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.

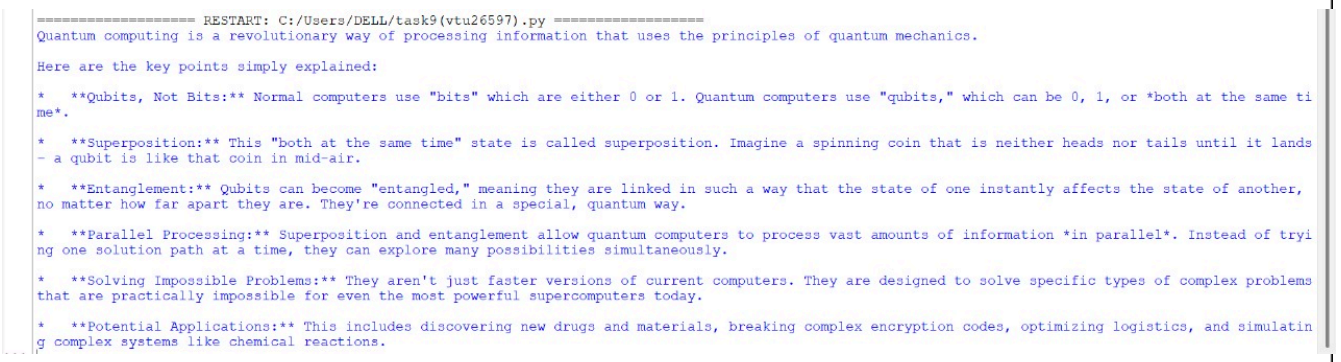
7. Test your chatbot frequently, and use user feedback to improve its performance and provide the best possible experience for your users.

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:



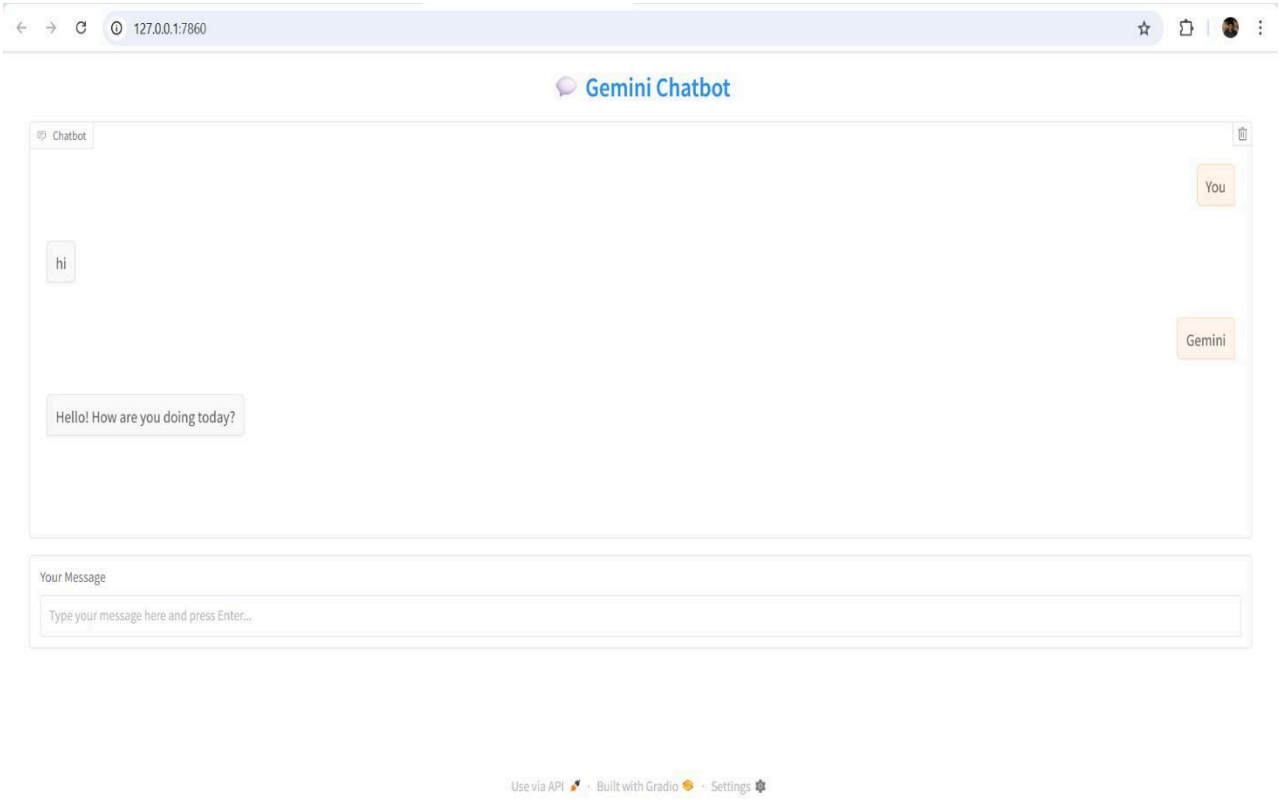
```
========== RESTART: C:/Users/DELL/task9(vtu26597).py ==========
Quantum computing is a revolutionary way of processing information that uses the principles of quantum mechanics.

Here are the key points simply explained:

*   **Qubits, Not Bits:** Normal computers use "bits" which are either 0 or 1. Quantum computers use "qubits," which can be 0, 1, or *both at the same time*.

*   **Superposition:** This "both at the same time" state is called superposition. Imagine a spinning coin that is neither heads nor tails until it lands - a qubit is like that coin in mid-air.

*   **Entanglement:** Qubits can become "entangled," meaning they are linked in such a way that the state of one instantly affects the state of another, no matter how far apart they are. They're connected in a special, quantum way.

*   **Parallel Processing:** Superposition and entanglement allow quantum computers to process vast amounts of information *in parallel*. Instead of trying one solution path at a time, they can explore many possibilities simultaneously.

*   **Solving Impossible Problems:** They aren't just faster versions of current computers. They are designed to solve specific types of complex problems that are practically impossible for even the most powerful supercomputers today.

*   **Potential Applications:** This includes discovering new drugs and materials, breaking complex encryption codes, optimizing logistics, and simulating complex systems like chemical reactions.
```

B. CHATGPT ASSISTANT USING GEMINI

CODE:

```
import gradio as gr from
google import genai
from google.genai import types

GEMINI_API_KEY = "AIzaSyAlA_xxrr5_YXyrp2hIQPPtCyUT4-QMSOs"
client = genai.Client(api_key=GEMINI_API_KEY)
MODEL = "gemini-2.5-flash"

def chat_with_gemini(user_message):
   response = client.models.generate_content(
      model=MODEL,        contents=user_message,
config=types.GenerateContentConfig(
thinking_config=types.ThinkingConfig(thinking_budget=0)
      )
   )
   return response.text

iface = gr.Interface(     fn=chat_with_gemini,
inputs="text",     outputs="text",     title="Gemini Chatbot",
description="Chat with Gemini AI directly from Python"
)

iface.launch()
```

OUTPUT:


127.0.0.1:7860
Gemini Chatbot
Chatbot
You
hi
Gemini
Hello! How are you doing today?
Your Message
Type your message here and press Enter...
Use via API
Built with Gradio
Settings

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:

RESULT:

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