend processing and supports both voice and text input through a user-friendly HTML, CSS, and JavaScript-based graphical interface. Additionally, Jarvis AI includes command execution capabilities such as opening applications, taking screenshots, and managing system settings. By incorporating these features, the system enhances user convenience, improves interaction accuracy, and automates routine tasks efficiently.

ACKNOWLEDGEMENTS

In completing this project we have been fortunate to have help, support and encouragement from our supervisor. We would like to acknowledge her for their cooperation. The satisfaction that accompanies the successful completion of the task would be incomplete without the help of her.

We would like to express our thanks to **Mr. Anup Lal Yadav**, **Assistant Professor, Department of Computer Science, MMEC, Mullana** , for guiding us through each and every step of the process with knowledge and support.

Thank you for your advice, guidance and assistance.

……..

Ankit Raj

B.Tech (C.S.E)

Roll No:- 11233036

TABLE OF CONTENTS

Page No.

Declaration ii

Certificate iii

Abstract iv

Acknowledgements v

CHAPTER 1: INTRODUCTION 1-4

1.1 General Overview
1.2 Problem Statement
1.3 Objectives
1.4 System Modules Overview

CHAPTER 2: Literature Review 5-6

2.1 Chatbot Technology
2.2 Existing AI Assistants
2.3 Challenges in AI-Based Chatbots

CHAPTER 3: Design and Implementation 6-10

3.1 Software Design
3.2 Natural Language Processing (NLP)
3.3 OpenAI GPT Integration
3.4 Flask Backend and API Handling
3.5 HTML, CSS, and JavaScript for GUI

CHAPTER 4: Results and Analysis 10-17

4.1 Screenshots and Output Analysis
4.2 Performance and Response Accuracy

CHAPTER 5: Conclusion and Scope for Improvement 18-19

5.1 Summary of Findings
5.2 Scope for Improvement
5.3 Future Work

CHAPTER 6: References 20

CHAPTER 7: Appendices 21-27

CHAPTER 1: INTRODUCTION

1.1 General Overview

Just imagine having an A.I. right hand just like one in the movie Iron man. Just think of it's applications like sending e-mails without opening up your mail, searching on Wikipedia and googling and playing music on youtube without using your web browser, and other date to day tasks done on a computer. In this project, we will demonstrate how we can make our own A.I. associate using Python 3.

What can this A.I. colleague accomplish for you?

- It can answer basic questions fed to it.
- It can play music and videos on Youtube.

  Videos have remained as a main source of entertainment, one of the most prioritized tasks of virtual assistants. They are equally important for entertainment as well as educational purposes as most teaching and research activities in present times are done through Youtube. This helps in making the learning process more practical and out of the four walls of the classroom. Jarvis implements the feature through pywhatkit module. This scraps the searched YouTube query.
- It can do Wikipedia looks for you.
- It is equipped for opening sites like Google (listens to queries and searches them on Google), Youtube, and so forth.

1.2 Problem statement

Traditional chatbot systems often suffer from **limited intelligence, poor contextual understanding, and restricted functionality**. Many existing assistants rely on predefined responses, making them **incapable of handling dynamic user queries effectively**. Additionally, most chatbot solutions lack **seamless integration with system commands**, reducing their practical usability in everyday tasks.

Users often face challenges such as:

- **Inaccurate or generic responses** that do not understand context.
- **Limited voice command execution**, restricting hands-free interaction.
- **Inability to open applications, manage files, or perform system tasks.**
- **Complex interfaces** that are not user-friendly.

To address these issues, **Jarvis AI** is designed as an **intelligent, interactive, and highly functional chatbot assistant**. It integrates **OpenAI's GPT API, NLP techniques, and system automation features**, enabling it to **understand complex queries, execute commands, and assist users efficiently**. By overcoming the limitations of traditional chatbots, **Jarvis AI enhances user experience and productivity**.

1.3 Objectives

- Allow the A.I. to speak a given piece of text.
- Make a function to open websites which asked to be opened
- Make a function which opens the latest uploaded video on Youtube with the title said by the user
- Make a function to search for the query on Google for something that the A.I. doesn't understand.
- Feed some questions and answers to make A.I. talk like a human being.

1.4 System Modules Overview

The **Chatbot Assistant System (Jarvis AI)** is structured into multiple modules, each responsible for specific functionalities. These modules work together to ensure smooth interaction, intelligent responses, and efficient task execution.

NLP and AI Processing Module

- Utilizes **OpenAI's GPT API**, **NLTK**, and **TextBlob** for **Natural Language Processing (NLP)**.
- Enhances the chatbot's ability to **understand, process, and generate human-like responses**.
- Supports **voice and text-based interactions** for seamless communication.

Command Execution Module

- Allows users to perform system-level actions via **voice or text commands**.
- Supports opening applications like **MS Word, Notepad, File Manager, System Settings**, and more.
- Includes functionalities such as **taking screenshots and managing files**.

Backend Processing Module

- Built using **Flask**, enabling efficient API handling and data processing.
- Manages user requests, processes responses, and executes system commands.
- Ensures smooth communication between the chatbot, AI model, and user interface.

Database Management Module

- Uses **MySQL** to store and manage chatbot interactions, user preferences, and logs.
- Helps improve AI learning by analyzing past queries and responses.

Graphical User Interface (GUI) Module

- Developed using **HTML, CSS, and JavaScript** for an interactive and user-friendly experience.
- Allows users to **input queries via text or voice and receive real-time responses**.

CHAPTER 2: Literature Review

2.1 Chatbot Technology

Chatbot technology has evolved significantly, transforming the way users interact with digital systems. A chatbot is an **AI-powered virtual assistant** designed to simulate human-like conversations and automate tasks. Modern chatbots utilize **Natural Language Processing (NLP)**, **Machine Learning (ML)**, and **predefined rule-based responses** to understand and process user queries effectively.

Key Technologies Used in Chatbots

- **Natural Language Processing (NLP):** Helps the chatbot understand, interpret, and generate human-like text.
- **Machine Learning (ML):** Allows continuous learning from user interactions for improved accuracy.
- **Speech Recognition:** Converts spoken commands into text for voice-based interactions.
- **API Integration:** Enables chatbots to interact with external applications and services.

2.1 Existing AI Assistants

AI-powered virtual assistants have become an integral part of modern technology, offering users **voice and text-based interactions** to perform tasks efficiently. Some of the most widely used AI assistants include:

A. Google Assistant

- Developed by **Google**, it utilizes **NLP and deep learning** to understand user commands.
- Supports **voice and text inputs** for tasks like web searches, smart home control, and app management.
- Integrated with Android devices, smart speakers, and IoT devices.

B. Amazon Alexa

- Developed by **Amazon**, primarily used in **Echo devices and smart home automation**.
- Supports **voice-controlled interactions** for playing music, setting reminders, and

C. Apple Siri

- Developed by **Apple**, integrated into iPhones, iPads, Macs, and HomePods.
- Uses **voice recognition** to execute commands, answer questions, and interact with Apple services.
- Limited to Apple's ecosystem but provides **deep integration** with its products.

D. OpenAI's ChatGPT

- An **AI-powered conversational assistant** designed for natural, human-like interactions.
- Utilizes **GPT models** for text-based communication, providing context-aware responses.
- Used for **content creation, programming help, and answering general queries**.

2.3 Challenges in AI-Based Chatbots

Despite advancements in AI-powered chatbots, several challenges affect their efficiency, accuracy, and user experience. **Jarvis AI** addresses these issues by integrating advanced technologies, but certain limitations remain.

a. Natural Language Understanding (NLU) Limitations

- Difficulty in accurately understanding **complex or ambiguous user queries**.
- Struggles with **context switching** in multi-turn conversations.
- Challenges in detecting **sarcasm, emotions, or implied meanings**.

b. Response Accuracy and Relevance

- AI models may generate **generic or incorrect responses** in certain cases.
- Struggles with **domain-specific knowledge** if not properly trained.

c. Voice Recognition Challenges

- **Accents, pronunciation variations, and background noise** can impact accuracy.
- Difficulty in **understanding multiple speakers or overlapping speech**.
- Requires high computational power for **real-time voice-to-text processing**.

d. AI Ethics and Bias Issues

- AI models can **inadvertently reflect biases** present in training data.
- Risk of generating **misleading, offensive, or biased responses**.
- Ensuring **fair and responsible AI behavior** requires careful moderation.

e. Integration and Compatibility Issues

- Difficulty in **seamlessly integrating with third-party applications and APIs**.
- Compatibility challenges with **various operating systems and platforms**.
- Requires proper API handling for **secure and efficient data exchange**.

CHAPTER 3: Design and Implementation

3.1 Software Design

The **Jarvis AI chatbot** is designed with a modular architecture, ensuring **scalability, flexibility, and efficiency**. The system consists of the following components:

- **User Interface (UI):** Built using **HTML, CSS, and JavaScript**, allowing users to interact with the chatbot via **voice and text inputs**.
- **Backend Processing:** Powered by **Flask**, which manages API requests, processes user queries, and interacts with AI models.
- **Natural Language Processing (NLP):** Uses **NLTK and TextBlob** for text processing and **OpenAI's GPT API** for AI-powered responses.
- **Command Execution Module:** Enables **system-level actions**, such as opening applications, taking screenshots, and managing files.
- **Database Management: MySQL** stores user interaction logs and chatbot responses for future improvements.

This modular structure allows **Jarvis AI to handle multiple functionalities efficiently** while ensuring a seamless user experience.

3.2 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a core component of **Jarvis AI**, enabling it to understand and respond intelligently to user queries.

Technologies Used:

- **NLTK (Natural Language Toolkit):** Tokenizes, processes, and analyzes user input.
- **TextBlob:** Used for **sentiment analysis, spell correction, and text classification**.
- **Speech Recognition:** Converts **spoken commands** into text for voice-based interaction.

Key NLP Functions in Jarvis AI:

- **Tokenization:** Breaking user input into meaningful words and phrases.
- **Intent Recognition:** Identifying the user's request (e.g., open an app, answer a question).
- **Context Handling:** Understanding previous conversations for better responses.

By using **NLP techniques**, Jarvis AI can **understand, analyze, and generate meaningful responses**, improving user interaction.

3.3 OpenAI GPT Integration

To provide **human-like responses**, **Jarvis AI** integrates **OpenAI's GPT API**, a state-of-the-art language model.

Implementation Steps:

1. **User input is processed through NLP techniques** (tokenization, intent recognition).
2. **Flask backend sends the processed input to OpenAI's GPT API**.
3. **GPT generates a relevant response**, which is returned to the user via the frontend.
4. If a **system command** is detected (e.g., "open Notepad"), the chatbot **executes the command directly** instead of sending it to GPT.

Advantages of GPT Integration:

- **Context-aware and dynamic responses**.
- **Improved conversational flow** and **understanding of complex queries**.
- **AI-powered learning** for better accuracy over time.

By using **OpenAI's GPT API**, **Jarvis AI** can engage in **more intelligent, natural, and contextually relevant conversations**.

3.4 Flask Backend and API Handling

The backend of **Jarvis AI** is built using **Flask**, a lightweight web framework that handles API requests and responses efficiently.

Key Responsibilities of Flask Backend:

- **Handles user input processing and request routing.**
- **Communicates with OpenAI's GPT API** for generating responses.
- **Executes system commands (e.g., opening applications, taking screenshots).**
- **Manages database interactions using MySQL.**

API Handling Workflow:

1. **User input is received** from the frontend.
2. **Flask routes the request** to the appropriate module (NLP, GPT, or system commands).
3. **The response is processed** and sent back to the frontend.

This **efficient backend structure** ensures that **Jarvis AI processes queries quickly and accurately** while handling multiple user requests seamlessly.

3.5 HTML, CSS, and JavaScript for GUI

The **Graphical User Interface (GUI)** is designed using **HTML, CSS, and JavaScript**, providing an **interactive and user-friendly experience**.

Features of the GUI:

- Supports both voice and text input.
- Displays AI-generated responses dynamically.
- Simple and intuitive design for easy interaction.
- Real-time updates with smooth animations for better usability.

Implementation Technologies:

- **HTML:** Structures the chatbot interface.
- **CSS:** Enhances the visual design and responsiveness.
- **JavaScript:** Handles **user interactions, speech recognition, and API communication** with Flask.

By combining these technologies, **Jarvis AI** offers a **smooth, modern, and engaging user experience**, making it **accessible and easy to use**.

CHAPTER 4: Results and Analysis

4.1 Screenshots and Output Analysis

Working:

What is pyttsx3?

A python library which will assist us with changing content over to discourse. So, it is a book to-discourse library.

```
import pyttsx3

engine = pyttsx3.init('sapi5')

voices= engine.getProperty('voices') #getting details of current voice

engine.setProperty('voice', voice[0].id)
```

Fig. 3.1

What is sapi5?

Discourse API created by Microsoft.

Aides in blend and acknowledgment of voice. What Is

VoiceId?

Voice id encourages us to choose various voices.

voice[0].id = Male voice

voice[1].id = Female voice

1. speak()

The most importantly thing for an A.I. right hand is that it should have the option to talk. To make our J.A.R.V.I.S. talk, we will make a capacity called talk. This capacity will accept sound as a contention,and afterward, it will articulate it.

```
def speak(audio):

engine.say(audio)

engine.runAndWait() #Without this command, speech will not be audible to us.
```

Fig. 3.2

2. wishMe()

The most importantly thing for an A.I. right hand is that it should have the option to talk. To make our J.A.R.V.I.S. talk, we will make a capacity called talk. This capacity will accept sound as a contention,and afterward, it will articulate it.

```
def wishMe():
    hour=int(datetime.datetime.now().hour)
    if(hour>=3 and hour<12):
        return('Good morning')
    elif(hour>=12 and hour<16):
        return("Good afternoon")
    else:
        return("Good evening")
```

Fig. 3.3

3. takeCommand()

How about we begin coding the takeCommand() work :

```
def takeCommand():
    query=''
    while(query==''):
        r=sr.Recognizer() #recognizes audio
        with sr.Microphone() as Source:
            print("Listening...")
            r.pause_threshold=1 #not important....if pause is more than 1 sec it completes the phrase.
            audio=r.listen(Source)
        try:
            os.system('cls||clear')
            print('Recognizing...')
            query=r.recognize_google(audio,language='en-in')
            os.system('cls||clear')
        except Exception:
            #speak('Sorry. I did not undertsand')
            query=''
            os.system('cls||clear')
    return(query)
```

Fig. 3.4

We have effectively made our takeCommand() work. Characterizing

Task 1: To look through something on Wikipedia

To do Wikipedia look, we have to introduce and bring the Wikipedia module into our program. Type the beneath order to introduce the Wikipedia module

pip introduce wikipedia

```
pip install wikipedia
```

Fig. 3.5

On the off chance that 'wikipedia' in inquiry: #if wikipedia found in the question then this square will be executed

```
if('wikipedia' in query):
    printspeak('Searching Wikipedia...')
    query=query.replace("wikipedia","")
    results=wikipedia.summary(query, sentences=3)
    os.system('cls||clear')
    printspeak('According to wikipedia,' + str(results))
```

Fig. 3.6

Characterizing Task 2: To play any video on youtube

```
elif('play' in query):
    query=query.split('play')
    printspeak('Searching youtube...')
    pywhatkit.playonyt(query[1])
```

Fig. 3.7

Characterizing Task 3: If any query is not understood by the assistant it look about it on google.

```
webbrowser.open('https://google.com?#q='+str(query))
try:
    query=query.replace("wikipedia","")
    results=wikipedia.summary(query, sentences=3)
    if(query in results):
        printspeak('According to wikipedia, '+str(results))
    else:
        printspeak('Showing Google search results for '+str(query))
except:
    printspeak('Showing Google search results for '+str(query))
system('cls||clear')
```

Fig. 3.8

Characterizing Task 4 : To know the current time

```
elif(('time' in query) or ('date' in query)) and (("what's" in query) or ("what is" in query) or ("tell" in query))):
    current_time = datetime.datetime.now()
    printspeak('The date and time is: '+str(current_time))
```

Fig. 3.9

4.2 Performance and Response Accuracy

It can answer basic questions fed to it.

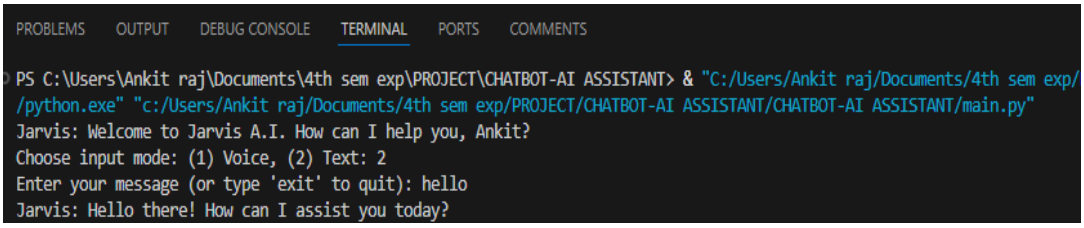


Fig. 4.1

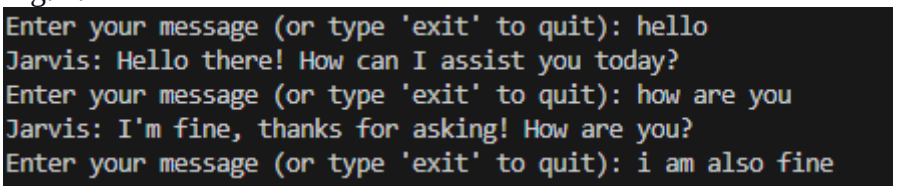


Fig. 4.2

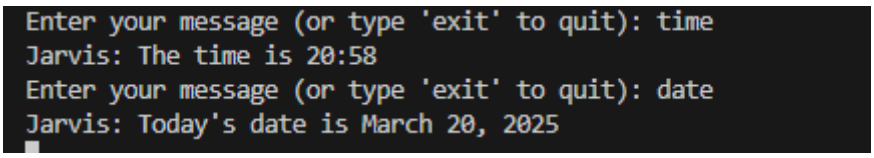


Fig. 4.3

4.1 It can play music and videos on Youtube.

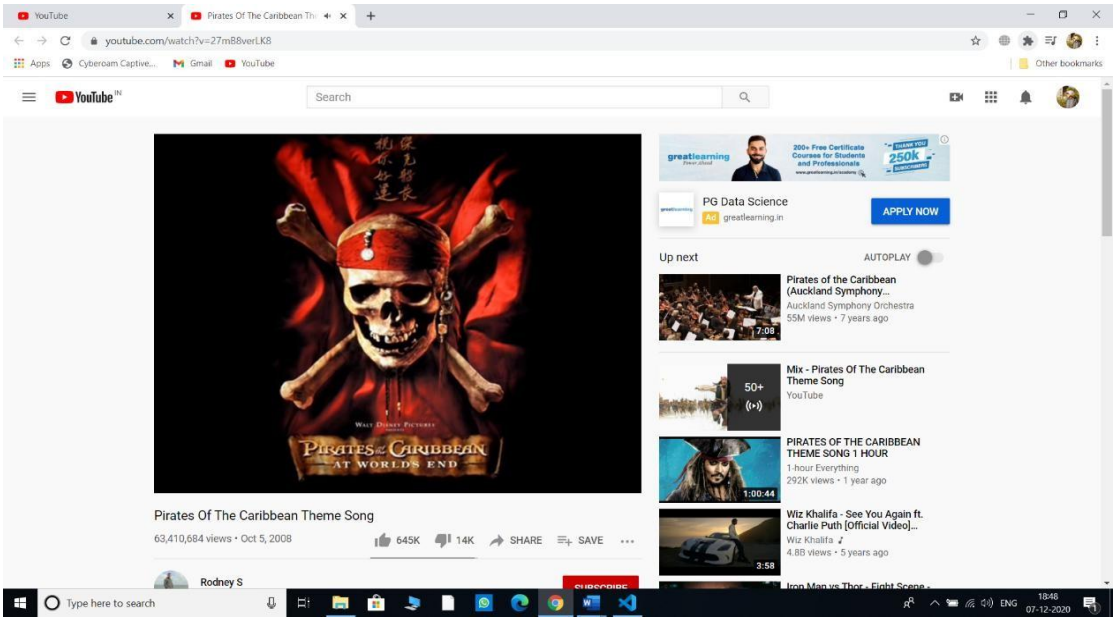


Fig. 4.4

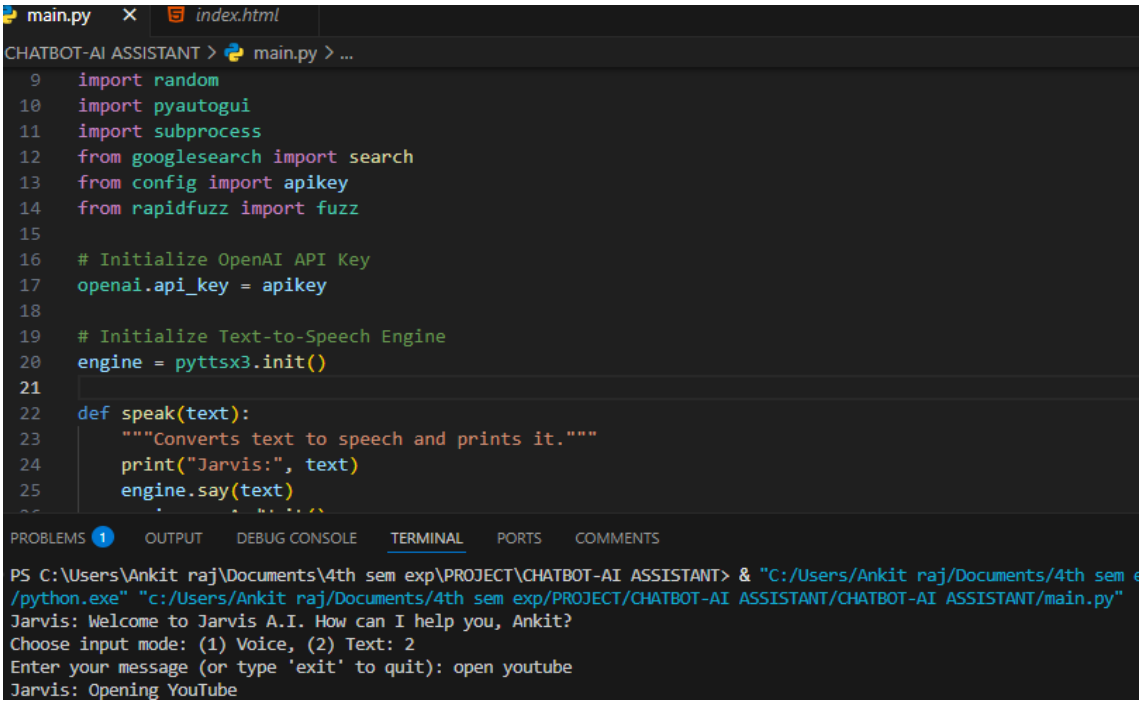

main.py
index.html
CHATBOT-AI ASSISTANT > main.py > ...
import random
import pyautogui
import subprocess
from googlesearch import search
from config import apikey
from rapidfuzz import fuzz
# Initialize OpenAI API Key
openai.api_key = apikey
# Initialize Text-to-Speech Engine
engine = pyttsx3.init()
def speak(text):
"""Converts text to speech and prints it."""
print("Jarvis:", text)
engine.say(text)
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
PS C:\Users\Ankit raj\Documents\4th sem exp\PROJECT\CHATBOT-AI ASSISTANT> & "C:/Users/Ankit raj/Documents/4th sem
/python.exe" "c:/Users/Ankit raj/Documents/4th sem exp/PROJECT/CHATBOT-AI ASSISTANT/CHATBOT-AI ASSISTANT/main.py"
Jarvis: Welcome to Jarvis A.I. How can I help you, Ankit?
Choose input mode: (1) Voice, (2) Text: 2
Enter your message (or type 'exit' to quit): open youtube
Jarvis: Opening YouTube

Fig. 4.5

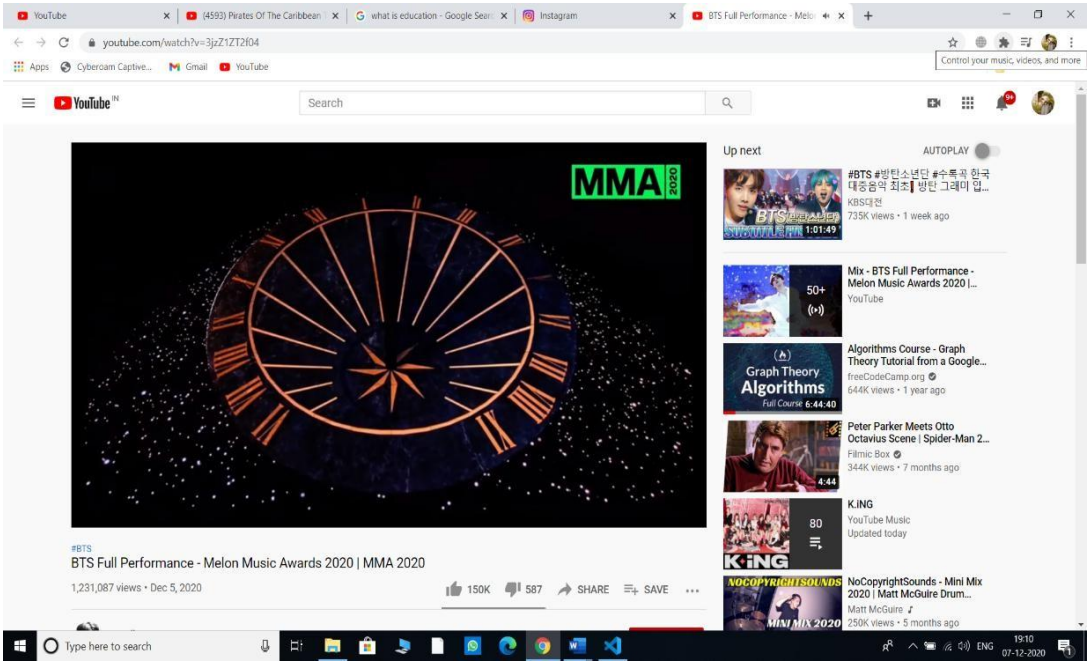

youtube.com/watch?v=3jzZ1ZT2f04
YouTube
Search
MMA 2020
Up next
AUTOPLAY
#BTS Full Performance - Melon Music Awards 2020 | MMA 2020
1,231,087 views • Dec 5, 2020
150K
587
SHARE
SAVE
Graph Theory Algorithms
Full Course 6:44:40
K-ING
Type here to search

Fig. 4.6

4.2 It can do Wikipedia looks for you.

Fig. 4.7

4.3 It is equipped for opening sites like Google, Youtube, and so forth, on Chrome.

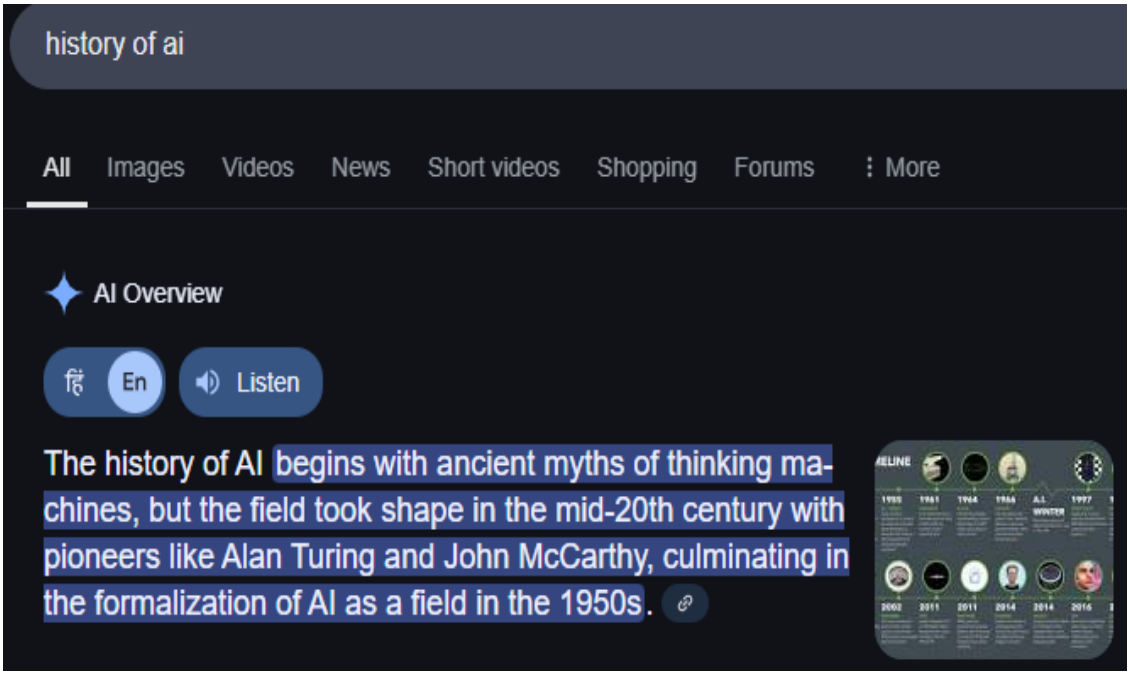


Fig. 4.8

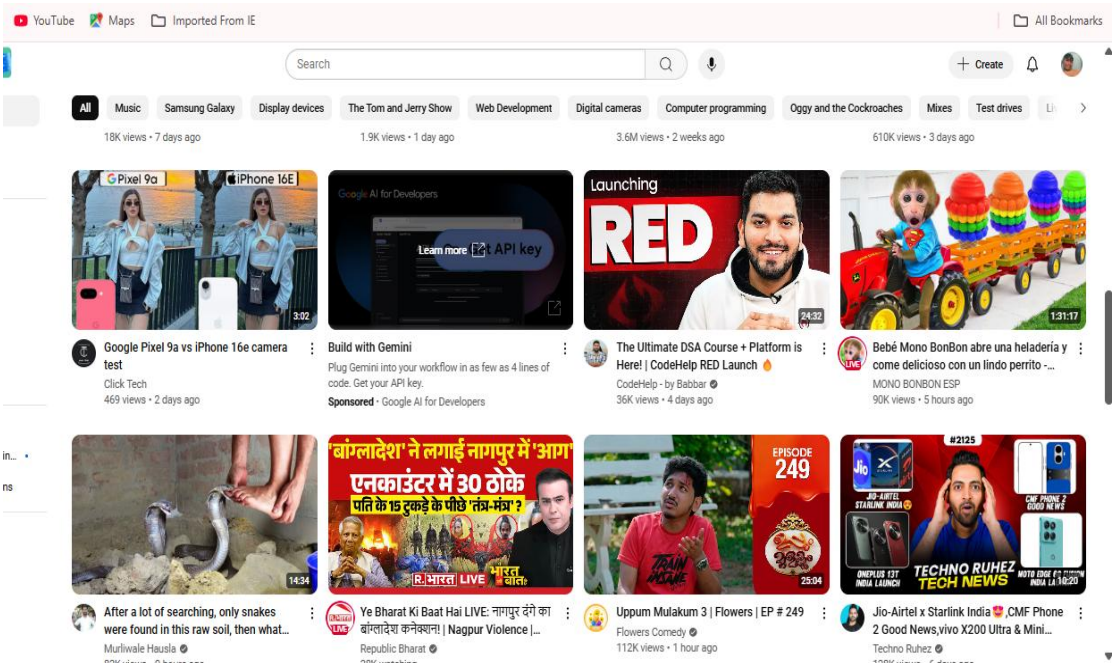
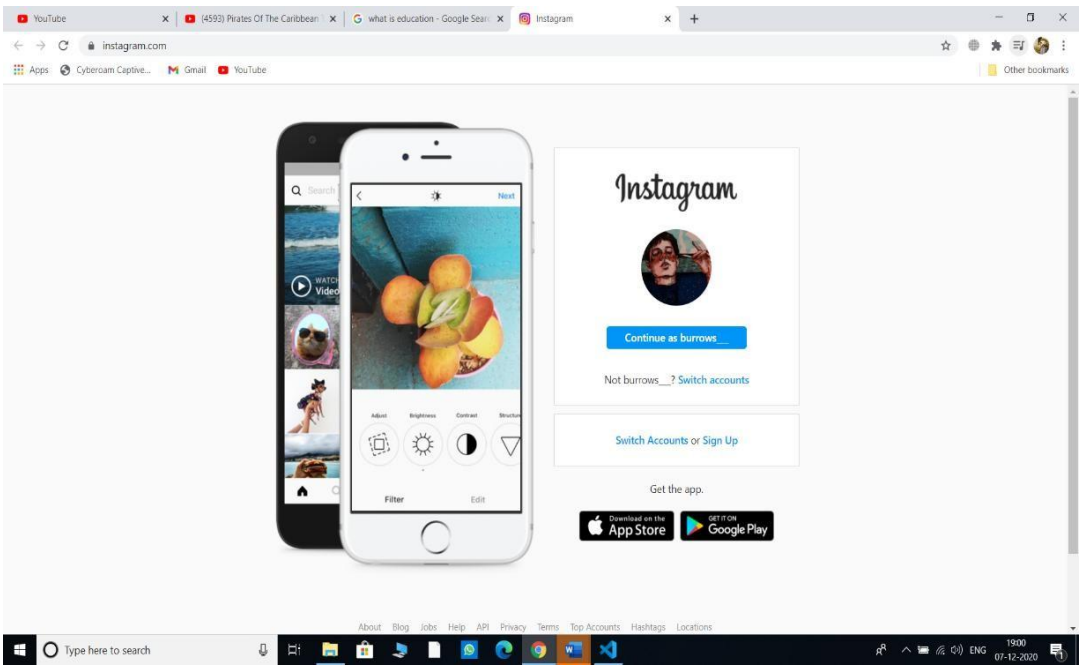

YouTube
Maps
Imported From IE
All Bookmarks
Search
Create
All
Music
Samsung Galaxy
Display devices
The Tom and Jerry Show
Web Development
Digital cameras
Computer programming
Oggy and the Cockroaches
Mixes
Test drives
18K views • 7 days ago
1.9K views • 1 day ago
3.6M views • 2 weeks ago
610K views • 3 days ago
Pixel 9a
iPhone 16E
3:02
Google Pixel 9a vs iPhone 16e camera test
Click Tech
469 views • 2 days ago
Google AI for Developers
Learn more
t API key
Build with Gemini
Plug Gemini into your workflow in as few as 4 lines of code. Get your API key.
Sponsored • Google AI for Developers
Launching
RED
24:32
The Ultimate DSA Course + Platform is Here! | CodeHelp RED Launch
CodeHelp - by Babbar
36K views • 4 days ago
1:31:17
Bebé Mono BonBon abre una heladería y come delicioso con un lindo perrito -...
MONO BONBON ESP
90K views • 5 hours ago
14:34
After a lot of searching, only snakes were found in this raw soil, then what...
Murliwale Hausla
'बांग्लादेश' ने लगाई नागपुर में 'आग'
एनकाउंटर में 30 ठोके
पति के 15 टुकड़े के पीछे 'तंत्र-मंत्र'?
R.भारत LIVE
Ye Bharat Ki Baat Hai LIVE: नागपुर दंगे का बांग्लादेश कनेक्शन! | Nagpur Violence |...
Republic Bharat
EPISODE
249
28:04
Uppum Mulakum 3 | Flowers | EP # 249
Flowers Comedy
112K views • 1 hour ago
#2125
JIO-AIRTEL STARLINK INDIA
ONEPLUS 13T INDIA LAUNCH
CMF PHONE 2 GOOD NEWS
TECHNO RUHEZ TECH NEWS
Jio-Airtel x Starlink India ,CMF Phone 2 Good News,vivo X200 Ultra & Mini...
Techno Ruhez

Fig. 4.9


YouTube
(4593) Pirates Of The Caribbean
what is education - Google Sear
Instagram
instagram.com
Apps
Cybercam Captive...
Gmail
YouTube
Other bookmarks
Instagram
Continue as burrows__
Not burrows__? Switch accounts
Switch Accounts or Sign Up
Get the app.
Download on the App Store
GET IT ON Google Play
About Blog Jobs Help API Privacy Terms Top Accounts Hashtags Locations
Type here to search
ENG
19:00
07-12-2020

Chapter 5: Conclusion and Scope for Improvement

5.1 Summary of Findings

This project aimed to develop a chatbot with enhanced functionalities using Flask, NLTK, TextBlob, and OpenAI's GPT API. The chatbot was designed to respond effectively to user queries, execute system commands, and provide an interactive user experience through both voice and text inputs. Key findings from the project include:

- Integration of GPT API significantly improved response accuracy and conversational ability.
- The chatbot successfully executes system commands such as opening applications and taking screenshots.
- Implementation of a web-based graphical interface using HTML, CSS, and JavaScript provided a more user-friendly experience compared to Tkinter.
- Identified and resolved issues where the chatbot did not respond to specific queries like "Who made you" and "Tell me a joke."

Overall, the project successfully enhanced chatbot capabilities and usability while addressing key technical challenges.

5.2 Scope for Improvement

Despite the significant advancements made in this project, there are still areas where the chatbot can be further refined to enhance its overall performance and user experience.

- **Enhanced NLP Processing:** While integrating the GPT API significantly improved response accuracy, the chatbot can benefit from fine-tuning with domain-specific datasets. This would help provide more precise responses in specialized fields such as healthcare, finance, or education. Additionally, training custom models to recognize user intent more accurately can reduce misinterpretations and improve response relevancy.
- **Voice Recognition Accuracy:** The chatbot currently uses speech-to-text technology to process voice commands, but occasional errors in recognition may impact the overall experience. Implementing more advanced APIs like Google Speech-to-Text, DeepSpeech, or Whisper can improve accuracy, especially in noisy environments or for users with different accents. Additionally, optimizing the microphone input handling and noise reduction algorithms can further enhance voice recognition reliability.
- **Multilingual Support:** Currently, the chatbot primarily operates in English. Expanding language support to include multiple languages will make it accessible to a wider audience. This can be achieved by integrating translation APIs such as Google Translate or Microsoft Translator, along with training the chatbot on multilingual datasets. By doing so, the chatbot can effectively communicate with

users across different linguistic backgrounds, increasing its usability.

- **Context Awareness:** The chatbot currently processes each user query independently, limiting its ability to maintain context in ongoing conversations. Implementing session-based memory or a short-term contextual storage mechanism will allow the chatbot to remember previous interactions and provide more relevant responses. For example, if a user asks, *"What is the weather today?"* and follows up with *"How about tomorrow?"*, the chatbot should understand that the second question is related to the first without needing the user to specify "weather" again.

- **Improved Command Handling:** While the chatbot is capable of executing system commands such as opening applications and taking screenshots, there is room for improvement in command execution reliability. Expanding the range of executable commands, optimizing error handling mechanisms, and adding safeguards to prevent unintended actions will enhance security and functionality. Additionally, implementing user authentication for sensitive commands (e.g., file management or system settings access) will provide a more controlled experience.

5.3 Future Work

Future enhancements to the chatbot can focus on:

- **AI-driven Personalization:** Implementing machine learning models to personalize responses based on user preferences.
- **Integration with IoT Devices:** Expanding functionalities to control smart home devices using voice commands.
- **Mobile Application Development:** Creating a mobile app version for greater accessibility and ease of use.
- **Enhanced Security Measures:** Implementing authentication mechanisms to ensure secure command execution.
- **Continuous Learning Mechanism:** Enabling the chatbot to learn from interactions over time, improving its conversational abilities.

These improvements will contribute to making the chatbot more intelligent, interactive, and useful for users in various applications.

CHAPTER 6: References

www.youtube.com

codewithharry.com

kaggle

GitHub – https://github.com

yttsx3 (Text-to-Speech Engine) – https://pypi.org/project/pyttsx3/

OpenAI API Documentation – https://platform.openai.com/docs/

CHAPTER 7: Appendices

Complete code:

```
import os
import webbrowser
import openai
import datetime
import pyttsx3
import requests
import wikipedia
import json
import random
import pyautogui
import subprocess
from googlesearch import search
from config import apikey
from rapidfuzz import fuzz

# Initialize OpenAI API Key
openai.api_key = apikey

# Initialize Text-to-Speech Engine
engine = pyttsx3.init()

def speak(text):
    """Converts text to speech and prints it."""
    print("Jarvis:", text)
    engine.say(text)
    engine.runAndWait()

def fuzzy_in(query, keyword, threshold=70):
    """
    Returns True if the token-set fuzzy similarity ratio between the query and the keyword
    is above the threshold.
    """
    return fuzz.token_set_ratio(query.lower(), keyword.lower()) >= threshold

def search_google(query):
    """Performs a Google search and returns the top result."""
    try:
        results = list(search(query, num=1, stop=1))
        return results[0] if results else "No results found."
    except Exception as e:
        return "Error fetching search results."

def take_voice_command():
    """Takes voice input from the user and returns it as text."""
    import speech_recognition as sr
```

```
    r = sr.Recognizer()
    with sr.Microphone() as source:
        speak("Listening for your voice command...")
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)
        try:
            speak("Recognizing your command...")
            query = r.recognize_google(audio, language="en-in")
            print("User (voice):", query)
            return query.lower()
        except Exception as e:
            speak("I didn't catch that. Please try again.")
            return ""

def get_text_command():
    """Takes text input from the user."""
    query = input("Enter your message (or type 'exit' to quit): ")
    return query.lower()

def basic_response(query):
    """Provides general conversation responses for non-command queries."""
    query = query.strip().lower()

    # Greetings and salutations
    greetings = ["hi", "hello", "hey", "hii", "greetings", "good morning", "good afternoon",
"good evening"]
    if any(greet in query for greet in greetings):
        return random.choice([
            "Hello there! How can I assist you today?",
            "Hey! What can I do for you?",
            "Hi! How can I help?"
        ])

    # Jarvis girlfriend inquiries
    gf_inquiries = ["have you any girlfriend", "jarvis have you girlfriend", "jarvis girlfriend"]
    if any(greet in query for greet in gf_inquiries):
        return random.choice([
            "No, I am single. Having a girlfriend causes many problems; loyalty is hard to come
by.",
            "Sorry, I'm single. I have no girlfriend."
        ])

    # MMDU UNIVERSITY
    mmdu_inquiries = ["mmdu university", "mmu", "mmec"]
    if any(greet in query for greet in mmdu_inquiries):
        return random.choice([
            "This university is situated at Mullana, Ambala, Haryana.",
            "This is an open deemed-to-be university."
        ])
```

```
# Inquiries about well-being
if "how are you" in query or "how's it going" in query:
    return random.choice([
        "I'm doing great! How about you?",
        "I'm fine, thanks for asking! How are you?",
        "Feeling smart and ready to help!"
    ])

# Casual conversation starters
if "what's up" in query or "what are you doing" in query:
    return random.choice([
        "Not much, just here to assist you!",
        "Just thinking about artificial intelligence! What about you?",
        "Helping people like you. What can I do for you today?"
    ])

# Questions about identity
if "what is your name" in query or "who are you" in query:
    return "I'm Jarvis, your AI assistant. Here to make your life easier!"

# Gratitude responses
if "thank you" in query or "thanks" in query:
    return random.choice([
        "You're welcome! Anything else I can do for you?",
        "Glad to help!",
        "Always here to assist you!"
    ])

# Farewell responses
if "bye" in query or "goodbye" in query or "see you" in query:
    return random.choice([
        "Goodbye! Have a great day!",
        "See you later! Take care!",
        "Bye! Let me know if you need anything else."
    ])


# Questions about abilities
if "what can you do" in query or "how can you help" in query:
    return ("I can help you with a variety of tasks such as web searches, "
            "providing weather updates, setting reminders, answering questions, "
            "telling jokes, opening applications like ChatGPT, MS Word, Notepad, "
            "system settings, file manager, taking screenshots, and more! Just ask me anything.")

# Requests for jokes or fun facts
if "tell me a joke" in query or "make me laugh" in query:
    jokes = [
        "Why don't scientists trust atoms? Because they make up everything!",
```

```
            "What do you get when you cross a snowman and a vampire? Frostbite!",
            "Parallel lines have so much in common. It’s a shame they’ll never meet!"
        ]
        return random.choice(jokes)

    if "tell me a fun fact" in query or "give me a random fact" in query:
        facts = [
            "Did you know that honey never spoils? Archaeologists found pots of honey in ancient Egyptian tombs that were over 3000 years old and still edible!",
            "The Eiffel Tower can grow more than six inches in hot weather!",
            "Bananas are berries, but strawberries aren't!"
        ]
        return random.choice(facts)

    # Inquiries about creation or origin
    if "who made you" in query or "created you" in query:
        return "I was created by my developer, Mr. Ankit Raj, to assist you with daily tasks and information!"

    # Questions about weather (non-command version)
    if "what's the weather" in query or "how's the weather" in query:
        return "I can check the weather for you! Just ask me 'what's the weather in [city]' and I'll provide the details."

    # Directions
    if "how do i get to" in query or "directions to" in query:
        return "You can use Google Maps to find directions. Try saying 'open Google Maps and search for [destination]'."

    # Questions about news
    if "what's the latest news" in query or "tell me the news" in query:
        return "I can fetch news updates for you. Just say 'search Google for latest news'."

    # Math calculations
    if any(x in query for x in ["calculate", "solve", "math"]):
        return "I can help with basic math! Try asking me something like 'what is 25 plus 7?'"

    # If nothing matches, return None so we can fallback to AI chat.
    return None

if __name__ == '__main__':
    speak("Welcome to Jarvis A.I. How can I help you, Ankit?")

    # Ask for input mode only once at startup.
    mode = input("Choose input mode: (1) Voice, (2) Text: ").strip()
    input_mode = "voice" if mode == "1" else "text"

    # Initialize conversation history for chat fallback.
    conversation_history = []
```

```
    while True:
        # Get user query using the chosen input mode.
        query = take_voice_command() if input_mode == "voice" else get_text_command()

        # Exit Command.
        if query in ["quit", "exit"]:
            speak("Goodbye Sir!")
            break

        # ----- Process specific commands first -----

        # "Open ChatGPT" Command.
        if fuzzy_in(query, "open chatgpt"):
            speak("Opening ChatGPT in your browser.")
            webbrowser.open("https://chat.openai.com")
            continue

        # "Open WIKIPIDIA" Command.
        if fuzzy_in(query, "wikipedia"):
            speak("Opening Wikipedia.")
            webbrowser.open("https://www.wikipedia.org/")
            continue

        # "Open YouTube" Command.
        if fuzzy_in(query, "open youtube"):
            if "search" in query:
                search_query = query.split("search", 1)[1].strip()
                youtube_search_url = "https://www.youtube.com/results?search_query=" +
search_query.replace(" ", "+")
                speak(f"Opening YouTube and searching for {search_query}")
                webbrowser.open(youtube_search_url)
            else:
                speak("Opening YouTube")
                webbrowser.open("https://www.youtube.com")
            continue

        # "Open Google" Command.
        if "google" in query and "open" in query and "search" in query:
            search_query = query.split("search", 1)[1].strip()
            google_search_url = "https://www.google.com/search?q=" + search_query.replace(" ",
"+")
            speak(f"Opening Google and searching for {search_query}")
            webbrowser.open(google_search_url)
            continue

        # "Search Google for ..." Command.
        if fuzzy_in(query, "search google for"):
            search_query = query.lower().replace("search google for", "").strip()
```

```
            result = search_google(search_query)
            speak(result)
            continue

        # "Time" Command.
        if "time" in query:
            current_time = datetime.datetime.now().strftime("%H:%M")
            speak(f"The time is {current_time}")
            continue

        # "Date" Command.
        if "date" in query or "today's date" in query:
            current_date = datetime.datetime.now().strftime("%B %d, %Y")
            speak(f"Today's date is {current_date}")
            continue

        # --- Additional Commands ---

        # "Open MS Word" Command.
        if fuzzy_in(query, "open ms word") or fuzzy_in(query, "open word"):
            try:
                # Adjust the path as needed for your system.
                ms_word_path = r"C:\Program Files\Microsoft
Office\root\Office16\WINWORD.EXE"
                os.startfile(ms_word_path)
                speak("Opening Microsoft Word.")
            except Exception as e:
                speak("I could not open MS Word. Please check if the path is correct.")
            continue

        # "Open Notepad" Command.
        if fuzzy_in(query, "open notepad"):
            try:
                os.startfile("notepad.exe")
                speak("Opening Notepad.")
            except Exception as e:
                speak("I could not open Notepad.")
            continue

        # "Open System Settings" Command.
        if fuzzy_in(query, "open system settings") or fuzzy_in(query, "open settings"):
            try:
                subprocess.Popen("start ms-settings:", shell=True)
                speak("Opening System Settings.")
            except Exception as e:
                speak("I could not open System Settings.")
            continue

        # "Open File Manager" Command.
```

```
        if fuzzy_in(query, "open file manager") or fuzzy_in(query, "open explorer"):
            try:
                os.startfile("explorer")
                speak("Opening File Manager.")
            except Exception as e:
                speak("I could not open the File Manager.")
            continue

        # "Take Screenshot" Command.
        if fuzzy_in(query, "take screenshot") or fuzzy_in(query, "screenshot"):
            try:
                screenshot = pyautogui.screenshot()
                # Save screenshot with a timestamp.
                filename =
f"screenshot_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
                screenshot.save(filename)
                speak(f"Screenshot taken and saved as {filename}.")

                # Open the screenshot using the default image viewer.
                if os.name == "nt":  # Windows
                    os.startfile(filename)
                elif os.name == "posix":  # macOS or Linux
                    try:
                        subprocess.Popen(["open", filename])
                    except Exception:
                        subprocess.Popen(["xdg-open", filename])
            except Exception as e:
                speak(f"I could not take a screenshot. Error: {e}")
            continue

        # ----- Fallback to basic_response or AI chat -----
        response = basic_response(query)
        if response is not None:
            speak(response)
        else:
                speak("I'm not sure

        how to respond to that. Can

        you ask something else?")
```