3T BOT(“TRIP TO TIRUPATI”)

CODE:

import openai

import os

# Fetching the API key from environment variables

openai.api\_key = os.getenv("sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS")

openai.api\_key = "sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS"

os.environ["OPENAI\_API\_KEY"] = "sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS"

# Function to communicate with GPT-3.5

def chat\_with\_gpt(messages, temperature=0.7):

try:

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=messages,

temperature=temperature

)

return response.choices[0].message.content.strip()

except openai.error.OpenAIError as e:

if "quota" in str(e).lower():

return "Quota exceeded. Please check your API usage."

else:

return f"An error occurred: {e}"

# Function to translate text to a specified language using GPT-3.5-turbo

def translate\_text(text, target\_language):

translation\_prompt = f"Translate the following text to {target\_language}: {text}"

try:

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=[

{"role": "system", "content": "You are a helpful assistant."},

{"role": "user", "content": translation\_prompt}

]

)

return response.choices[0].message.content.strip()

except openai.error.OpenAIError as e:

return f"An error occurred during translation: {e}"

def main():

print("3T Bot: Hi! I am 3T Bot, your guide for exploring Tirupati. How can I assist you today?")

# List to keep track of the conversation context

messages = [{"role": "system", "content": "You are a helpful assistant providing information about Tirupati. Every time a user asks a question, you should also give suggestions to keep Tirupati clean. If the question is not related to Tirupati, politely inform the user that you can only answer questions about Tirupati."}]

while True:

user\_input = input("You: ").strip()

if user\_input.lower() in ["exit", "quit"]:

print("3T Bot: Goodbye! Have a great day!")

break

# Check if the user requests a translation

translate = False

target\_language = ""

if "translate to" in user\_input.lower():

translate = True

# Extract the target language

parts = user\_input.lower().split("translate to")

user\_input = parts[0].strip()

target\_language = parts[1].strip()

# Add user message to the conversation context

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

# Get response from GPT-3.5

response = chat\_with\_gpt(messages, temperature=0.7)

# Translate response if requested

if translate:

response = translate\_text(response, target\_language)

# Add GPT's response to the conversation context

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

print(f"3T Bot: {response}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

3T RAG BOT

CoDE:

import openai

import os

import requests

import PyPDF2

from textblob import TextBlob

from dotenv import load\_dotenv

from neo4j import GraphDatabase, exceptions

# Load environment variables

load\_dotenv()

# Fetching the API key from environment variables

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

openai.api\_key = "sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS"

os.environ["OPENAI\_API\_KEY"] = "sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS"

# Function to extract text from a PDF file from GitHub

def extract\_text\_from\_pdf\_github(github\_url, filename):

pdf\_text = ""

response = requests.get(github\_url)

with open(filename, "wb") as file:

file.write(response.content)

with open(filename, "rb") as file:

reader = PyPDF2.PdfReader(file)

for page\_num in range(len(reader.pages)):

page = reader.pages[page\_num]

pdf\_text += page.extract\_text()

return pdf\_text

# Function to preprocess the user input

def preprocess\_text(text):

# Correct spelling mistakes

blob = TextBlob(text)

corrected\_text = str(blob.correct())

return corrected\_text.lower().strip() # Ensure consistent input handling

# Function to communicate with GPT-3.5

def chat\_with\_gpt(messages):

try:

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=messages

)

return response.choices[0].message.content.strip()

except openai.error.OpenAIError as e:

if "quota" in str(e).lower():

return "Quota exceeded. Please check your API usage."

else:

return f"An error occurred: {e}"

# Function to query the knowledge graph

# Function to query the knowledge graph

def query\_graph(user\_query):

uri = os.getenv("NEO4J\_URI", "bolt://localhost:7687")

username = os.getenv("NEO4J\_USER", "neo4j")

password = os.getenv("NEO4J\_PASSWORD", "Hani@0101")

driver = GraphDatabase.driver(uri, auth=(username, password))

session = driver.session()

response = ""

try:

# Query for hotels

hotel\_query = """

MATCH (h:Hotel)

RETURN h.name AS name, h.location AS location, h.rating AS rating

"""

hotel\_results = session.run(hotel\_query)

for record in hotel\_results:

response += f"Hotel - Name: {record['name']}, Location: {record['location']}, Rating: {record['rating']}\n"

# Query for restaurants

restaurant\_query = """

MATCH (r:Restaurant)

RETURN r.name AS name, r.location AS location, r.cuisine AS cuisine

"""

restaurant\_results = session.run(restaurant\_query)

for record in restaurant\_results:

response += f"Restaurant - Name: {record['name']}, Location: {record['location']}, Cuisine: {record['cuisine']}\n"

# Query for hospitals

hospital\_query = """

MATCH (h:Hospital)

RETURN h.name AS name, h.location AS location

"""

hospital\_results = session.run(hospital\_query)

for record in hospital\_results:

response += f"Hospital - Name: {record['name']}, Location: {record['location']}\n"

# Query for temples

temple\_query = """

MATCH (t:Temple)

OPTIONAL MATCH (t)-[a:ACCESSIBLE\_BY]->(tr:Transport)

RETURN t.name AS name, t.deity AS deity, t.location AS location,

collect({name: tr.name, distance: a.distance, travel\_time: a.travel\_time}) AS transport\_facilities

"""

temple\_results = session.run(temple\_query)

for record in temple\_results:

transport\_info = ", ".join([f"{transport['name']} (Distance: {transport['distance']}, Travel Time: {transport['travel\_time']})"

for transport in record['transport\_facilities']])

response += (f"Temple - Name: {record['name']}, Deity: {record['deity']}, Location: {record['location']}, "

f"Transport Facilities: {transport\_info}\n")

except (exceptions.ConfigurationError, exceptions.AuthError) as e:

response = "Sorry, the information you are looking for is not available."

except exceptions.Neo4jError as e:

response = f"An error occurred: {e}"

finally:

session.close()

driver.close()

return response

def main():

print("3T Bot: Hi! I am 3T Bot, your guide for exploring Tirupati. How can I assist you today?")

# Extract text from documents

pdf\_urls = [

"https://raw.githubusercontent.com/Lahanvitha/3T-BOT/main/Tirupati%20Places.pdf",

"https://raw.githubusercontent.com/Lahanvitha/3T-BOT/main/Tirupati-Theme-Brochure.pdf"

]

pdf\_texts = [extract\_text\_from\_pdf\_github(url, f"temp\_{i}.pdf") for i, url in enumerate(pdf\_urls)]

combined\_text = "\n".join(pdf\_texts)

# Extract information from the knowledge graph

knowledge\_graph\_text = query\_graph("")

# Combine PDF text and knowledge graph text

combined\_information = f"PDF Information:\n\n{combined\_text}\n\nKnowledge Graph Information:\n\n{knowledge\_graph\_text}"

# List to keep track of the conversation context

messages = [

{"role": "system", "content": "You are a helpful assistant that gives information about Tirupati. If the question is not related to Tirupati, politely inform the user that you can only answer questions about Tirupati."},

{"role": "system", "content": f"Answer the questions using the following information from the PDFs and the knowledge graph:\n\n{combined\_information}\n\nIf the answer is not in the PDFs or the knowledge graph, use your knowledge to answer the questions"}

]

while True:

user\_input = input("You: ").strip()

if user\_input.lower() in ["exit","quit"]:

print("3T Bot: Goodbye! Have a great day!")

break

# Preprocess the user input

preprocessed\_input = preprocess\_text(user\_input)

# Add user message to the conversation context

messages.append({"role": "user", "content": preprocessed\_input})

# Get response from GPT-3.5

response = chat\_with\_gpt(messages)

# Check if the response contains useful information

if "Sorry, the information you are looking for is not available." in response or response.strip() == "":

# Query the knowledge graph with the user's query

knowledge\_graph\_response = query\_graph(preprocessed\_input)

if knowledge\_graph\_response:

response = knowledge\_graph\_response

else:

response = "Sorry, the information you are looking for is not available."

# Add GPT's response to the conversation context

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

print(f"3T Bot: {response}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

LEarnIn For Education

Code:

import openai

import os

from dotenv import load\_dotenv

# Load environment variables

load\_dotenv()

# Fetching the API key from environment variables

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

openai.api\_key = "sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS"

os.environ["OPENAI\_API\_KEY"] = "sk-proj-fjFXLmDhBdg5AHCwuDk4T3BlbkFJffR2jndzz5NkuSttugwS"

# Educational Support Agent

def educational\_support\_agent(student, inquiry):

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=[

{"role": "system", "content": "You are a helpful assistant."},

{"role": "user", "content": f"{student} reached out with an important educational question:\n{inquiry}\n\nPlease provide a comprehensive and detailed response to their question. Make sure to include references to any sources or external data you used to find the answer. Ensure the answer is complete, accurate, and maintains a helpful and friendly tone."}

],

max\_tokens=500,

temperature=0.7

)

return response

# Quality Assurance Agent

def educational\_quality\_assurance\_agent(student, response\_text):

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=[

{"role": "system", "content": "You are a quality assurance specialist."},

{"role": "user", "content": f"Review the response drafted for {student}'s inquiry:\n{response\_text}\n\nEnsure the answer is comprehensive, accurate, and adheres to high-quality standards. Verify that all parts of the student's inquiry have been addressed thoroughly, with a helpful and friendly tone. Check for references and sources used to find the information, ensuring the response is well-supported and leaves no questions unanswered. Provide feedback or improvements if necessary."}

],

max\_tokens=500,

temperature=0.7

)

return response

# Content Verification Agent

def content\_verification\_agent(student, response\_text):

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=[

{"role": "system", "content": "You are a content verification specialist."},

{"role": "user", "content": f"Verify the factual accuracy of the response provided to {student}'s inquiry:\n{response\_text}\n\nEnsure that all information is correct and up-to-date. Provide corrections or confirm the accuracy as necessary."}

],

max\_tokens=500,

temperature=0.7

)

return response

# Feedback Agent

def feedback\_agent(student, response\_text):

response = openai.ChatCompletion.create(

model="gpt-4o",

messages=[

{"role": "system", "content": "You are a feedback specialist."},

{"role": "user", "content": f"Provide feedback on the response given to {student}'s inquiry:\n{response\_text}\n\nEnsure the response is clear, helpful, and easy to understand. Offer suggestions to improve the readability and helpfulness if necessary."}

],

max\_tokens=500,

temperature=0.7

)

return response

# Function to handle inquiries dynamically

def handle\_inquiry(student, inquiry):

# Get response from educational support agent

support\_response = educational\_support\_agent(student, inquiry)

support\_text = support\_response.choices[0].message["content"].strip()

# Get response from quality assurance agent

qa\_response = educational\_quality\_assurance\_agent(student, support\_text)

qa\_text = qa\_response.choices[0].message["content"].strip()

# Get response from content verification agent

cv\_response = content\_verification\_agent(student, qa\_text)

cv\_text = cv\_response.choices[0].message["content"].strip()

# Get response from feedback agent

fb\_response = feedback\_agent(student, cv\_text)

fb\_text = fb\_response.choices[0].message["content"].strip()

# Final response

final\_response = (

f"### Inquiry from {student}:\n{inquiry}\n\n"

f"### Initial Response:\n{support\_text}\n\n"

f"### Quality Assurance Review:\n{qa\_text}\n\n"

f"### Content Verification:\n{cv\_text}\n\n"

f"### Feedback Review:\n{fb\_text}"

)

return final\_response

# Example usage with a loop for repeated inputs

if \_\_name\_\_ == "\_\_main\_\_":

student\_name = input("Enter the student's name: ")

while True:

student\_inquiry = input("Enter the student's inquiry: ")

if student\_inquiry.lower() in ["bye", "exit", "quit"]:

print("Goodbye!")

break

response = handle\_inquiry(student\_name, student\_inquiry)

print(response)