Date:24.09.25

TASK:9

To Build an Intelligent **Chatbot system** with Python and Dialog-flow using Interactive Text Mining Framework for Exploration of Semantic Flows in Large Corpus of Text.

To Build an Intelligent Chatbot system with Python and Dialog-flow using Interactive Text Mining Framework for Exploration of Semantic Flows in Large Corpus of Text. **CO4** **S3**

- To integrate with Google Cloud Speech-to-Text and third-party services such as Google Assistant, Amazon Alexa, and Facebook Messenger.
- Configure Dialogflow to manage your data across GCP services and let you optionally integrate Google Assistant.

Tools- Python, Dialog-flow Framework

DATE:24.09.25

TO BUILD AN INTELLIGENT **CHATBOT SYSTEM** WITH PYTHON AND DIALOG-FLOW USING INTERACTIVE TEXT MINING FRAMEWORK FOR EXPLORATION OF SEMANTIC FLOWS IN LARGE CORPUS OF TEXT

AIM:

To build an intelligent chatbox system with Python and dialog-flow using interactive text mining framework for exploration of semantic flow in large corpus of Text

ALGORITHM:

Steps to create an intelligent chatbot using OpenAI APIs:

1. Sign up for OpenAI API access at https://beta.openai.com/signup/. Once you sign up, you will receive your API key.

2. Choose the type of chatbot you want to create. For example, you can create an FAQ chatbot, a customer support chatbot, or a conversational chatbot.

3. Use OpenAI's GPT-3 language model to generate responses to user input. You can use the API to train the language model on your chatbot's intended use case/s.

4. Use Natural Language Processing (NLP) techniques to understand user input and provide relevant responses. You can use OpenAI's API to extract entities (such as dates and names) from user input.

5. Use Machine Learning to continually improve the chatbot's ability to understand and respond to user input.

6. Integrate the chatbot with your preferred messaging platform or channel (e.g., web chat, social media, etc.) using API connectors.

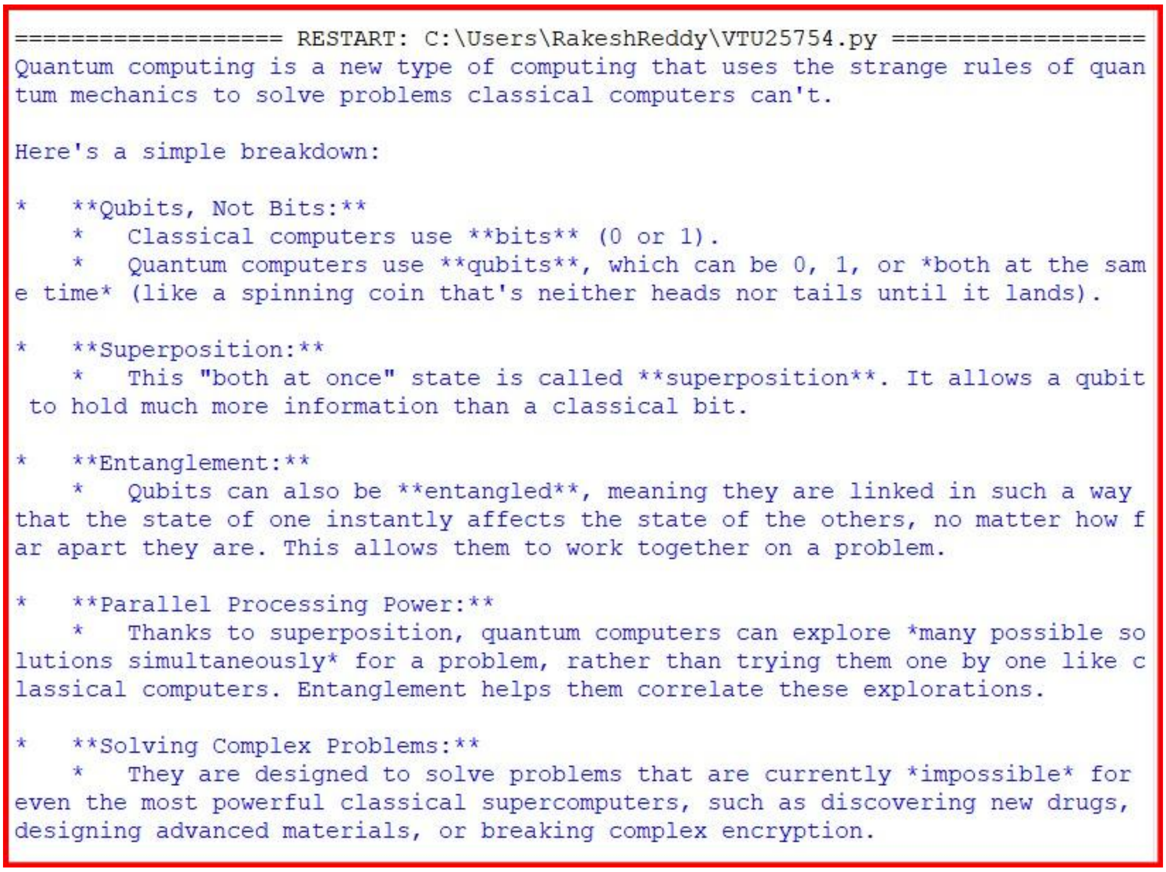
7. Test your chatbot frequently, and use user feedback to improve its performance and provide the best possible experience for your users.

A. SIMPLE CHATGPT USING GEMINI

CODE:

```
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(

   model="gemini-2.5-flash",  # Or "gemini-1.5-pro-latest" if available

     google_api_key="AIzaSyCp7RYEV2grZ3GkemVEGyqFQW_LXF9fUk4",  # Keep this secure!

   temperature=0.7

)

response = llm.invoke("Explain quantum computing simply,breif in points")

print(response.content)
```

OUTPUT:



```
=================== RESTART: C:\Users\RakeshReddy\VTU25754.py ==================
Quantum computing is a new type of computing that uses the strange rules of quan
tum mechanics to solve problems classical computers can't.

Here's a simple breakdown:

*    **Qubits, Not Bits:**
     *    Classical computers use **bits** (0 or 1).
     *    Quantum computers use **qubits**, which can be 0, 1, or *both at the sam
e time* (like a spinning coin that's neither heads nor tails until it lands).

*    **Superposition:**
     *    This "both at once" state is called **superposition**. It allows a qubit
 to hold much more information than a classical bit.

*    **Entanglement:**
     *    Qubits can also be **entangled**, meaning they are linked in such a way
that the state of one instantly affects the state of the others, no matter how f
ar apart they are. This allows them to work together on a problem.

*    **Parallel Processing Power:**
     *    Thanks to superposition, quantum computers can explore *many possible so
lutions simultaneously* for a problem, rather than trying them one by one like c
lassical computers. Entanglement helps them correlate these explorations.

*    **Solving Complex Problems:**
     *    They are designed to solve problems that are currently *impossible* for
even the most powerful classical supercomputers, such as discovering new drugs,
designing advanced materials, or breaking complex encryption.
```

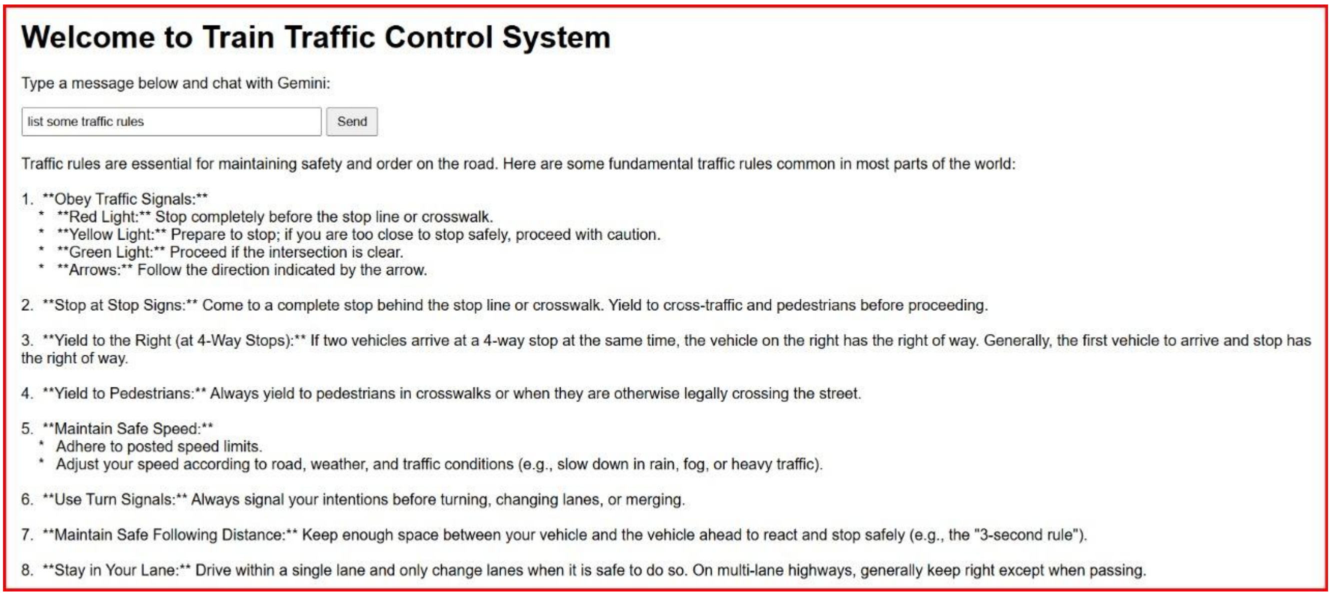
B. CHATGPT ASSISTANT USING GEMINI

CODE:

```
# gemini_chatbot.py


from flask import Flask, request, jsonify

import os

from google import genai

from google.genai import types


app = Flask(__name__)

GEMINI_API_KEY="AIzaSyCp7RYEV2grZ3GkemVEGyqFQW_LXF9fUk4"


# --- Configure API Key ---

# Using the hardcoded API key from above

api_key = GEMINI_API_KEY


# Initialize the client

client = genai.Client(api_key=api_key)


# Choose the Gemini model you want to use

MODEL = "gemini-2.5-flash"  # or “gemini-2.5-pro” etc, depending on access


def generate_reply_from_gemini(prompt: str) -> str:

    """

    Send the user prompt to Gemini and return the response text.

    """
    response = client.models.generate_content(
        model=MODEL,
        contents=prompt,
        # You can optionally provide a config, e.g. thinking_budget etc.
        # config=types.GenerateContentConfig(thinking_config=types.ThinkingConfig(thinking_budget=0))
    )
    return response.text


@app.route("/")
def home():
    return app.send_static_file('index.html')


@app.route("/chat", methods=["POST"])
def chat():
    data = request.get_json()
    user_message = data.get("message", "")
    if not user_message:
        return jsonify({"error": "No message provided"}), 400

    try:
        reply = generate_reply_from_gemini(user_message)
        return jsonify({"reply": reply})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

```
if __name__== "__main__":
    # Run in debug for development
    app.run(host="0.0.0.0", port=5000, debug=True)
```

OUTPUT:



Welcome to Train Traffic Control System

Type a message below and chat with Gemini:

Traffic rules are essential for maintaining safety and order on the road. Here are some fundamental traffic rules common in most parts of the world:

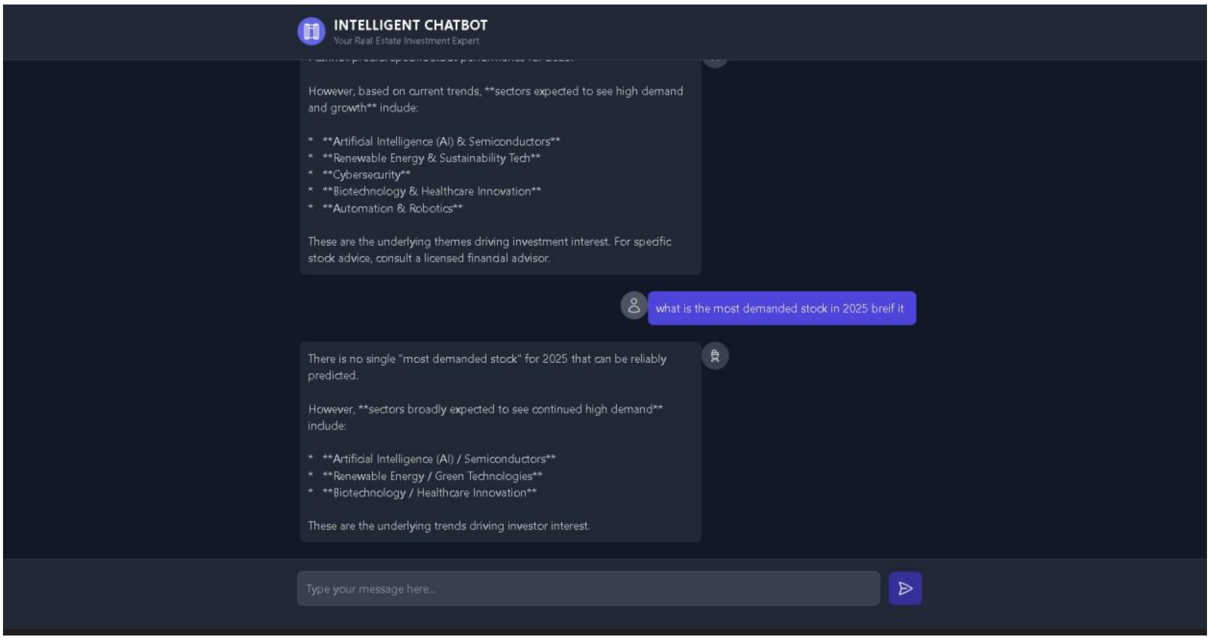
```
1. **Obey Traffic Signals:**
   * **Red Light:** Stop completely before the stop line or crosswalk.
   * **Yellow Light:** Prepare to stop; if you are too close to stop safely, proceed with caution.
   * **Green Light:** Proceed if the intersection is clear.
   * **Arrows:** Follow the direction indicated by the arrow.

2. **Stop at Stop Signs:** Come to a complete stop behind the stop line or crosswalk. Yield to cross-traffic and pedestrians before proceeding.

3. **Yield to the Right (at 4-Way Stops):** If two vehicles arrive at a 4-way stop at the same time, the vehicle on the right has the right of way. Generally, the first vehicle to arrive and stop has the right of way.

4. **Yield to Pedestrians:** Always yield to pedestrians in crosswalks or when they are otherwise legally crossing the street.

5. **Maintain Safe Speed:**
   * Adhere to posted speed limits.
   * Adjust your speed according to road, weather, and traffic conditions (e.g., slow down in rain, fog, or heavy traffic).

6. **Use Turn Signals:** Always signal your intentions before turning, changing lanes, or merging.

7. **Maintain Safe Following Distance:** Keep enough space between your vehicle and the vehicle ahead to react and stop safely (e.g., the "3-second rule").

8. **Stay in Your Lane:** Drive within a single lane and only change lanes when it is safe to do so. On multi-lane highways, generally keep right except when passing.
```

C. CHATBOT CHAT ASSISTANT WEBSITE

CODE:

```
import openai

import gradio

openai.api_key = "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"

messages = [{"role": "system", "content": "You are a financial experts that specializes in real estate investment and negotiation"}]

def CustomChatGPT(user_input):

    messages.append({"role": "user", "content": user_input})

    response = openai.ChatCompletion.create(

        model = "gpt-3.5-turbo",

        messages = messages

    )

    ChatGPT_reply = response["choices"][0]["message"]["content"]

    messages.append({"role": "assistant", "content": ChatGPT_reply})

    return ChatGPT_reply

demo = gradio.Interface(fn=CustomChatGPT, inputs = "text", outputs = "text", title = "INTELLIGENT CHATBOT")

demo.launch(share=True)
```

OUTPUT:



However, based on current trends, **sectors expected to see high demand and growth** include:

*   **Artificial Intelligence (AI) & Semiconductors**
*   **Renewable Energy & Sustainability Tech**
*   **Cybersecurity**
*   **Biotechnology & Healthcare Innovation**
*   **Automation & Robotics**

These are the underlying themes driving investment interest. For specific stock advice, consult a licensed financial advisor.

what is the most demanded stock in 2025 breif it

There is no single "most demanded stock" for 2025 that can be reliably predicted.

However, **sectors broadly expected to see continued high demand** include:

*   **Artificial Intelligence (AI) / Semiconductors**
*   **Renewable Energy / Green Technologies**
*   **Biotechnology / Healthcare Innovation**

These are the underlying trends driving investor interest.

RESULT:

Thus, to build an intelligent chatbox system with Python and dialogue flow was successfully completed and output was verified.