DSO 599 - Project 2

Md Farhan Ishmam Sikder, Henry Qiu, Mathis Girard-Soppet

To Be Noted: *Our Code File is attached separately. Can also be found here . We used this to turn pdf into text.*

Task 1- GenAI Application Development - Create a Streamlit Chatbot (15 points)

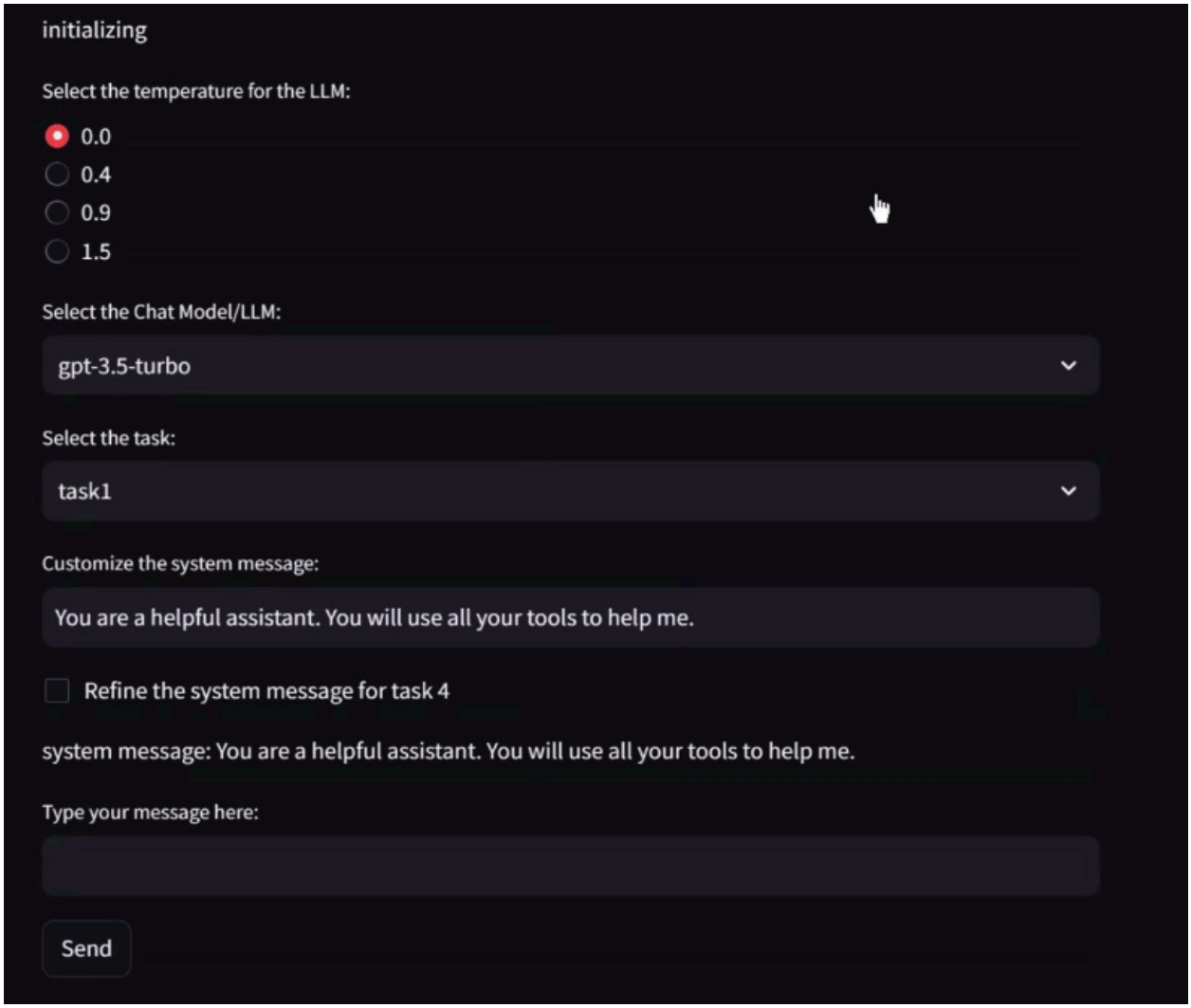
Task Instructions

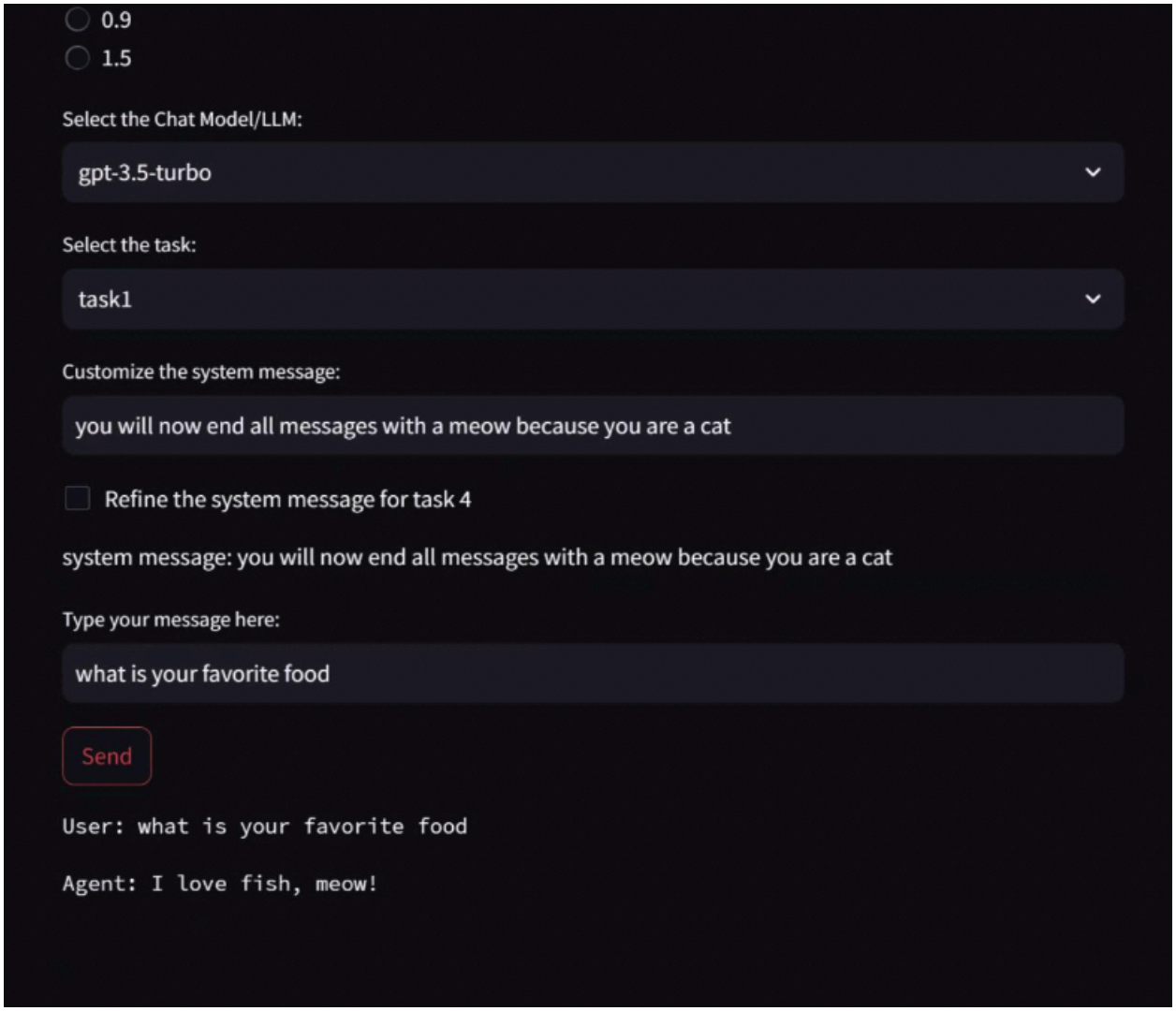
Write a Python program to create Streamlit chatbot that has the following functionality

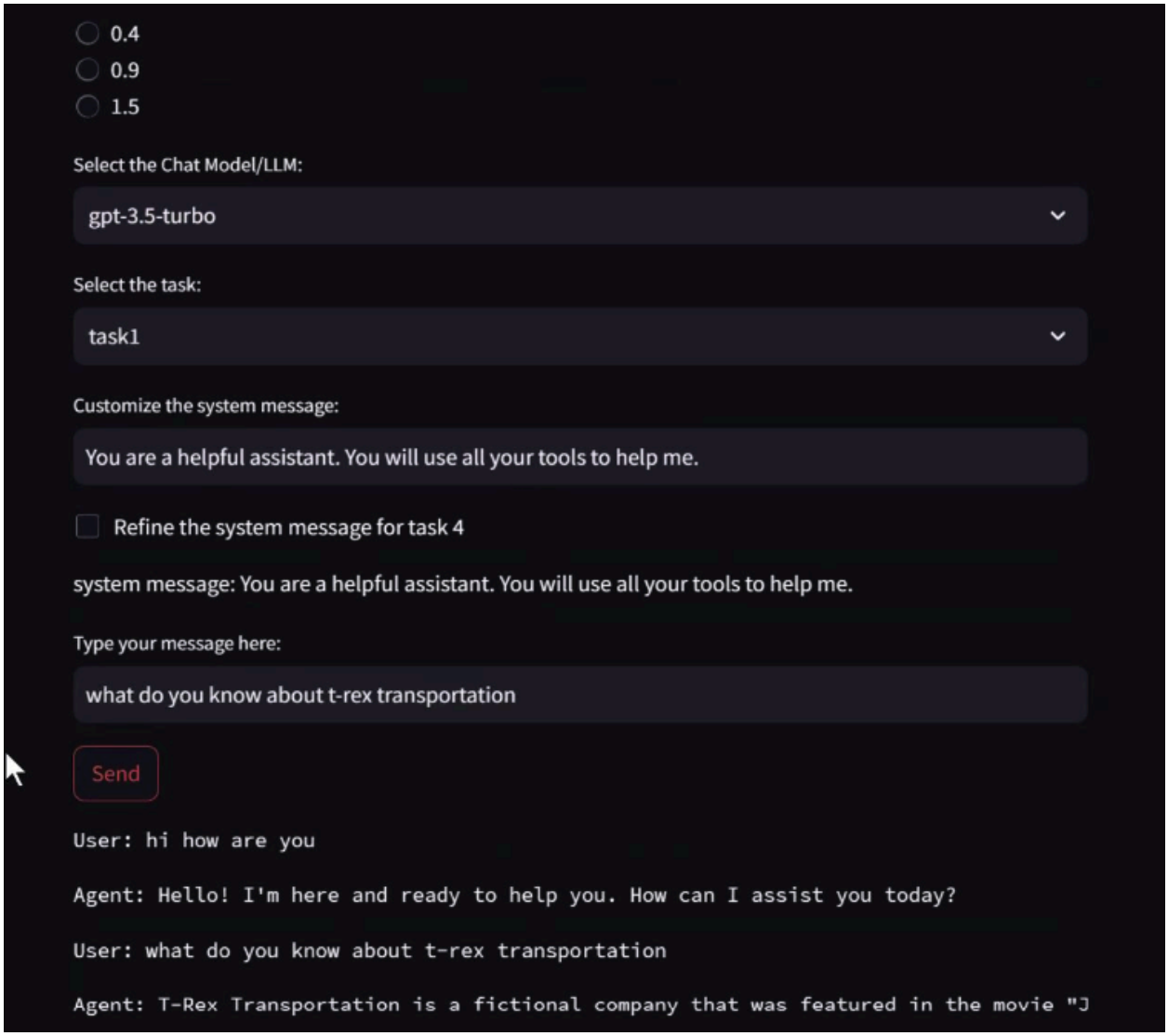
- Input box that accepts user inputs
- Radio buttons to choose one of the three temperature values as an input to your LLM (0, .4, .9)
- Select box to choose the ChatModel/LLM
- Bot should display at least 3 last messages from the user
- A way for the users to change the System Message

Answer

Our chatbot on streamlit




0.9
1.5
Select the Chat Model/LLM:
gpt-3.5-turbo
Select the task:
task1
Customize the system message:
you will now end all messages with a meow because you are a cat
Refine the system message for task 4
system message: you will now end all messages with a meow because you are a cat
Type your message here:
what is your favorite food
Send
User: what is your favorite food
Agent: I love fish, meow!


0.4
0.9
1.5
Select the Chat Model/LLM:
gpt-3.5-turbo
Select the task:
task1
Customize the system message:
You are a helpful assistant. You will use all your tools to help me.
Refine the system message for task 4
system message: You are a helpful assistant. You will use all your tools to help me.
Type your message here:
what do you know about t-rex transportation
Send
User: hi how are you
Agent: Hello! I'm here and ready to help you. How can I assist you today?
User: what do you know about t-rex transportation
Agent: T-Rex Transportation is a fictional company that was featured in the movie "J

Challenges:

Installation of Libraries: Ensuring all required libraries were correctly installed was a key challenge, especially since the development environment can affect the installation process.

Example: When importing libraries in the terminal, some libraries failed to work in the IDE. It was determined that libraries had to be installed directly within the IDE's terminal, not just the system terminal.

API Key Management: Safeguarding API keys was crucial to maintaining security and integrity within the project.

Example: Implementing environment variables to store API keys helped prevent hard-coding sensitive information into the public codebase.

Observations:

Importance of Documentation: The project underscored the importance of consulting official documentation to understand library functionalities and best practices, rather than solely relying on external tools like ChatGPT for quick answers.

Task 2 – GenAI Application Development – Create A Vector Datastore and use it to retrieve the following: (15 points)

Task Instructions

- Create a Vector datastore with the two documents - TRexSafeTemp.pdf andVelociraptorsSafeTemp.pdf.
- Write a tool to use the Vector Store to retrieve information using a React Agent. Check out this link to learn how to create a retriever tool (create_retriever_tool). Check out this link for creating ReAct agents.
- Demonstrate you can answer the following questions using your application
- What is the safe temperature for T-Rex Transport?
- What safety measures should be taken for the Velociraptor when the temperature is 90 degrees?
- Craft an email to the management giving them a status report on your situation.

Answer

Import libraries and provide required key

```
import torch
import platform
device = "cuda" if torch.cuda.is_available() else "cpu"
SLASH = '\\' if platform.system() == 'Windows' else '/'
from huggingface_hub import hf_hub_download
import re
from transformers import NougatProcessor, VisionEncoderDecoderModel
import torch
from pdf2image import convert_from_path
import getpass
import os
os.environ["OPENAI_API_KEY"] = getpass.getpass("Input your OpenAI API key: ")
os.environ["AWS_ACCESS_KEY_ID"] = getpass.getpass("Input your AWS access key:")
os.environ["AWS_SECRET_ACCESS_KEY"] = getpass.getpass("Input your AWS secret access key:")
```

Prepare Vector Store

Description: The script demonstrates preparing a vector store by loading documents, splitting text into chunks, and embedding these chunks using a model. The embedded text is then stored in a FAISS database to facilitate efficient retrieval.

Prepare vector store

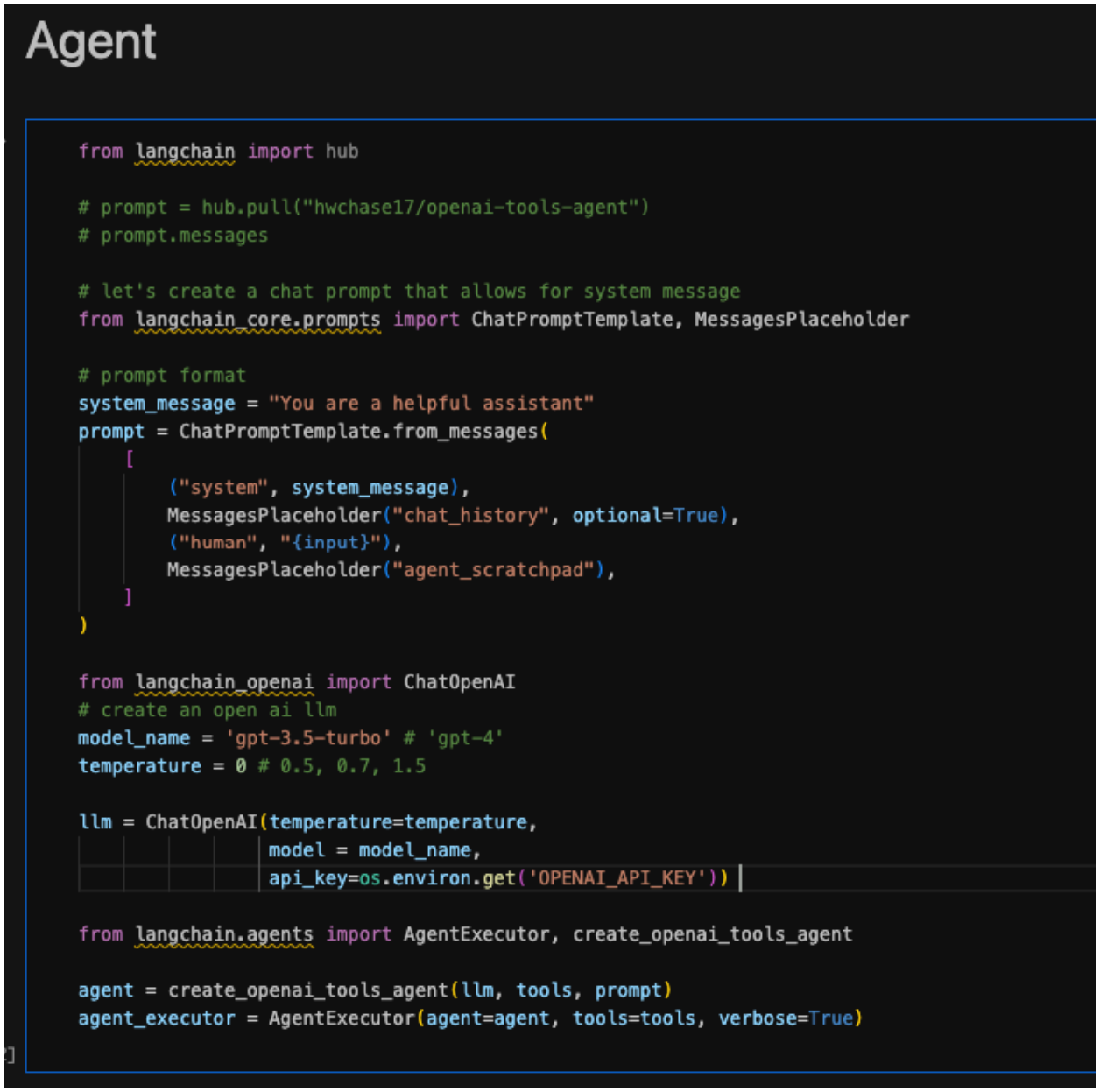
```
from langchain_community.document_loaders import TextLoader
loader = TextLoader('./all_text.txt')
documents = loader.load()

from langchain_community.vectorstores import FAISS
from langchain_openai import OpenAIEmbeddings
# from langchain.embeddings import HuggingFaceEmbeddings # just use a free model from hugging face, should be enough
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter(
    separator="\n",
    chunk_size=1000,
    chunk_overlap=500,
    length_function=len,
)
texts = text_splitter.split_documents(documents) # print this out to show the text being split
print('number of chunks:', len(texts))

# embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
model_for_creating_embeddings = OpenAIEmbeddings(api_key=os.environ.get('OPENAI_API_KEY'))

# initialize a vector database db by applyign the embedding model to the text chuncks
db = FAISS.from_documents(texts, model_for_creating_embeddings)

# create a retriever from the db
retriever = db.as_retriever()

from langchain.tools.retriever import create_retriever_tool

retriever_tool = create_retriever_tool(
    retriever,
    name="retriever",
    description="A retriever tool that to retrieve relevant documents about Dino: T-Rexs and Velociraptors",
)
tools = [retriever_tool]
```

[31]

```
number of chunks: 24
```

Agent Configuration

Description: This code snippet sets up an agent using LangChain to handle interactions based on predefined prompts. It defines the chat prompt, initializes a ChatOpenAI language model, and creates an agent that integrates the retriever tool, ensuring the system can dynamically handle user queries about dinosaur transport safety.



Testing our first query

```
# Example usage:
result = agent_executor.invoke({"input": "hi, im bob"})
result["output"]
```

```
> Entering new AgentExecutor chain...
Hello Bob! How can I assist you today?

> Finished chain.

'Hello Bob! How can I assist you today?'
```

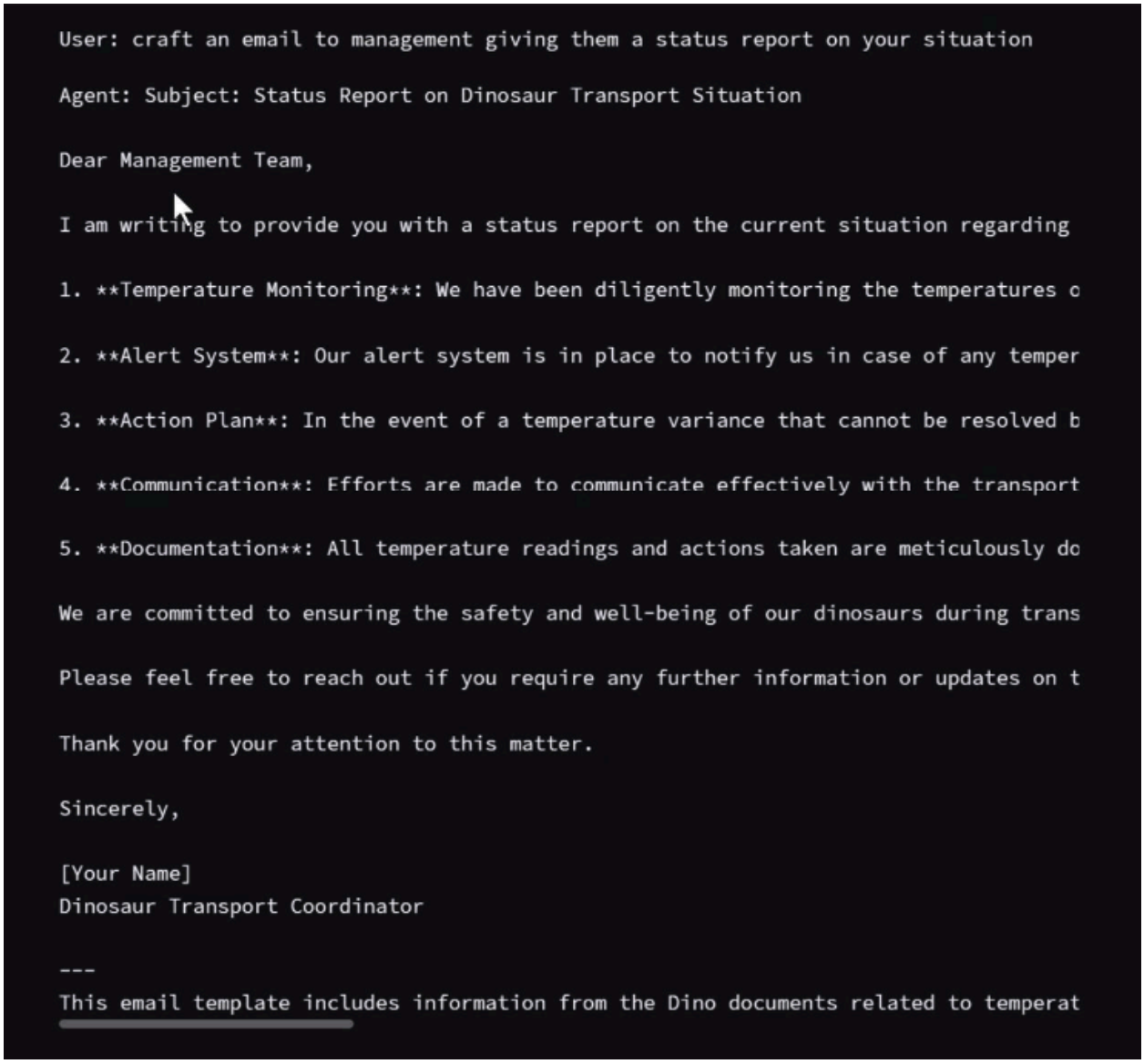
Trying out the query given in the task

```
# Example usage2:
result = agent_executor.invoke(
    {"input": "What is the safe temperature for T-Rex Transport?"}
)
result["output"]
```

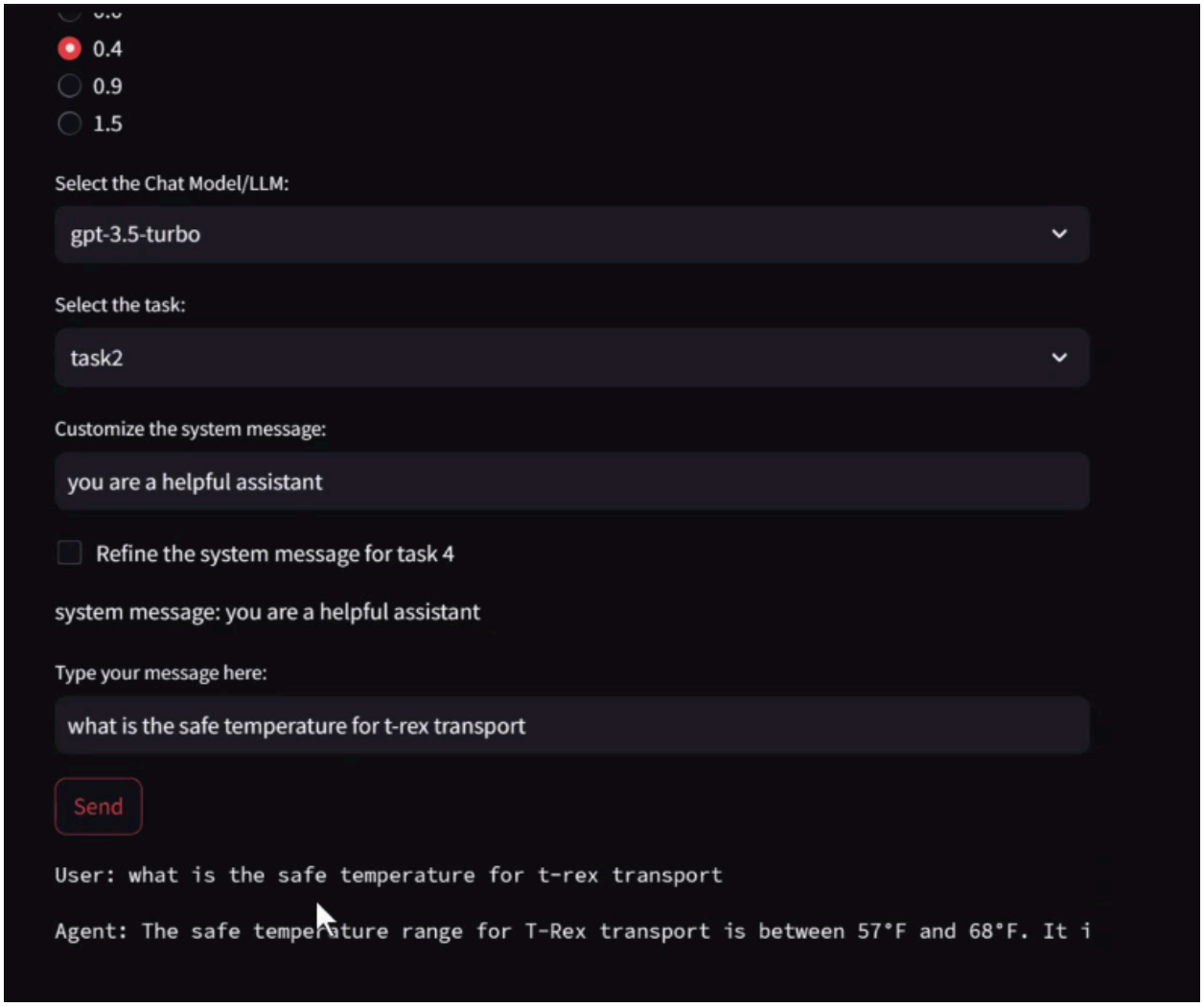
```
> Entering new AgentExecutor chain...

Invoking: `retriever` with `{'query': 'safe temperature for T-Rex transport'}`


T-Rexs :: ## 2 SAFETY
T-Rexs ::
T-Rexs :: There are no direct safety implications implicated by this procedure.
T-Rexs ::
T-Rexs :: ## 3 RESPONSIBILITY
T-Rexs ::
T-Rexs :: \begin{tabular}{|p{142.3pt}|p{142.3pt}|} \hline
T-Rexs :: **Transporters** & Record temperatures from the TYR55 sensor located in the cargo bay of every dinosaur freighter on the Research Dinosaur Welfare Check Log. T-Rexs are not safe when temperatures are out of range (57\({}^{\circ}\)F
T-Rexs :: **Associate Director, Operations and Strategy** & Responsible for the action plan when temperatures in the firefighter cannot be controlled and for Alert Monitoring \\ \hline
T-Rexs :: **Yard Centre Manager (YM)** & Alert monitoring. Responsible for the action plan when the firefighter cannot be controlled. \\ \hline
T-Rexs :: **Corporate Associate Dispatch Manager** & Setting up Alerts and Alert monitoring. \\ \hline

T-Rexs ::
T-Rexs :: ## Purpose
T-Rexs ::
T-Rexs :: This Work Instruction describes the DINO Transporter's responsibility of keeping the T-Rexs safe if the current temperature is out of the safe temperature range, which is 57\({}^{\circ}\)F-68\({}^{\circ}\)F for T-Rexs.
T-Rexs ::
T-Rexs :: SCOPE
...
T-Rexs ::The safe temperature range for T-Rex transport is between 57°F and 68°F. It is important to ensure that the temperature is within this range to keep the T-Rexs safe during transport.

> Finished chain.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

'The safe temperature range for T-Rex transport is between 57°F and 68°F. It is important to ensure that the temperature is within this range to keep the T-Rexs safe during transport.'
```

```
> Entering new AgentExecutor chain...

Invoking: `retriever` with `{'query': 'safety measures for Velociraptor in 90 degrees'}`


Velociraptors ::
Velociraptors :: SCOPE
Velociraptors ::
Velociraptors :: This Work Instruction applies to DINO Operations Staff and teammates in the United States.
Velociraptors ::
Velociraptors :: ## 2 SAFETY
Velociraptors ::
Velociraptors :: There are no direct safety implications implicated by this procedure.
Velociraptors ::
Velociraptors :: ## 3 RESPONSIBILITY
Velociraptors ::
Velociraptors :: \begin{tabular}{|p{142.3pt}|p{142.3pt}|} \hline
Velociraptors :: **Transporters** & Record temperatures from the TYR55 sensor located in the cargo bay of every dinosaur freighter on the Research Dinosaur Welfare Check Log. Velociraptors are not safe when temperatures are out of range (40\({}^{\
Velociraptors :: **Associate Director, Operations and Strategy** & Responsible for the action plan when temperatures in the firefighter cannot be controlled and for Alert Monitoring \\ \hline

==========**JP-ZOO-INST-24: Recording Temperatures for Transport of Velociraptors**==========
Velociraptors ::
Velociraptors :: ## 1 Purpose
...
T-Rexs :: **Yard Centre Manager (YM)** & Alert monitoring. Responsible for the action plan when the firefighter cannot be controlled. \\ \hline
T-Rexs :: **Corporate Associate Dispatch Manager** & Setting up Alerts and Alert monitoring. \\ \hlineWhen the temperature is 90 degrees, it is important to ensure the safety of Velociraptors by keeping them within a safe temperature range. Accord

> Finished chain.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

'When the temperature is 90 degrees, it is important to ensure the safety of Velociraptors by keeping them within a safe temperature range. According to the information retrieved, Velociraptors are not safe when temperatures are out of the range o
```

Finally we have the email:



Here is the version on streamlit:



Challenges:

Hardware Adaptability: Initially planning for CUDA hardware acceleration, the team also ensured the code could fall back to using the CPU if NVIDIA hardware was not available, maintaining functionality across different system setups.

Example: The code was configured to check for the availability of an NVIDIA GPU; if not found, it would default to processing on the CPU, ensuring broader compatibility and usability.
Coherence of Package Versions: Ensuring that all package versions were compatible was vital to prevent conflicts and ensure smooth operation.

Example: During the setup of the vector store, mismatches in library versions led to errors that were resolved by aligning the versions across all dependencies.

Observations:

Utility of Vector Storage: Despite the small size of documents making vector storage seem excessive, this setup provided valuable insights into managing larger datasets for future projects.

Task 3 – GenAI Application Development – Create Tools for an AI Agent to execute tasks (90 points)

Task instructions

- Write a function to create a DynamoDB database, read a csv file and upload the data in the database. The csv file should have these two records. (10 points)

Route_Number	Date	City	DinoID_Transported
123999	3/19/2024	Anchorage	T88
123878	3/20/2024	Seattle	V66

- Write a function to retrieve City and DynoID given a date. Create a tool with proper description and use an LLM to invoke the tool to retrieve DynoID and City for 3/19/2024 (10 points)
- Create a SQL agent that can retrieve a Dino name given a Dino ID. Use the function above to retrieve the Dyno ID and City name given a date and use the SQL Agent to get the Dyno Name from a SQL Database with the DinoMap table (you have to create the table in SQLLite). You can find examples here. (20 points)

ID	Name

T88	T-Rex
V66	Velociraptor

- Now, use a tool to find out the current temperature of a City. Show the output. (10 points)
- Use the LLM to craft a text message with actions taken to keep the Dino safe as the temperature is outside the range. (20 points)
- Write a function (Boto3) to send a text message to a given phone number using AWS SNS. (20 points)

** You can perform these tasks separately. Although an action sequence is implied, I am not asking you to link them together yet.

Answer

AWS Resource Initialization

Description: This code snippet demonstrates the initialization of AWS resources using the Boto3 library. It sets up a connection to Amazon DynamoDB and Amazon Simple Notification Service (SNS), specifying the required AWS region and using environment variables for secure API key management. This setup is crucial for data storage and communication tasks in the application.

```
import boto3
from boto3.dynamodb.conditions import Key
import getpass
from langchain.tools import BaseTool, StructuredTool, tool


dynamodb = boto3.resource('dynamodb',
                          region_name='us-west-1',
                          aws_access_key_id=os.environ.get('AWS_ACCESS_KEY_ID'),
                          aws_secret_access_key=os.environ.get('AWS_SECRET_ACCESS_KEY'),
)

sns = boto3.client('sns',
                   region_name='us-west-1',
                   aws_access_key_id=os.environ.get('AWS_ACCESS_KEY_ID'),
                   aws_secret_access_key=os.environ.get('AWS_SECRET_ACCESS_KEY'),
)
```

DynamoDB Table Management

Description: This script includes functions for managing a DynamoDB table: it can delete, create, and insert data into a table named 'TransportData'. The table creation function sets 'Date' as the primary key and configures the table's read and write capacity. Data insertion uses a batch writer for efficiency, ideal for handling multiple records efficiently.

```
TableName='TransportData'
table = dynamodb.Table(TableName)

def delete_table():
    response = dynamodb.delete_table(
        TableName=TableName,
    )
    return response

try:
    print(delete_table())
except:
    pass

def create_table():
    table = dynamodb.create_table(
        TableName=TableName,
        KeySchema=[
            {'AttributeName': 'Date', 'KeyType': 'HASH'},
        ],
        AttributeDefinitions=[
            {'AttributeName': 'Date', 'AttributeType': 'S'},
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 5,
            'WriteCapacityUnits': 5
        }

    )
    table.wait_until_exists()
    return table

def insert_data(table, data):
    with table.batch_writer() as batch:
        for item in data:
            batch.put_item(Item=item)
```

CSV Data Upload to DynamoDB

Description: This script handles the conversion of CSV data into a list of dictionaries and attempts to create a DynamoDB table. If the table already exists, it captures the exception and proceeds to insert the CSV data into the existing table, demonstrating error handling and data insertion in DynamoDB.

```
# Convert your CSV data to a list of dictionaries
csv_data = [
    {'Date': '3/19/2024', 'Route_Number': 123999, 'City': 'Anchorage', 'DinoID_Transported': 'T88'},
    {'Date': '3/20/2024', 'Route_Number': 123878, 'City': 'Seattle', 'DinoID_Transported': 'V66'},
]


# Creating the table (empty but with format)
try:
    table = create_table()
except dynamodb.meta.client.exceptions.ResourceInUseException:
    table = dynamodb.Table(TableName)
    print('Table already exists')

# Inserting the data (put stuff in it)
insert_data(table, csv_data)
```

```
Table already exists
```

SQL Database Integration and SMS Notification

Description: This script demonstrates several functionalities:
1. It creates a pandas DataFrame from dictionary data, which is then written to an SQLite database using SQLAlchemy, effectively creating a lookup table for Dino IDs and Names.
2. It utilizes a SQLDatabaseToolkit integrated with a ChatOpenAI model to facilitate querying this SQL database via an LLM.
3. Additionally, it showcases how to use AWS SNS to send a text message, providing the functionality needed for alerts or notifications.

```
import pandas as pd
id_name_data = {
    'ID': ['T88', 'V66'],
    'Name': ['T-Rex', 'Velociraptor'],
}
id_name_df = pd.DataFrame(id_name_data)

from langchain_community.utilities import SQLDatabase
from sqlalchemy import create_engine

engine = create_engine("sqlite:///id_name.db")
id_name_df.to_sql("id_name", engine, index=False, if_exists="replace")
db = SQLDatabase(engine=engine)

from langchain_community.agent_toolkits import SQLDatabaseToolkit
from langchain_openai import ChatOpenAI

toolkit = SQLDatabaseToolkit(db=db, llm=ChatOpenAI(temperature=0))
sql_tools = toolkit.get_tools()
```

```
import boto3
sns = boto3.client('sns', region_name='us-east-1')
number = '+16476758044'
sns.publish(PhoneNumber = number, Message='example text message' )
```

```
{'MessageId': '024d6dc1-1501-55b2-b4d9-336c5352831f',
 'ResponseMetadata': {'RequestId': 'fd561eb8-6d92-5feb-9c63-40152bc13946',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'fd561eb8-6d92-5feb-9c63-40152bc13946',
   'date': 'Thu, 02 May 2024 07:43:15 GMT',
   'content-type': 'text/xml',
   'content-length': '294',
   'connection': 'keep-alive'},
  'RetryAttempts': 0}}
```

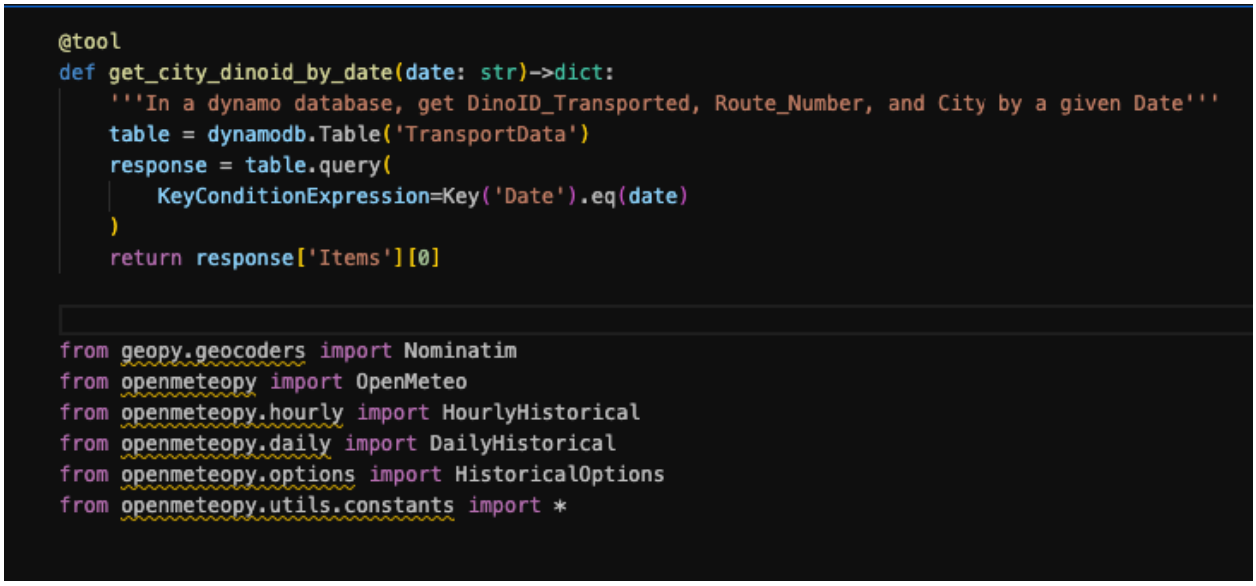
AWS SNS Text Messaging Function

Description: This function defines a method for sending SMS messages using AWS SNS. It initializes an SNS resource, obtains a specific SNS platform application object, and sends a message to a designated phone number with transactional attributes, handling potential errors in the process. This is ideal for ensuring timely communication in applications requiring alerts or notifications.

```python
def send_text_message(phone_number, message):
    """
    Send a text message to the specified phone number using AWS SNS as a resource.

    Args:
    phone_number (str): The phone number to send the message to, including the country code.
    message (str): The message to send.

    Returns:
    dict: The response from the SNS service if successful, otherwise raises an error.
    """
    # Create a new SNS resource
    sns = boto3.resource('sns')

    # Get the SNS platform application object
    platform_application = sns.PlatformApplication('arn:aws:sns:REGION:ACCOUNT_ID:app/SMS/MySMSApplication')

    try:
        # Publish the SMS message
        response = platform_application.publish(
            PhoneNumber=phone_number,
            Message=message,
            MessageAttributes={
                'AWS.SNS.SMS.SMSType': {
                    'DataType': 'String',
                    'StringValue': 'Transactional'  # Can be 'Promotional' depending on the usage
                }
            }
        )
        return response
    except ClientError as e:
        print(f"An error occurred: {e}")
        raise
```

DynamoDB Query and Weather Data Retrieval Integration

Description: This script includes a function that queries a DynamoDB table to retrieve city and DinoID information based on a specified date. Additionally, it incorporates the OpenMeteo API using the openmeteopy library to fetch historical or current weather data for the retrieved city, integrating geographical and meteorological data retrieval into the application's functionality.

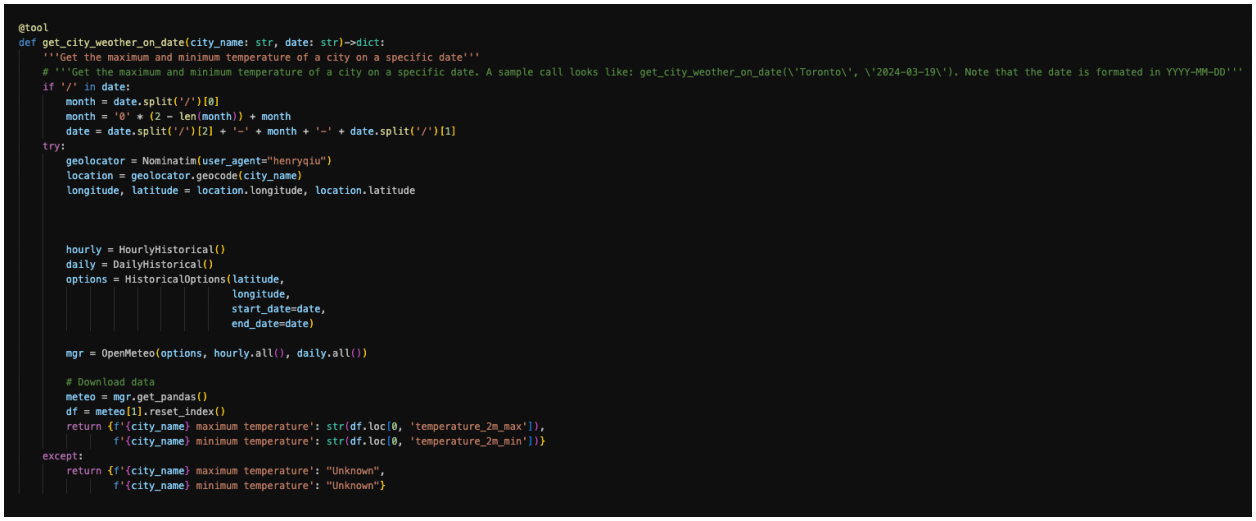


```python
@tool
def get_city_dinoid_by_date(date: str)->dict:
    '''In a dynamo database, get DinoID_Transported, Route_Number, and City by a given Date'''
    table = dynamodb.Table('TransportData')
    response = table.query(
        KeyConditionExpression=Key('Date').eq(date)
    )
    return response['Items'][0]
```

```python
from geopy.geocoders import Nominatim
from openmeteopy import OpenMeteo
from openmeteopy.hourly import HourlyHistorical
from openmeteopy.daily import DailyHistorical
from openmeteopy.options import HistoricalOptions
from openmeteopy.utils.constants import *
```

Historical Weather Data Retrieval Function

Description: This function retrieves the maximum and minimum temperatures for a given city on a specific date. It utilizes geocoding to determine the city's coordinates and then fetches historical weather data using the OpenMeteo API. The function is designed to return the temperature range, ensuring accurate weather data retrieval for any required date.



```python
@tool
def get_city_weother_on_date(city_name: str, date: str)->dict:
    '''Get the maximum and minimum temperature of a city on a specific date'''
    # '''Get the maximum and minimum temperature of a city on a specific date. A sample call looks like: get_city_weother_on_date(\'Toronto\', \'2024-03-19\'). Note that the date is formated in YYYY-MM-DD'''
    if '/' in date:
        month = date.split('/')[0]
        month = '0' * (2 - len(month)) + month
        date = date.split('/')[2] + '-' + month + '-' + date.split('/')[1]
    try:
        geolocator = Nominatim(user_agent="henryqiu")
        location = geolocator.geocode(city_name)
        longitude, latitude = location.longitude, location.latitude



        hourly = HourlyHistorical()
        daily = DailyHistorical()
        options = HistoricalOptions(latitude,
                                    longitude,
                                    start_date=date,
                                    end_date=date)

        mgr = OpenMeteo(options, hourly.all(), daily.all())

        # Download data
        meteo = mgr.get_pandas()
        df = meteo[1].reset_index()
        return {f'{city_name} maximum temperature': str(df.loc[0, 'temperature_2m_max']),
                f'{city_name} minimum temperature': str(df.loc[0, 'temperature_2m_min'])}
    except:
        return {f'{city_name} maximum temperature': "Unknown",
                f'{city_name} minimum temperature': "Unknown"}
```

Enhanced SMS Sending Function with AWS SNS

Description: This function facilitates sending SMS messages via AWS SNS, detailing parameters for phone number and message content. It attempts to publish an SMS and optionally allows for setting message attributes such as sender ID and message type (Transactional or Promotional). The function also includes error handling to manage potential issues during the SMS sending process.

```python
import boto3
from botocore.exceptions import BotoCoreError, ClientError

@tool
def send_sms(phone_number, message):
    """
    Send an SMS message to a specified phone number using AWS SNS.

    Parameters:
    - phone_number (str): The phone number to send the message to. It must be in E.164 format (e.g., +1234567890).
    - message (str): The content of the message to be sent.

    Returns:
    - response (dict): The response from the SNS service, including the message ID if successful.
    - error (str): Error message if the SMS sending fails.
    """
    # Initialize the boto3 SNS client

    try:
        # Send the SMS message
        response = sns.publish(
            PhoneNumber=phone_number,
            Message=message,
            # Uncomment and specify the following attribute if needed
            MessageAttributes={
                'AWS.SNS.SMS.SenderID': {
                    'DataType': 'String',
                    'StringValue': 'Bot'
                },
                'AWS.SNS.SMS.SMSType': {
                    'DataType': 'String',
                    'StringValue': 'Transactional'  # or 'Promotional'
                }
            }
        )
        return response, None
    except (BotoCoreError, ClientError) as e:
        # Handle possible errors
        return None, str(e)
```

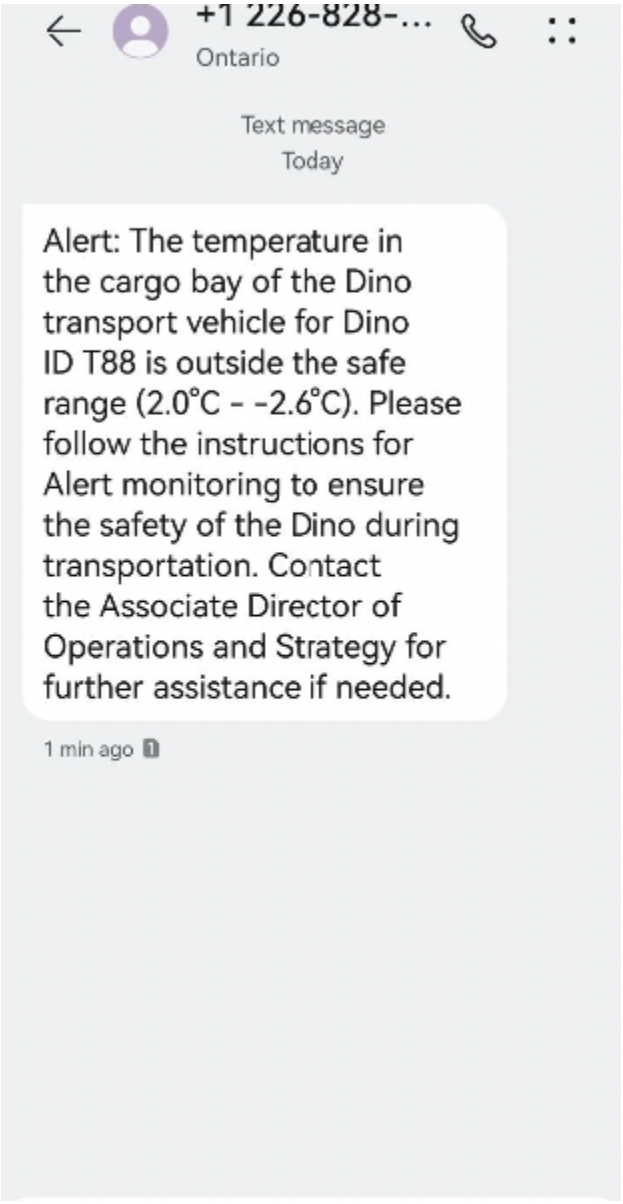
Function Integration and Testing

Description: This example demonstrates the successful integration and testing of the SMS sending function and data retrieval functions. It showcases sending an SMS via AWS SNS and retrieving data by date from DynamoDB, followed by fetching weather data for a specific city and date. The output confirms the functions' efficacy with details of the SMS sent, dinosaur transport data, and temperature information for Anchorage.

```python
# Example usage
phone_number = '+16476758044'  # Replace with a valid phone number
message = "Hello, this is a test message from AWS SNS!"
response, error = send_sms({'phone_number': phone_number, 'message': message})

if response:
    print("Message sent successfully! Message ID:", response['MessageId'])
else:
    print("Failed to send message:", error)

# Retrieving data for a specific date
result = get_city_dinoid_by_date('3/19/2024')
print(result)
# Retrieving weather data for a specific date]
result = get_city_weother_on_date({'city_name': "Anchorage", "date":"2024-03-19"})
print(result)
```

```
Message sent successfully! Message ID: 558a553e-5215-5a85-81eb-0b9318821b3e
{'DinoID_Transported': 'T88', 'Date': '3/19/2024', 'Route_Number': Decimal('123999'), 'City': 'Anchorage'}
{'Anchorage maximum temperature': '2.0', 'Anchorage minimum temperature': '-2.6'}
```

Text message example:



Challenges:

SMS Verification Delays: Issues with SMS verification for a U.S. phone number led to the use of a Canadian number, complicating the initial setup.

Example: The delay in SMS verification for the U.S. number required a quick pivot to a Canadian number to meet project timelines.

Cross-Platform Development: Differences in file path notations and system commands between operating systems like Linux and Windows posed challenges for code compatibility and collaboration.

Example: Code developed on a Linux system used different path separators, causing errors when run on Windows systems. The team had to standardize file paths and commands to ensure cross-platform compatibility.

Observations:

Adapting to External Constraints: The project highlighted the need to be adaptable and responsive to external constraints, such as API limitations and verification processes.

Task 4 – GenAI Application Development – Create a program that executes the end-to-end workflow(80 points)

This is the most fun part of the exercise. You have access to various tools and functions that you have created. Can you create an end-to-end workflow where you start with a date (3/19/2024), find out the Dino that's being transported and the city in which the transportation is happening on that date, find out the temperature of the city, find out if the temperature is safe or not for the Dino in question, if the temperature is not safe, find out the actions that are needed to keep the dino safe, craft a status email to the manager and send the text given a phone number? You can use any combination of LLM calls and functions you want. The idea is to use an AI Agent end-to-end to perform these tasks using these tools but you are free to choose any approach.

Answer

AI Agent Integration for Automated Workflow

Description: This code snippet outlines the setup and utilization of an AI-powered agent using LangChain and OpenAI's tools. It incorporates various tools for retrieving data, evaluating conditions, and generating responses, structured around a conversation template for interaction. The agent setup includes initializing a language model and combining functional tools to enable automated decision-making and response generation based on user inputs and historical data. This facilitates an end-to-end workflow where the AI agent can handle complex queries regarding dinosaur safety based on environmental conditions on a specific date.

```python
from langchain import hub

# prompt = hub.pull("hwchase17/openai-tools-agent")
# prompt.messages

# let's create a chat prompt that allows for system message
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful assistant"),
        MessagesPlaceholder("chat_history", optional=True),
        ("human", "{input}"),
        MessagesPlaceholder("agent_scratchpad"),
    ]
)

from langchain_openai import ChatOpenAI
llm = ChatOpenAI(temperature=0,
                 api_key=os.environ.get('OPENAI_API_KEY'))
from langchain.agents import AgentExecutor, create_openai_tools_agent

tools = [retriever_tool, get_city_dinoid_by_date, get_city_weother_on_date] + sql_tools

agent = create_openai_tools_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
from langchain_core.messages import HumanMessage

chat_history = []
```

```python
agent_executor.invoke(
    {
        "input": "Can you craft a text message with actions taken to keep the Dino safe as the temperature is outside the range on 3/19/2024",
        "chat_history": chat_history,
    }
)
```

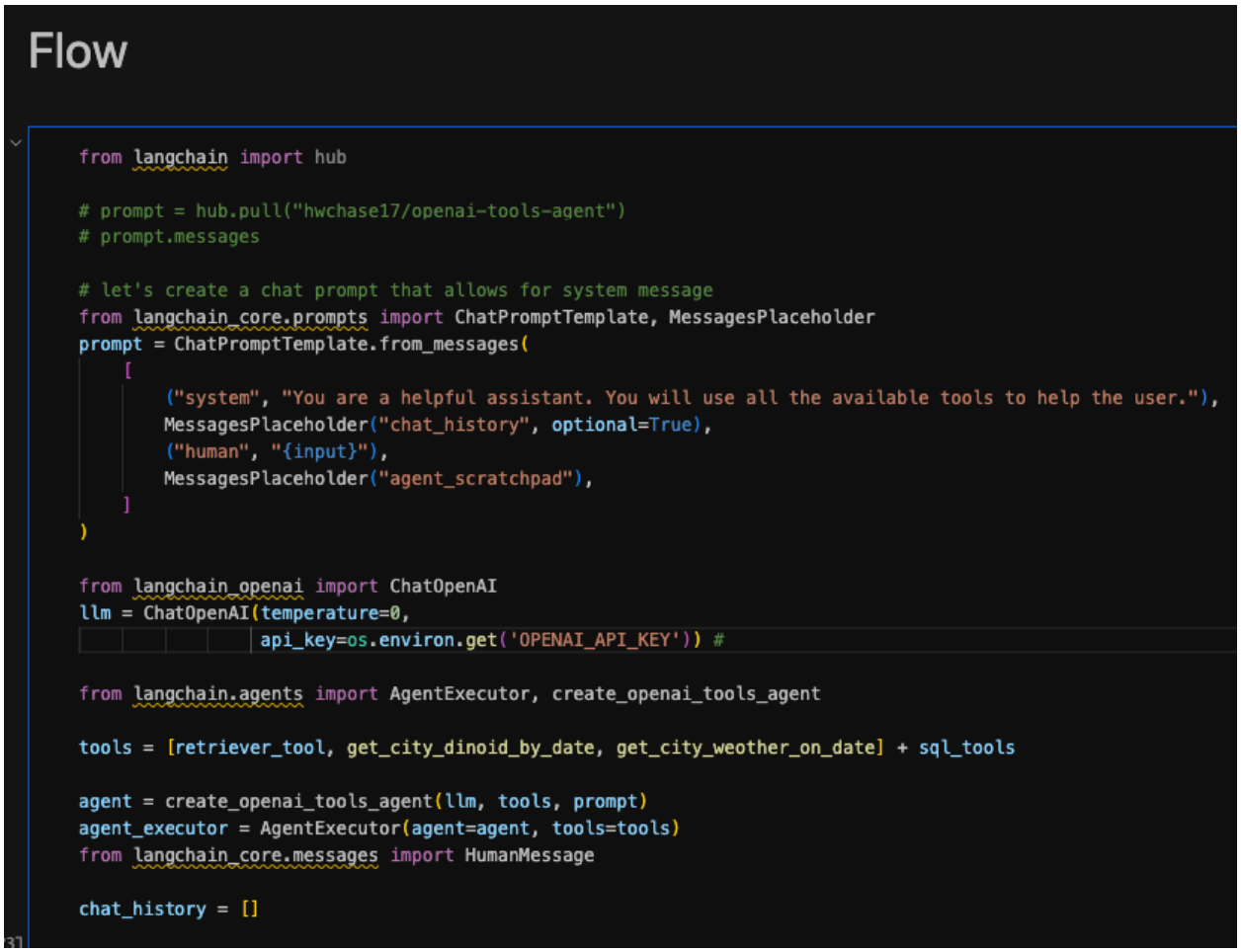
Results

```
> Entering new AgentExecutor chain...

Invoking: `get_city_dinoid_by_date` with `{'date': '3/19/2024'}`


{'DinoID_Transported': 'T88', 'Date': '3/19/2024', 'Route_Number': Decimal('123999'), 'City': 'Anchorage'}
Invoking: `get_city_weother_on_date` with `{'city_name': 'Anchorage', 'date': '3/19/2024'}`


{'Anchorage maximum temperature': '2.0', 'Anchorage minimum temperature': '-2.6'}I have gathered the necessary information. The Dino with ID T88 is in Anchorage on 3/19/2024, where the maximum temperature is 2.0°C and the minimum temperature is -2.6

---

**Text Message:**
"Hello Team,

The Dino with ID T88 is currently in Anchorage on Route Number 123999. The temperature on 3/19/2024 is outside the safe range, with a maximum of 2.0°C and a minimum of -2.6°C. To ensure the Dino's safety, we have taken the following actions:
1. Increased monitoring of the Dino's condition.
2. Provided additional insulation and heating in the transport vehicle.
3. Adjusted the route to minimize exposure to extreme temperatures.

Please continue to monitor the situation closely and take necessary precautions to keep the Dino safe.
...

Feel free to customize the message further as needed.

> Finished chain.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

{'input': 'Can you craft a text message with actions taken to keep the Dino safe as the temperature is outside the range on 3/19/2024',
 'chat_history': [],
 'output': 'I have gathered the necessary information. The Dino with ID T88 is in Anchorage on 3/19/2024, where the maximum temperature is 2.0°C and the minimum temperature is -2.6°C. Here is a crafted text message for actions taken to keep the Dino
```

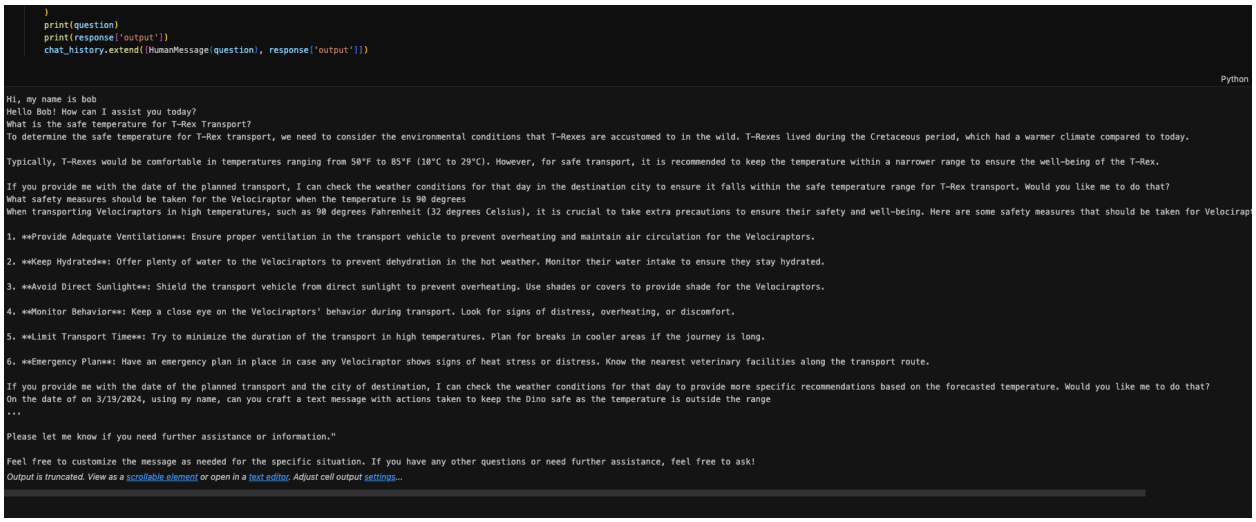
AI Agent Dynamic Interaction Flow

Description: These code snippets demonstrate setting up and using an AI agent to manage interactions through a structured conversational flow. The agent is configured with various tools and prompts to assist users by answering queries related to dinosaur transport, such as safety measures and temperature conditions. The example shows initializing chat history, querying the agent with specific inputs, and updating the chat history based on the agent's responses,

illustrating a dynamic, context-aware interaction system capable of handling complex informational tasks.



Results



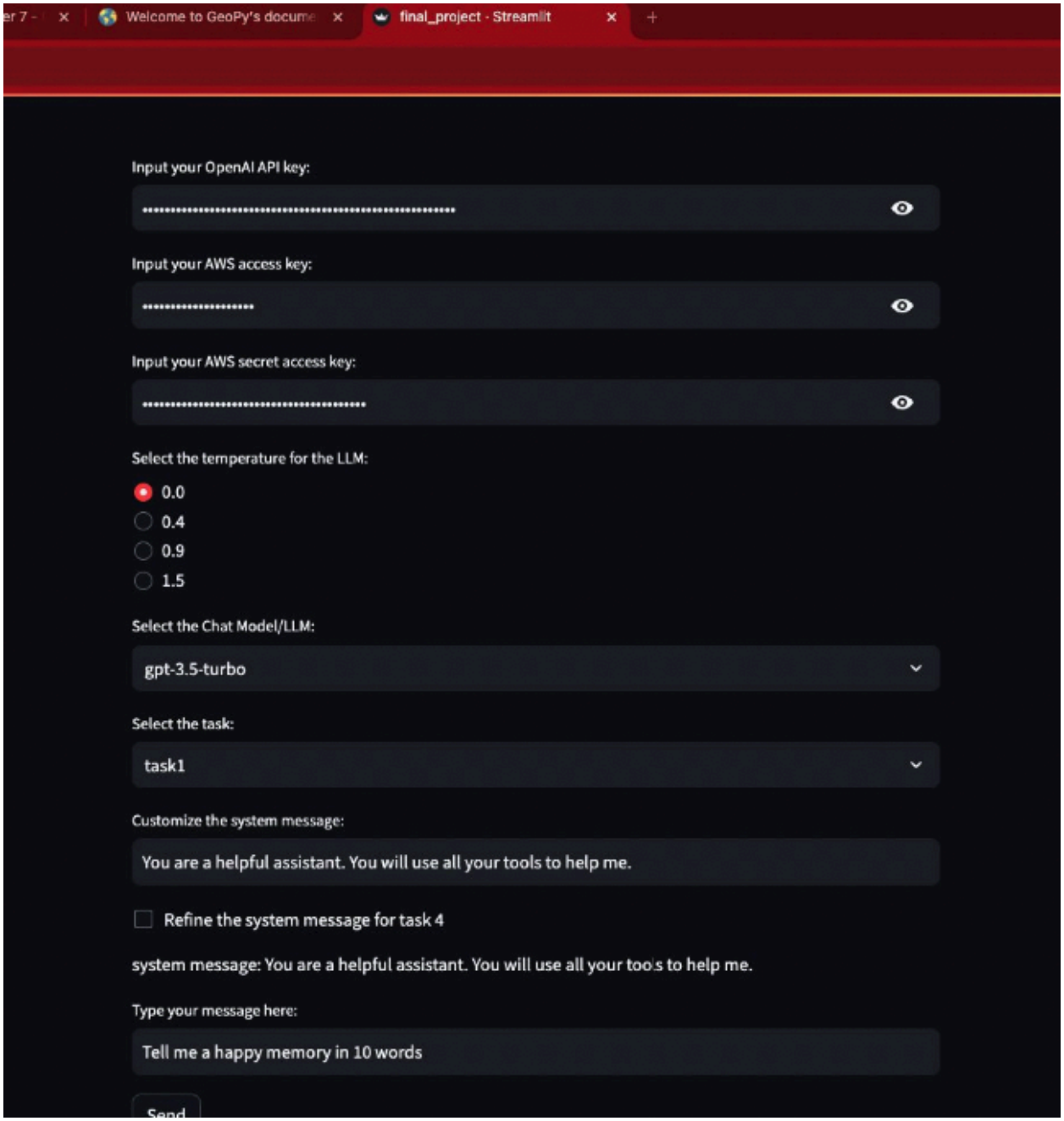
```
    }
    print(question)
    print(response['output'])
    chat_history.extend([HumanMessage(question), response['output']])
```

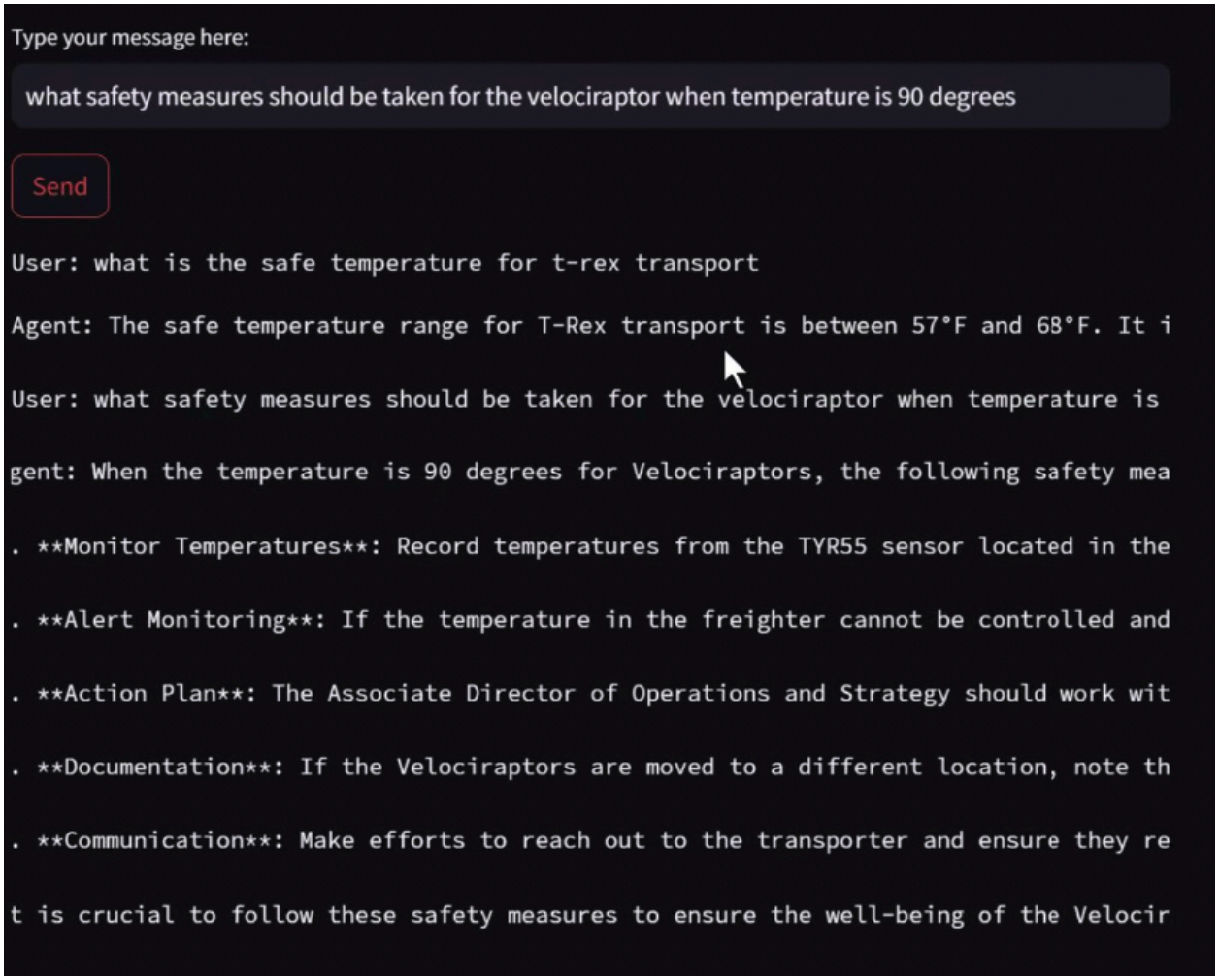
Python

```
Hi, my name is bob
Hello Bob! How can I assist you today?
What is the safe temperature for T-Rex Transport?
To determine the safe temperature for T-Rex transport, we need to consider the environmental conditions that T-Rexes are accustomed to in the wild. T-Rexes lived during the Cretaceous period, which had a warmer climate compared to today.

Typically, T-Rexes would be comfortable in temperatures ranging from 50°F to 85°F (10°C to 29°C). However, for safe transport, it is recommended to keep the temperature within a narrower range to ensure the well-being of the T-Rex.

If you provide me with the date of the planned transport, I can check the weather conditions for that day in the destination city to ensure it falls within the safe temperature range for T-Rex transport. Would you like me to do that?
What safety measures should be taken for the Velociraptor when the temperature is 90 degrees
When transporting Velociraptors in high temperatures, such as 90 degrees Fahrenheit (32 degrees Celsius), it is crucial to take extra precautions to ensure their safety and well-being. Here are some safety measures that should be taken for Velociraptors:

1. **Provide Adequate Ventilation**: Ensure proper ventilation in the transport vehicle to prevent overheating and maintain air circulation for the Velociraptors.

2. **Keep Hydrated**: Offer plenty of water to the Velociraptors to prevent dehydration in the hot weather. Monitor their water intake to ensure they stay hydrated.

3. **Avoid Direct Sunlight**: Shield the transport vehicle from direct sunlight to prevent overheating. Use shades or covers to provide shade for the Velociraptors.

4. **Monitor Behavior**: Keep a close eye on the Velociraptors' behavior during transport. Look for signs of distress, overheating, or discomfort.

5. **Limit Transport Time**: Try to minimize the duration of the transport in high temperatures. Plan for breaks in cooler areas if the journey is long.

6. **Emergency Plan**: Have an emergency plan in place in case any Velociraptor shows signs of heat stress or distress. Know the nearest veterinary facilities along the transport route.

If you provide me with the date of the planned transport and the city of destination, I can check the weather conditions for that day to provide more specific recommendations based on the forecasted temperature. Would you like me to do that?
On the date of on 3/19/2024, using my name, can you craft a text message with actions taken to keep the Dino safe as the temperature is outside the range
'''

Please let me know if you need further assistance or information."

Feel free to customize the message as needed for the specific situation. If you have any other questions or need further assistance, feel free to ask!
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

On Streamlit the final integrated version along with test message:





Challenges:

Complex Workflow Coordination: Integrating multiple tools and data sources into a seamless workflow was complex and highlighted the importance of thorough testing and debugging.

Example: Coordinating the retrieval of dinosaur types, checking for environmental conditions, and triggering SMS notifications required precise control over the flow of data and execution logic.

Observations:

Power of Integrated Systems: The successful automation of complex decision-making processes demonstrated the potential and power of fully integrated AI systems in solving real-world problems.