AI-Powered Chatbot using LLM and LangChain

Project report submitted

in partial fulfilment of the requirement for the degree of

Masters in Computer Applications

by

VIPIN KUMAR (2401560075)
PRADEEP SWAIN (2401560056)
HARSH SHARMA (2401560069)
ANUJ KUMAR MANGAL (2401560043)

Under the guidance of

Dr. Tannu Gupta

Department of Computer Science and Engineering

School of Engineering and Technology

K. R. Mangalam University, Gurugram – 122003

November – 2024

DECLARATION

We, the undersigned students of the **Master of Computer Applications** program at **K. R. Mangalam University, Gurugram**, hereby declare that the project report titled:

"AI-Powered Chatbot using LLM and LangChain"

is a **bonafide record of our original work** carried out during the academic session **2024–2025**. This work has been completed in partial fulfilment of the requirements for the award of the **Master of Computer Applications (MCA)** degree under the guidance of **Dr. Tannu Gupta** our project supervisors.

We affirm that the work presented in this report is entirely our own, and has not been submitted elsewhere, in part or full, for the award of any other degree, diploma, or certificate from any other institution or university.

We fully understand that any violation of the above declaration may result in disciplinary action in accordance with the rules and regulations of the university.

Submitted by:

Student Name	**Roll No.**
Vipin Kumar	2401560075
Pradeep Swain	2401560056
Harsh Sharma	2401560069
Anuj Kumar Mangal	2401560043

Place: K.R. Mangalam University

Date: 07 May, 2025

CERTIFICATE

It is certified that the work contained in the project report titled “Snake Game” by the student:

Name of the Student:	**Roll Number:**
Vipin Kumar	2401560075
Pradeep Swain	2401560056
Harsh Sharma	2401560075
Anuj Kumar Mangal	2401560075

has been carried out under our/my supervision and that this work has not been submitted elsewhere.

Signature of Supervisors

Dr. Tannu Gupta
(External Suupervisor)

Date: –07.05.2024

Place: – K.R. Mangalam University

ACKNOWLEDGEMENT

We express our sincere gratitude to all those who supported us during the successful completion of our project titled:

"AI-Powered Chatbot using LLM and LangChain"

We also extend our appreciation to our external mentor, **Dr. Tannu Gupta** for their insightful feedback and technical input.

Our heartfelt thanks to the **Department of Computer Science and Engineering**, **K. R. Mangalam University**, for providing the resources and academic environment that enabled us to complete this work.

We are equally grateful to our peers for their helpful discussions, and to our families for their constant support and motivation.

Place: – K.R. Mangalam University
Date: – 07.05.2025

Vipin Kumar
Pradeep Swain
Harsh Sharma
Anuj Kumar Mangal

ABSTRACT

The emergence of Large Language Models (LLMs) like OpenAI's GPT series has significantly advanced the capabilities of conversational AI. This project presents an **AI-powered chatbot** built using **LangChain**, an open-source framework that enables the integration of LLMs with tools such as memory, vector databases, and prompt engineering techniques. The primary objective of this project is to design a context-aware and intelligent chatbot capable of delivering meaningful and accurate responses to user queries.

A key feature of the chatbot is its integration with the **Google Search API**, which empowers it to retrieve up-to-date information from the web in real-time, thus extending its knowledge beyond the model's training data. The chatbot also incorporates **streaming output** for real-time interactions, **conversation memory** for maintaining context, and **custom tools and chains** for enhanced user experience.

This project demonstrates how combining LLMs with LangChain and external APIs can create intelligent assistants capable of reasoning, retrieving, and responding dynamically. The implementation serves as a foundational example of building production-ready conversational AI systems.

Key Words: Large Language Models (LLMs), LangChain, Conversational AI, Google Search API, Prompt Engineering, Chatbot, Natural Language Processing (NLP), Contextual Memory, Streaming Output, Tool Integration

INDEX

Report Structure

1. **Declaration**
2. **Certificate Page**
3. **Acknowledgement**
4. **Abstract**
5. **Table of Contents (Index)**
6. **Chapters with Full Content**
    - **Introduction**
    - **Objective**
    - **Problem Formulation and Objectives**
    - **Methodology of the Project**
    - **Applications**
    - **Challenges and Limitations**
    - **Implementation**
    - **Results and Analysis**
    - **Conclusion**
7. **References**

INTRODUCTION

In the age of artificial intelligence, chatbots have become an integral part of improving human-computer interaction and automating various tasks. Our **AI-Powered Chatbot** takes this a step further by using **Large Language Models (LLMs)** like OpenAI's GPT along with the **LangChain** framework, allowing the bot to provide real-time, intelligent responses. Unlike traditional chatbots that are limited by static datasets, this chatbot integrates dynamically with external data sources such as **Google Search**, **Wikipedia**, and **Gemini AI**. By doing so, it can fetch the most up-to-date information, making it a highly reliable tool for applications where real-time knowledge retrieval is essential. This feature not only enhances the chatbot's responsiveness but also expands its capabilities across multiple domains like **education**, **customer support**, and **automation**.

The chatbot is also equipped with advanced AI features such as **agent-based decision-making**, which enables it to handle complex user queries by breaking them down and interacting with various data sources intelligently. **Memory retention** further enhances the user experience by allowing the chatbot to remember past interactions, ensuring conversation continuity. **Prompt engineering** refines the way the bot processes user inputs, resulting in more accurate and relevant responses. Additionally, it supports **task automation**, such as adjusting system brightness and volume, offering users an all-in-one tool for both conversational interaction and everyday tasks. However, the chatbot does have some limitations. It depends heavily on a stable internet connection to retrieve real-time data, meaning it cannot function effectively without access to external sources. Furthermore, the chatbot lacks multimedia processing capabilities, meaning it cannot handle images, videos, or audio interactions, which limits its applicability in more interactive, media-rich environments. Despite these limitations, the chatbot remains an advanced and powerful tool for many real-world applications.

AI-Powered Chatbot: A Step Beyond Traditional Bots

Unlike conventional chatbots that rely solely on pre-existing, static datasets, our AI-powered chatbot is designed to continuously interact with and access up-to-date information from the web. By incorporating external data sources like Google Search, Wikipedia, and Gemini AI, the chatbot ensures that users receive the most current and accurate responses. This integration of real-time data retrieval makes the chatbot not just a conversational agent, but a powerful tool for providing useful information, answering questions, and assisting with tasks that require the latest knowledge. This makes the chatbot ideal for applications in education, where up-to-date information is critical, customer support, where accurate and timely responses are essential, and task automation, where the chatbot can streamline day-to-day operations.

Key Features:

1. **LangChain Framework**: Connects to external data sources and helps structure reasoning for more intelligent responses.

2. **Real-Time Data Retrieval**: Fetches the latest updates from sources like Google and Wikipedia for accurate answers.

3. **Agent-Based Decision Making**: Uses agents to structure queries and responses, improving efficiency in handling complex tasks.

4. **Memory Retention**: Remembers past interactions, offering personalized and coherent responses for conversational continuity.

5. **Advanced Prompt Engineering**: Refines user input to interpret complex or ambiguous questions with greater accuracy.

6. **Task Automation**: Automates basic system tasks like adjusting brightness and volume, enhancing usability.

OBEJECTIVE

The primary objectives of this project include:

• Developing a chatbot that provides context-aware responses using LLMs and continuously improves interactions through adaptive learning techniques.

• Integrating real-time knowledge retrieval tools such as Google Search, Wikipedia, and Gemini AI, ensuring that users receive the most accurate and recent information available.

• Implementing memory mechanisms to maintain conversation continuity, allowing the chatbot to provide meaningful responses based on previous interactions.

• Utilizing prompt engineering to optimize chatbot responses, ensuring that the chatbot understands user intent and delivers precise and coherent replies.

• Automating tasks such as web searches, system controls (brightness, volume), and knowledge extraction, making the chatbot a versatile tool for various applications.

• Ensuring the chatbot is scalable and user-friendly for a variety of applications, enabling integration with multiple platforms including websites, mobile applications, and enterprise solutions.

• Enhancing user experience by incorporating natural language understanding (NLU) to improve the chatbot's ability to comprehend and process complex queries.

• Improving response efficiency by leveraging optimized algorithms and caching mechanisms to reduce latency and enhance processing speed.

• Providing a secure and privacy-conscious interaction by implementing data protection measures and ensuring compliance with user data privacy regulations.

By achieving these objectives, the chatbot will serve as a cutting-edge AI assistant capable of providing real-time, intelligent, and interactive solutions for users in various domains.

PROBLEM FORMULATION AND OBJECTIVES

Problem Formulation

The rapid growth of artificial intelligence (AI) has led to the development of chatbots that can simulate human-like conversations, assist in customer support, provide information, and automate routine tasks. However, traditional chatbots have limitations, particularly when it comes to handling complex queries and providing up-to-date information. Many chatbots rely on static datasets, which restrict their ability to offer accurate and relevant answers to real-time, evolving questions.

This project aims to address the limitations of traditional chatbots by developing an AI-powered chatbot that can:

1. Integrate real-time data from external sources like Google, Wikipedia, and Gemini AI to provide current, accurate responses.

2. Enhance conversational continuity through memory retention, enabling the chatbot to offer personalized interactions and remember past conversations.

3. Support task automation, such as adjusting system settings, making the chatbot a more practical tool for daily use.

4. Leverage advanced AI technologies such as LangChain and prompt engineering to process and respond to complex user queries in a structured and context-aware manner.

The primary problem this project seeks to solve is the inability of traditional chatbots to provide dynamic, real-time information and offer personalized, context-driven responses. By leveraging the LangChain framework and external data sources, this chatbot aims to provide a more intelligent, efficient, and adaptable solution for users.

Solution Approach

The AI-Powered Chatbot is designed to overcome the limitations of traditional chatbots by integrating advanced AI technologies, external data sources, and task automation features. The solution is modular, scalable, and built to enhance user interaction through real-time information retrieval, memory retention, and voice-enabled functionality.

1. System Architecture

- User Interface (UI): Allows text and voice-based user interaction.
- Natural Language Understanding (NLU): Processes and interprets user queries for better comprehension.
- LangChain Framework: Manages interaction with external data sources like Google Search, Wikipedia, and Gemini AI for real-time data retrieval.
- Memory System: Retains information from previous conversations, enabling personalized, context-aware responses.
- Task Automation Module: Supports system tasks like volume and brightness control, enhancing practical utility.
- Voice Input/Output: Supports Hindi and English through Speech Recognition and pyttsx3 for voice interaction.

2. Integration with External APIs

- Google Search API: Retrieves real-time search results for dynamic responses.
- Wikipedia API: Provides general knowledge and factual data.
- Gemini AI: Offers advanced reasoning capabilities for complex queries.

3. Memory Retention

- The chatbot remembers past interactions to provide continuity, personalized responses, and improve engagement over time.

4. Prompt Engineering

- Optimizes how the chatbot interprets and responds to user input by refining queries, handling ambiguity, and ensuring context-aware answers.

5. Task Automation

- Automates simple system tasks, such as adjusting device brightness or volume, making the chatbot more useful beyond conversational interactions.

6. Voice Functionality

- Speech Recognition: Converts voice commands into text.

- pyttsx3: Converts text responses into speech in multiple languages for an accessible, hands-free user experience.

7. Challenges & Future Improvements

- Internet Dependency: The chatbot requires an active internet connection for real-time data.

- No Multimedia Processing: Currently limited to text and voice interactions, with no image/video processing capabilities.

- Scalability: Future work involves optimizing infrastructure for better scalability.

METHODOLOGY OF THE PROJECT

Technology Stack

The project is built upon the following technologies:

1. **LangChain**: Serves as the framework for LLM integration and modular AI implementation.

2. **Large Language Model (LLM)**: Used for generating text-based responses.

3. **Agents**: Follow a cycle of thinking, observing, and responding for intelligent decision-making.

4. **Memory**: Maintains chat continuity to provide more meaningful interactions.

5. **Prompt Engineering**: Optimizes queries and response generation for better user engagement.

6. **Tools**: Integrates Google Search, Wikipedia, and YouTube search for real-time knowledge extraction.

Development Process

- Setting up LangChain and integrating it with the chosen LLM.

- Implementing APIs for real-time information retrieval.

- Designing an intuitive chatbot interface.

- Refining chatbot performance through rigorous testing and optimization.

- Deploying and iterating based on user feedback.

Application

The chatbot can be applied in various domains:

• **Education** – Acts as an AI tutor providing real-time answers to student queries, explaining concepts interactively, assisting with homework, and generating study materials for various subjects.

• **Customer Support** – Automates customer service queries by offering instant responses, reducing response time, handling common inquiries, and escalating complex issues to human agents when necessary.

• **Research and Development** – Assists researchers by gathering, analyzing, and summarizing the latest information from multiple sources, helping in literature reviews, data analysis, and trend forecasting.

• **General Knowledge Assistance** – Helps users stay updated on various topics such as current affairs, science, technology, and entertainment by pulling real-time data from reliable sources.

• **Task Automation** – Performs small system tasks like volume and brightness control, scheduling reminders, setting alarms, and managing productivity tasks to enhance user convenience and efficiency.

• **Healthcare Assistance** – Provides general health-related guidance, symptom checking based on existing medical databases, and assists in retrieving information about medicines, diseases, and treatment options.

• **Business Support** – Aids organizations by streamlining workflows, automating routine administrative tasks, assisting in report generation, and enhancing communication efficiency.

• **Personal Assistant** – Acts as a virtual assistant by managing calendars, setting reminders, and helping users with day-to-day activities, making interactions more seamless and productive.

With these applications, the chatbot becomes a valuable tool that extends beyond basic conversation, providing intelligent, real-time solutions across different industries and user needs.

Challenges and Limitations

Developing an AI-powered chatbot that integrates Large Language Models (LLMs) with real-time data retrieval presents various challenges and limitations. These factors influence the chatbot's efficiency, accuracy, and scalability.

Challenges

- **Ensuring Response Accuracy When Fetching External Data**
  - The chatbot retrieves information from multiple sources, such as Google Search, Wikipedia, and Gemini AI.
  - Filtering misinformation and verifying the credibility of sources is a critical concern.
- **Managing Latency Issues When Integrating Multiple APIs**
  - Fetching real-time data from multiple external APIs may introduce delays, leading to slower chatbot responses.
  - Optimizing API calls, caching frequently requested data, and using efficient query-processing mechanisms are necessary to enhance speed.
- **Optimizing Prompt Engineering for Different Types of Queries**
  - The chatbot must understand diverse user inputs, ranging from casual conversations to complex technical queries.
  - Handling ambiguous queries and ensuring the chatbot correctly interprets user intent is an ongoing challenge.

Limitations

- **Requires an Active Internet Connection for Real-Time Knowledge Retrieval**
  - Since the chatbot depends on external APIs for real-time information, it cannot function effectively in offline mode.
  - Users in low-connectivity regions may experience inconsistent performance.
- **Supports Only Text-Based Interactions**
  - The chatbot currently lacks support for processing images, videos, or voice inputs.
  - Many modern chatbots support multimodal interactions, including speech recognition and image-based search, which this project does not currently implement.

IMPLEMENTATION

Agents.py

```
from langchain.agents import initialize_agent, AgentType

from .Llm import llm

import warnings

from .Template import TEMPLATE

from History import get_chat_history

from Preferences import get_preference

from Real_Time_Info import get_real_time_data

from .Tool_Initilization import tools


# Ignore deprecation warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)


# Initialize the agent with the specified parameters

agent = initialize_agent(

    tools=tools,

    llm=llm,

    agent=AgentType.STRUCTURED_CHAT_ZERO_SHOT_REACT_DESCRIPTION,

    verbose=False,

    prompt=TEMPLATE,
```

```
    handle_parsing_errors=True,
)


# Run Agent
def Run_Agent(Query: str) -> str:
    try:
        Format_Prompt = TEMPLATE.format(
            real_time_info = get_real_time_data(),
            preferences = get_preference(),
            chat_history = get_chat_history(),
            query = Query
        )


        response = agent({
            "input": Format_Prompt,
            })


        return response.get("output", "").replace("*", "").replace("`","").replace("\n","")


    except Exception as e:
        return f"Error processing request: {str(e)}"
```

Prefrences.py

```
import os

from .Template import PREFERENCE_TEMPLATE

from .Llm import llm

from .Setup import Preference_Path


# Function to update preferences based on the conversation

def set_preference(Conversation):

  # Check if the preference file exists

  if os.path.exists(Preference_Path):

    # Read existing preferences from the file

    with open(Preference_Path, 'r') as file:

      Existing_Preferences = file.read()

  else:

    Existing_Preferences = None


  # Format the prompt with existing preferences and conversation

  formatted_prompt = PREFERENCE_TEMPLATE.format(

    Existing_Preferences=Existing_Preferences,

    Conversation=Conversation

  )
```

```
    # Invoke the LLM with the formatted prompt
    result = llm.invoke(formatted_prompt)


    # Write the updated preferences back to the file
    with open(Preference_Path, 'w') as file:
        file.write(result.content)


# Function to read the current preferences from the file
def get_preference():
    # Check if the preference file exists
    if os.path.exists(Preference_Path):
        # Read and return the preferences from the file
        with open(Preference_Path, 'r') as file:
            return file.read()
```

History.py

```
import json
import os
from datetime import datetime
from .Setup import Chat_History_Path
```

```
def load_chat_history():
    """Load chat history from the JSON file, handling errors gracefully."""
    if not os.path.exists(Chat_History_Path):
        return {"Chat_history": []}


    try:
        with open(Chat_History_Path, "r", encoding="utf-8") as file:
            return json.load(file)
    except (json.JSONDecodeError, IOError):
        # Handle corrupted or inaccessible file
        return {"Chat_history": []}


def save_chat_history(data):
    """Save chat history to the JSON file safely."""
    try:
        with open(Chat_History_Path, "w", encoding="utf-8") as file:
            json.dump(data, file, indent=4, ensure_ascii=False)
    except IOError as e:
        print(f"Error saving chat history: {e}")


def set_chat_history(query, response):
```

```
    """Add a new chat entry with a timestamp while keeping only the last 5 entries."""
    data = load_chat_history()

    # Append new chat entry with timestamp
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    data.setdefault("Chat_history", []).append({
        "Timestamp": timestamp,
        "Query": query,
        "Response": response
    })

    # Keep only the last 5 entries
    data["Chat_history"] = data["Chat_history"][-5:]

    save_chat_history(data)

def get_chat_history():
    """Retrieve the latest chat history entries."""
    return load_chat_history().get("Chat_history", [])
```

Real Time info

```
from datetime import datetime


def get_real_time_data():
    """Returns the current time, date, and day."""
    now = datetime.now()
    return {
        "time": now.strftime('%I:%M %p'),
        "date": now.strftime('%d %B %Y'),
        "day": now.strftime('%A')
    }
```

Tools

```
# Import necessary modules
from langchain_google_community import GoogleSearchAPIWrapper
import os
from Config import GOOGLE_CSE_ID, GOOGLE_API_KEY


os.environ["GOOGLE_CSE_ID"] = GOOGLE_CSE_ID
os.environ["GOOGLE_API_KEY"] = GOOGLE_API_KEY
```

```
Google_Search_Description = """

This tool allows users to perform a Google search and retrieve relevant information.


## Parameters:

- `query` (str): The search term or query entered by the user.

"""


# Initialize Google Search API Wrapper

class CustomGoogleSearchAPIWrapper(GoogleSearchAPIWrapper):

    def run(self, query):

        return super().run(query)


G_search = CustomGoogleSearchAPIWrapper()
```

Speak.py

```
import os

import io

import pygame

from google.cloud import texttospeech

from Config import GOOGLE_SERVICE_CREDENTIALS


# Set Google Cloud credentials
```

```
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] =
GOOGLE_SERVICE_CREDENTIALS


# Initialize Google TTS Client
client = texttospeech.TextToSpeechClient()


# Voice configuration
voice = texttospeech.VoiceSelectionParams(
    language_code="en-IN",
    name="en-IN-Chirp-HD-D",
    ssml_gender=texttospeech.SsmlVoiceGender.MALE
)


# Audio configuration
audio_config = texttospeech.AudioConfig(
    audio_encoding=texttospeech.AudioEncoding.MP3
)


# Initialize pygame mixer
pygame.mixer.init()


def speak(text: str):
```

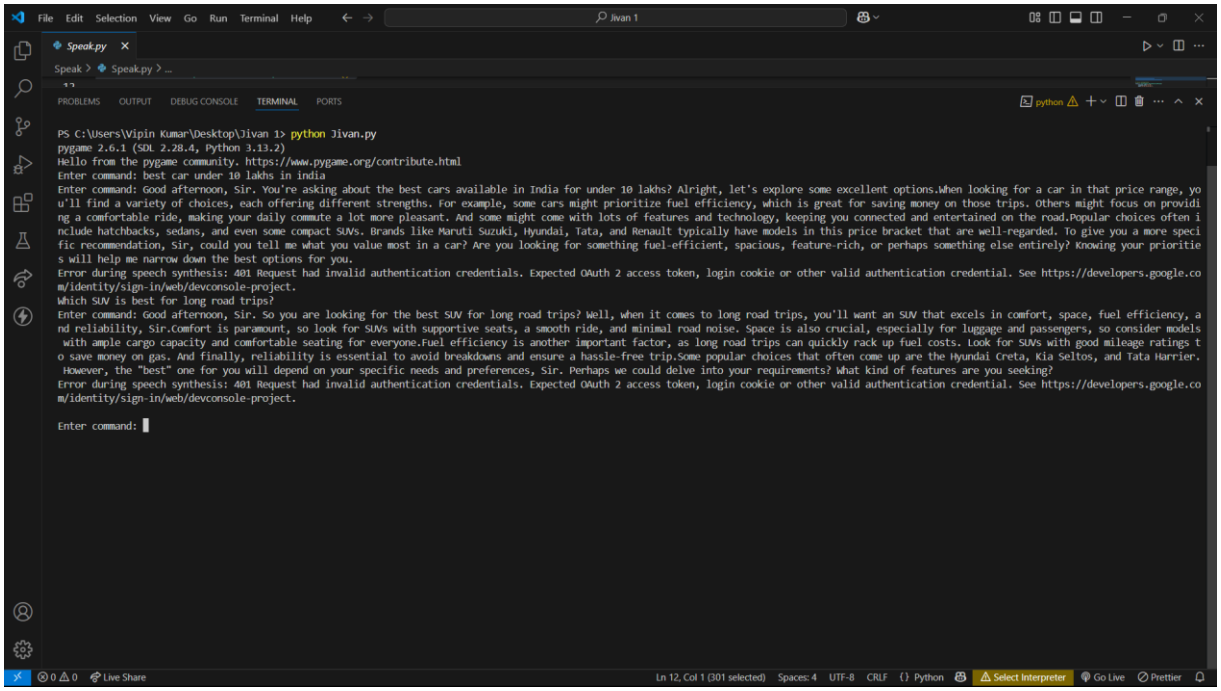
```
"""Synthesizes speech from text and plays it using pygame."""
try:
    # Synthesize speech
    synthesis_input = texttospeech.SynthesisInput(text=text)
    response = client.synthesize_speech(
        input=synthesis_input, voice=voice, audio_config=audio_config
    )

    # Load audio directly from memory
    audio_stream = io.BytesIO(response.audio_content)

    # Ensure pygame mixer is initialized
    if not pygame.mixer.get_init():
        pygame.mixer.init()

    # Play audio
    sound = pygame.mixer.Sound(audio_stream)
    sound.play()

    # Wait until playback is finished
    while pygame.mixer.get_busy():
```

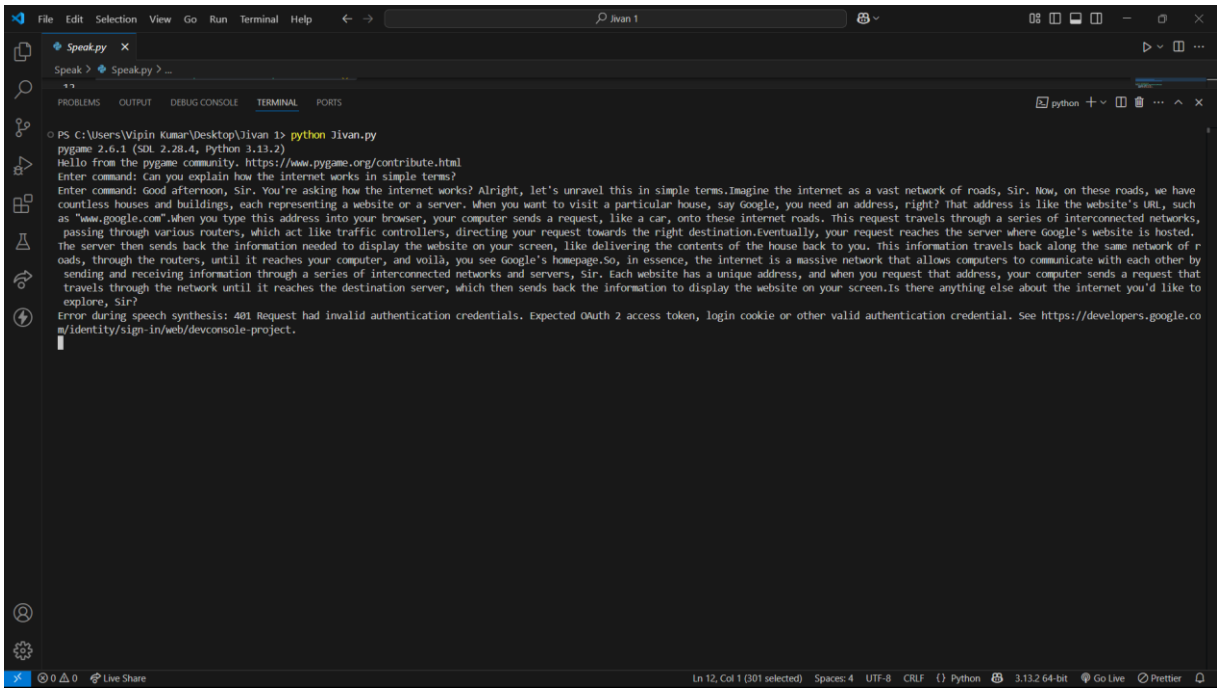
```
        pygame.time.wait(100)


    except Exception as e:

        print(f"Error during speech synthesis: {e}")


    finally:

        pygame.mixer.quit()
```

Jivan.py

```
from Jivan import QUERIES

from Scheduler import Queries_Scheduler, Speak_Scheduler

import threading

import logging

from Speak import *

logging.getLogger().addHandler(logging.NullHandler())

threading.Thread(target=Queries_Scheduler, daemon=True).start()

threading.Thread(target=Speak_Scheduler, daemon=True).start()


while True:

    command = input("Enter command: ")

    QUERIES.append(command)
```

Result and Analysis





CONCLUSION

Our AI-powered chatbot aims to enhance chatbot functionality by integrating LangChain and LLM to provide real-time, context-aware, and automated interactions. Unlike traditional models, this chatbot will remain updated by utilizing external data sources, making it a valuable tool for education, research, and customer service. Future advancements may include multimedia processing and improved task automation.

References

1. **LangChain Documentation – https://python.langchain.com/docs/**
2. **OpenAI API Documentation – https://platform.openai.com/docs/**
3. **Google Gemini API – https://ai.google.dev/gemini-api**
4. **Wikipedia API – https://www.mediawiki.org/wiki/API:Main_page**