

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.

- 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

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.

○ 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:

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
2025-09-08 00:00:17.602718117.013875 4764 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted ALTS is not enabled.

Here's a simple, brief explanation of quantum computing in points:

**New Type of Computer:** It's a fundamentally different kind of computer that uses the strange rules of quantum mechanics (how tiny particles behave) to process information.

**Qubits (Quantum Bits):** Unlike regular computer bits (which are either a 0 or a 1), a qubit can be both a 0 and a 1 *at the same time*.

**Superposition:** This is the 'both 0 and 1 at once' ability. It means a quantum computer can explore many possibilities simultaneously, like checking all paths in a maze at once.

**Entanglement:** Qubits can be linked in a way that the state of one instantly affects the others, even if they're far apart. This allows for incredibly complex and powerful calculations.

**Parallel Processing:** These properties allow quantum computers to process vast amounts of information and explore many solutions *at the same time*, rather than one by one.

**Solving Complex Problems:** It's designed to tackle problems practically impossible for even the most powerful 'classical' supercomputers, like designing new drugs, breaking encrypted encryption, or optimizing complex systems.
```

○ 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:



Gemini Chatbot

Powered by Google Gemini AI

Hello! I'm your Gemini AI assistant. How can I help you today?

Hi there! How can I assist you today?

Hi there! How can I help you today?

tell me some joke

Okay, here are a few to make you smile! 1. Why don't scientists trust atoms? * Because they make up everything! 2. What do you call a fake noodle? * An impasta! 3. Why did the scarecrow win an award? * Because he was outstanding in his field! 4. Knock, knock. * Who's there? * Lettuce. * Lettuce who? * Lettuce in, it's cold out here! Hope those gave you a chuckle! Want another one?

Type your message here...

Send

○ 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
```

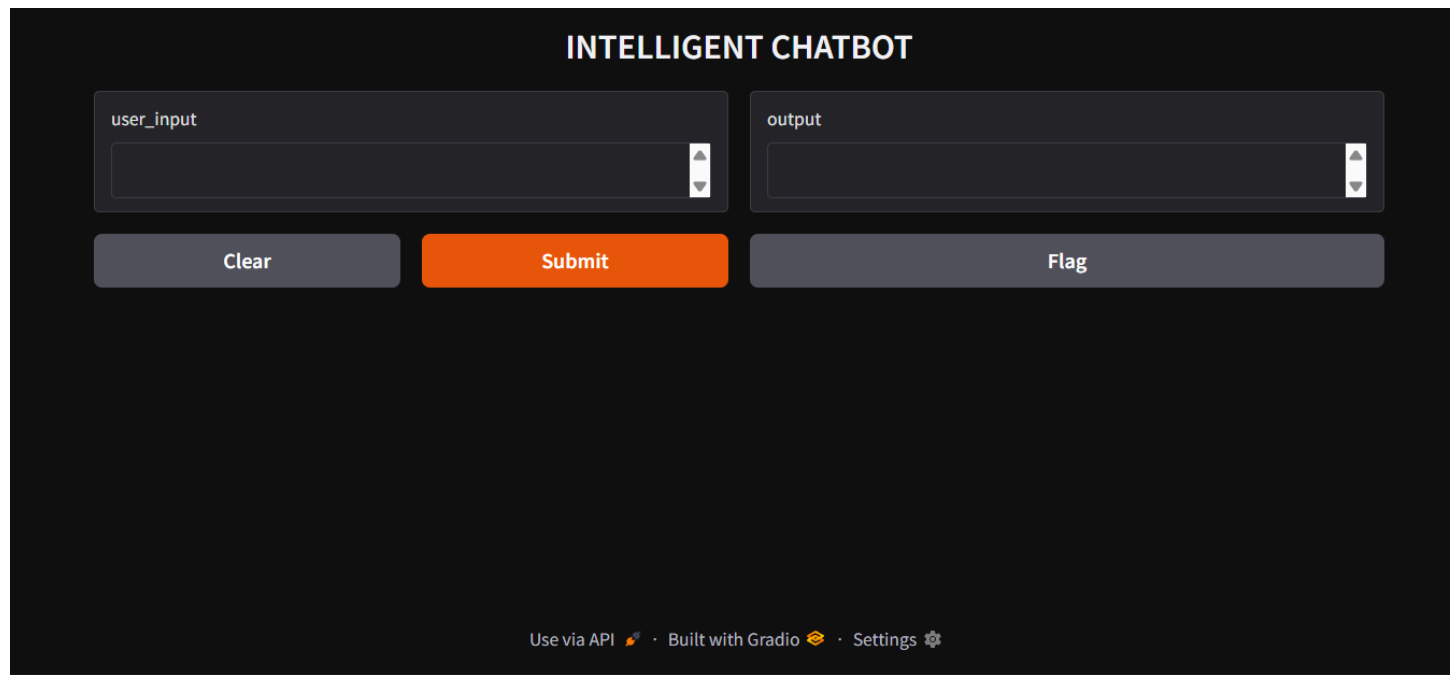
```
)
```

```
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:



The screenshot shows a web application titled "INTELLIGENT CHATBOT" with a dark theme. It features two text input fields: "user_input" on the left and "output" on the right. Below the "user_input" field are three buttons: "Clear", "Submit" (highlighted in orange), and "Flag". The "output" field is currently empty. At the bottom of the interface, there is a footer with the text "Use via API" followed by a lightning bolt icon, "Built with Gradio" followed by the Gradio logo, and "Settings" followed by a gear icon.

RESULT:

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

