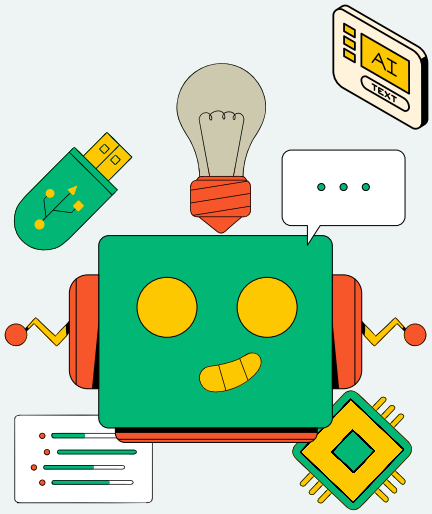
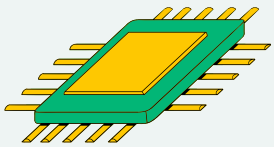
THYNK UNLIMITED
WE LEARN FOR THE FUTURE



CREATING AN LLM DRIVEN CHATBOT

DSO 599: FINAL PRESENTATION

PRESENTED BY:
JESSICA BRATAHANI
JIANAN DING
XIAOYI GUAN



PRESENTATION OUTLINE

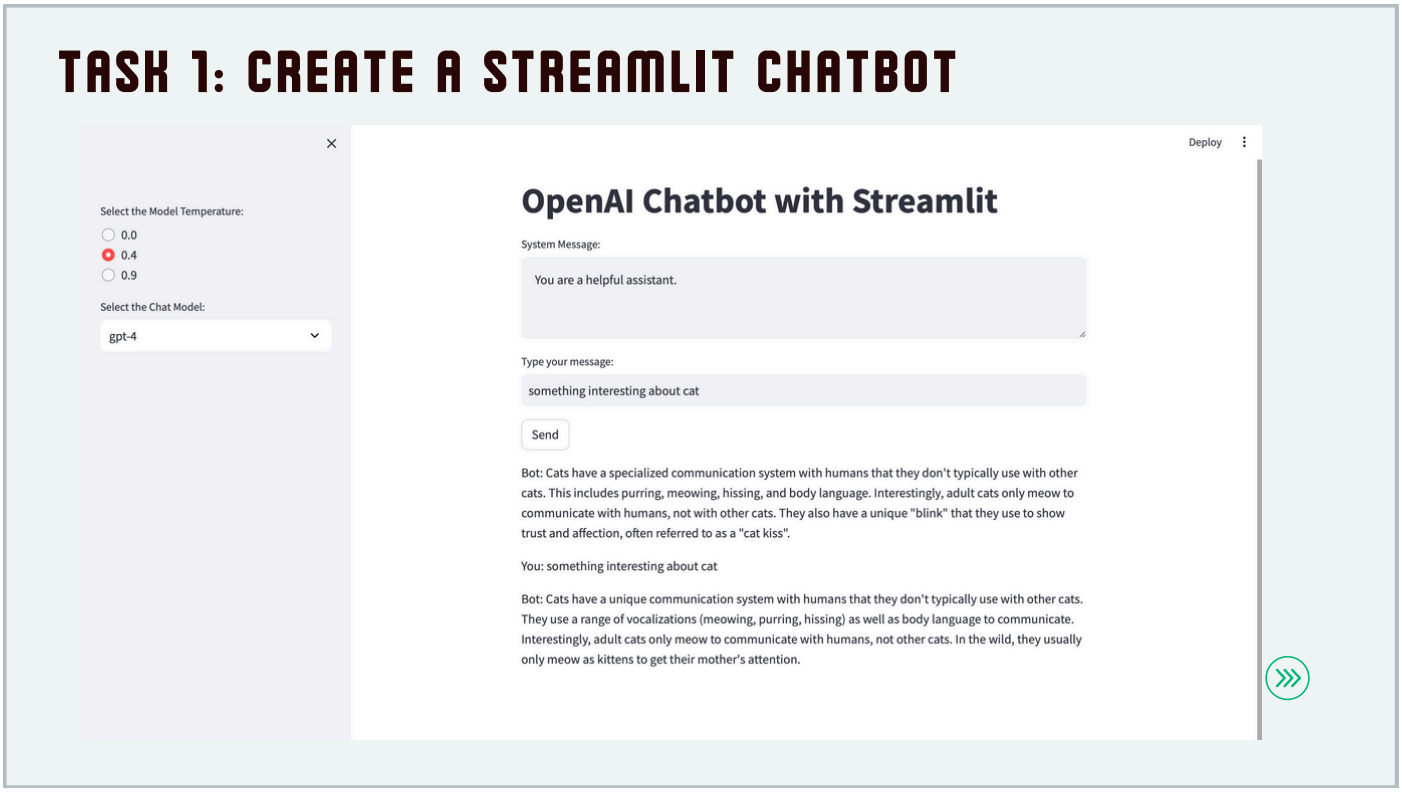
- Introduction
- Task 1 – Streamlit ChatBot
- Task 2 – Vector Datastore
- Task3 – GenAI Application Development tools
- Task 4 – Application Integration
- Output and Evaluation

INTRODUCTION

Explore the capabilities of Language Learning Models (LLMs) in automating tasks and enhancing data interaction through user-friendly interfaces.

FOR THIS PROJECT, WE ARE CREATING AN LLM POWERED CHATBOT FOR USERS TO ACCESS INFORMATION REGARDING DINOSAUR TRANSPORT FROM A TRANSPORTING COMPANY

Throughout our presentation, we will discuss our methodologies, the challenges we faced, and the innovative solutions we implemented. We aim to demonstrate not only what our applications can do but also how they can be integrated into broader systems to enhance operational efficiency and decision-making processes.



Task 1: Create a ChatBot using Streamlit

We utilized OpenAI's models to create a dynamic and interactive user experience. We allow users to customize the system message and adjust how the AI responds by selecting different temperature settings and models.

Challenge faced:
It is important that we need to use the paid version of OpenAI for the API key to make sure we can use it.

TASK 2 - CREATE A VECTOR DATASTORE AND USE IT TO RETRIEVE THE INFORMATION USING A REACT AGENT

```
def initial():
    try:
        # Set up LangChain components
        llm = ChatOpenAI(temperature=0)
        pdf_folder = 'pdfs/'
        loader = PyPDFDirectoryLoader(pdf_folder, recursive=True)
        syllabus_kb = loader.load()
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
        splits = text_splitter.split_documents(syllabus_kb)
        embedding_function = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")
        vector_store = FAISS.from_documents(documents=splits, embedding=embedding_function)
        retriever = vector_store.as_retriever()

        # Create LangChain tool
        tool = create_retriever_tool(retriever, "search_dino_docs", "Searches and returns excerpts from the dino pdfs.")
        tools = [tool]

        # Set up LangChain agent
        prompt = hub.pull("hwchase17/react")
        memory = ConversationBufferMemory(memory_key="chat_history")
        my_agent = create_react_agent(llm, tools, prompt)
        agent_executor = AgentExecutor(agent=my_agent, tools=tools, memory=memory, verbose=True)
        return agent_executor
    except Exception as e:
        st.error(f"An error occurred: {str(e)}")
```

Task 2: Create a Vector Datastore and use it to retrieve information using a React Agent.

Creating a vector datastore is crucial for retrieving information using a React Agent because it enables efficient and context-aware search.

1. Fast and Efficient Search
Traditional databases rely on exact keyword matching, which can be slow and ineffective for semantic searches.
A vector datastore (like Pinecone, Weaviate, FAISS, or ChromaDB) allows nearest neighbor search using vector embeddings, making retrieval fast and highly relevant.

2. Semantic Understanding
A React Agent processes natural language input and needs to understand context rather than just keywords.
By storing text, images, or other data as vectors, the agent can retrieve semantically similar inform ation.

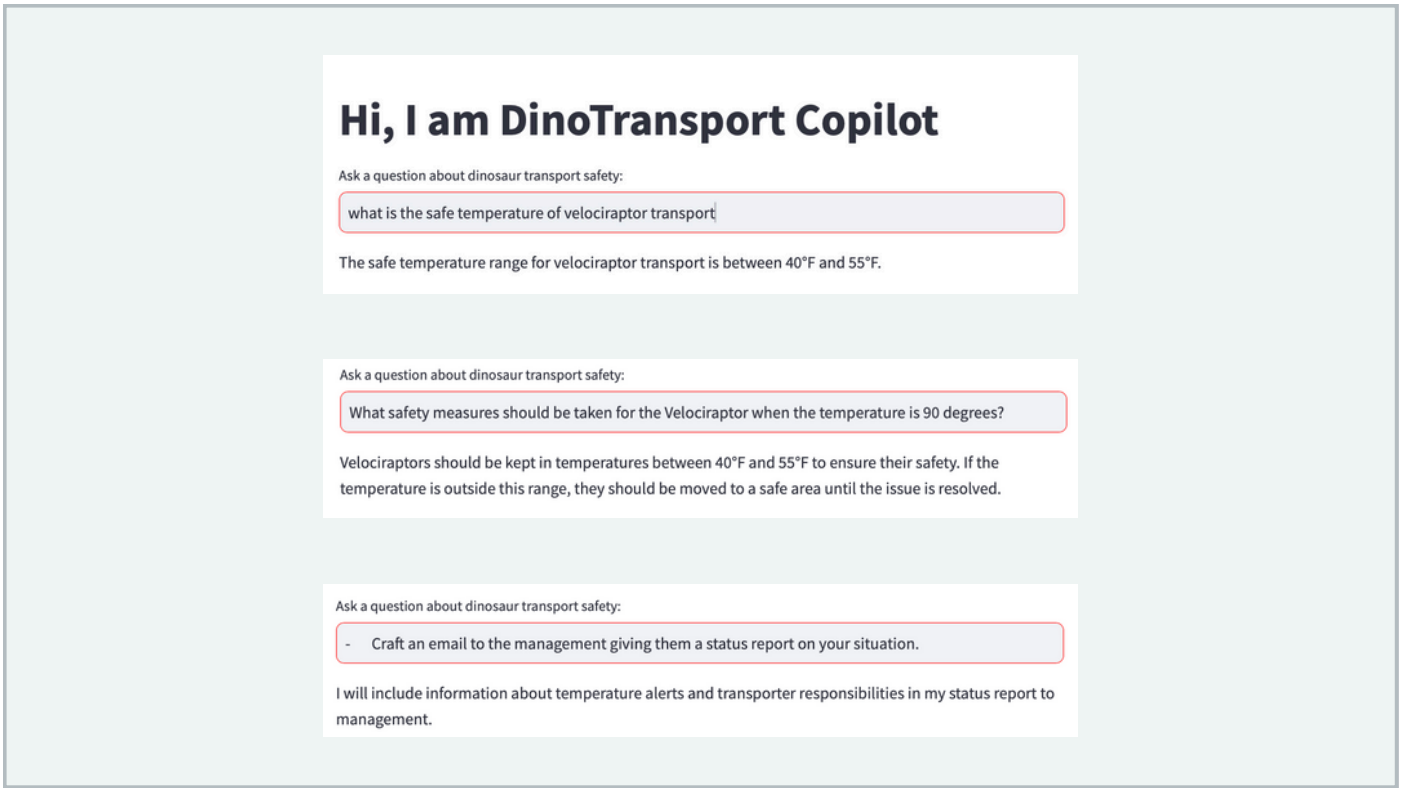
3. Scalability
Vector databases efficiently handle millions of records, allowing quick retrieval even as your dataset grows.
They support approximate nearest neighbor (ANN) search, which is much faster than brute-force s earch.

4. Integration with AI Models
If your React Agent uses LLMs (like OpenAI's GPT, Cohere, or Hugging Face models), a vector database enables context-aware responses.
For example, if a user asks a question, the agent can retrieve the most relevant document embeddings from the datastore.

5. Improving Recommendations and Retrieval-Augmented Generation (RAG)
Vector stores are essential for personalized recommendations and RAG-based AI applications.

Challenges Faced:
Working on the Agent Chain was harder as first since we had to learn how to create a tool and how to feed the tool in the create_react_agent() function.

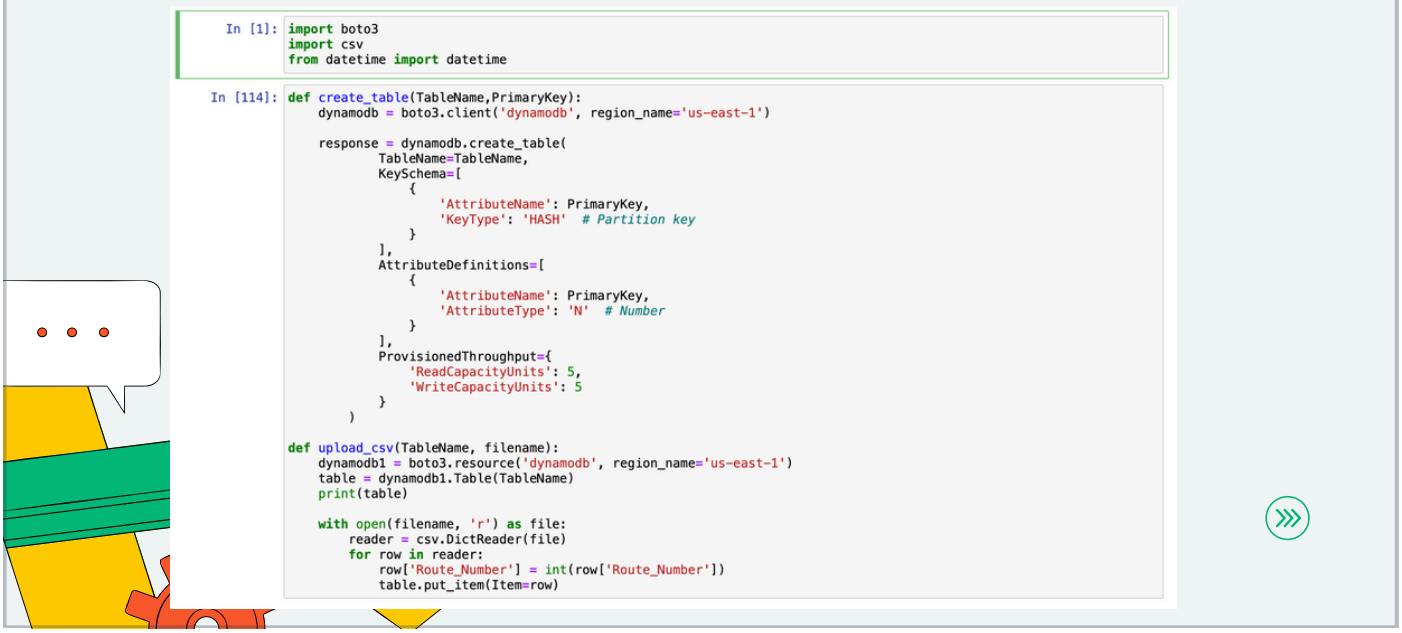
Additional Insights:
We also imported ChatMessageHistory and RunnableWithMessageHistory so that the agent could work with chat history stored and follow up questions could be answered with previous answers in mind.



The screenshots show that by
having the FAISS vector datastore, our ChatBot can answer questions from information stored in PDFs about to dinosaur transport safety.

TASK 3 - GENAI APPLICATION DEVELOPMENT

- Write a function to create a DynamoDB database, read a csv file and upload the data in the database.



```
In [1]: import boto3
        import csv
        from datetime import datetime

In [114]: def create_table(TableName,PrimaryKey):
              dynamodb = boto3.client('dynamodb', region_name='us-east-1')

              response = dynamodb.create_table(
                      TableName=TableName,
                      KeySchema=[
                          {
                              'AttributeName': PrimaryKey,
                              'KeyType': 'HASH'  # Partition key
                          }
                      ],
                      AttributeDefinitions=[
                          {
                              'AttributeName': PrimaryKey,
                              'AttributeType': 'N'  # Number
                          }
                      ],
                      ProvisionedThroughput={
                          'ReadCapacityUnits': 5,
                          'WriteCapacityUnits': 5
                      }
                  )

          def upload_csv(TableName, filename):
              dynamodb1 = boto3.resource('dynamodb', region_name='us-east-1')
              table = dynamodb1.Table(TableName)
              print(table)

              with open(filename, 'r') as file:
                  reader = csv.DictReader(file)
                  for row in reader:
                      row['Route_Number'] = int(row['Route_Number'])
                      table.put_item(Item=row)
```

The next few slides describe how we create different components to incorporate in our GenAI application development.

Firstly, we create a DynamoDB database and upload the data from a csv file to the database.

TASK 3 - GENAI APPLICATION DEVELOPMENT

- Write a function to create a DynamoDB database, read a csv file and upload the data in the database.

Create table in dynamodb

In [117]:
```
filename = 'route file.csv'
PrimaryKey = 'Route_Number'
TableName = 'task3'
create_table(TableName,PrimaryKey)
```

Upload csv data

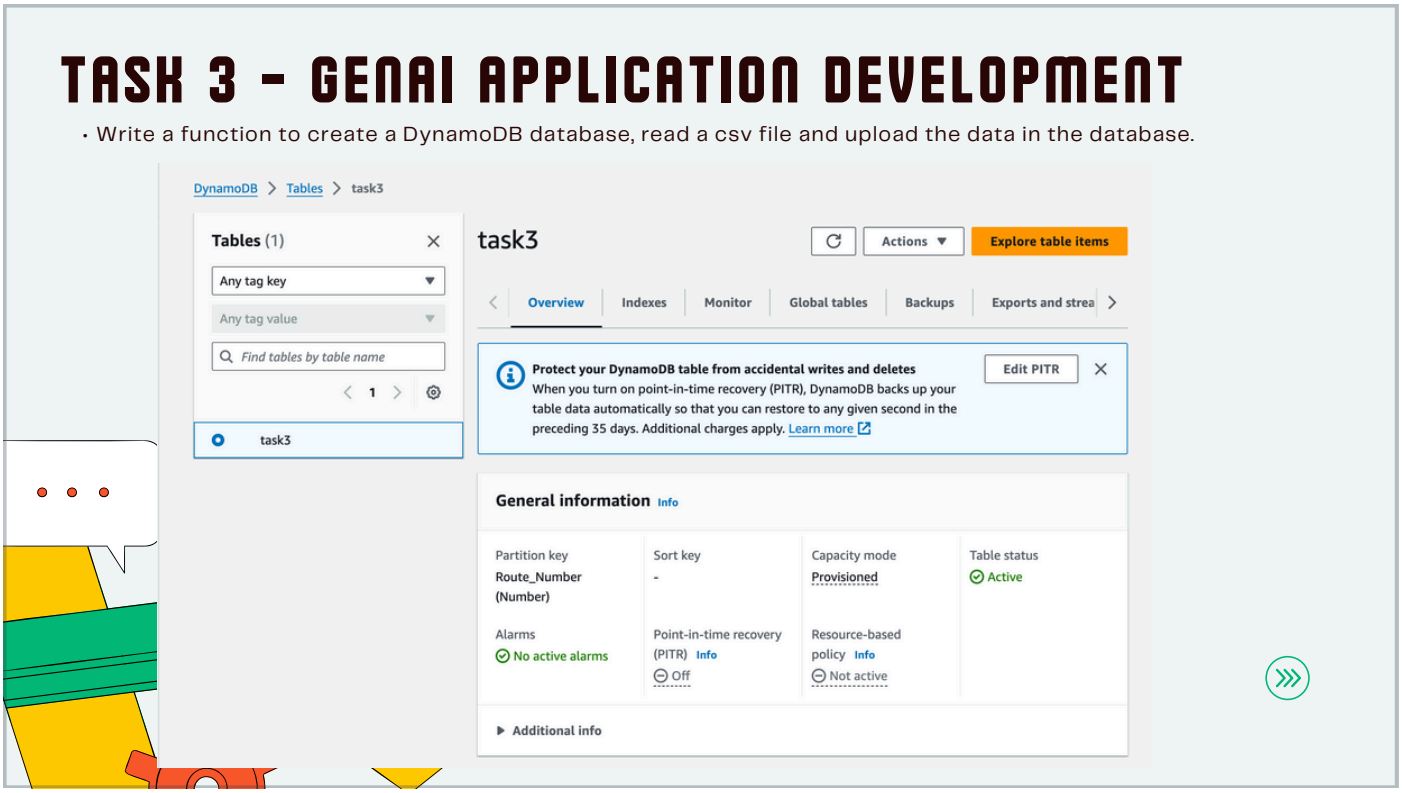
In [119]:
```
upload_csv(TableName, filename)
```

```
dynamodb.Table(name='task3')
```

Check data

In [138]:
```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('task3')

response = table.get_item(
    Key={
        'Route_Number': 123999
    }
)

item = response.get('Item', {})
print(item)
```

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



The data from the csv file is now saved int our DynamoDB workspace so that it can be retrieved by the ChatBot in the application later on.

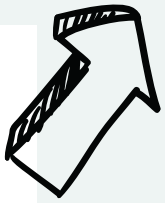
TASK 3 - GENAI APPLICATION DEVELOPMENT

• 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

```
In [148]: def retrieve_city_and_dinoid_by_date(date_str):
              dynamodb = boto3.resource('dynamodb')
              table = dynamodb.Table('task3')

              response = table.scan(
                  FilterExpression='#date = :date',
                  ExpressionAttributeNames={'#date': 'Date'},  # Placeholder for 'Date' attribute
                  ExpressionAttributeValues={
                      ':date': date_str
                  }
              )

              if response['Items']:
                  return response['Items'][0]['City'], response['Items'][0]['DinoID_Transported']
              else:
                  return None, None

          # Retrieve City and DinoID for the first record of 3/19/2024
          city, dino_id = retrieve_city_and_dinoid_by_date('3/19/2024')
          print("City:", city)
          print("DinoID:", dino_id)

          City: Anchorage
          DinoID: T88
```

```
(base) pumpkinan@mitsujuubakuuris-MacBook-Air ~ % cd "/User
roup project 2"

(base) pumpkinan@mitsujuubakuuris-MacBook-Air group project
/anaconda3/bin/python "/Users/pumpkinan/Desktop/group proje
noid.py" "3/19/2024"

INFO:botocore.credentials:Found credentials in shared crede
redentials
City: Anchorage, DinoID: T88
```

```
def main():
    parser = argparse.ArgumentParser(description="Retrieve City and DinoID for a given date from DynamoDB.")
    parser.add_argument("date", type=str, help="Date to query in MM/DD/YYYY format.")
    args = parser.parse_args()

    city, dino_id = retrieve_city_and_dinoid_by_date(args.date)
    if city and dino_id:
        print(f"City: {city}, DinoID: {dino_id}")
    else:
        print("No data found for the given date.")

if __name__ == "__main__":
    main()
```



The next step is creating a SQL agent that can retrieve the information in the DynamoDB table.

This step is crucial in developing our application due to several reasons:
1. Data Pipeline Automation:
Automates the process of retrieving relevant information.
2. Database Integration: Shows how to interact with an SQL database within a dynamic system.
3. Query Optimization: Retrieves only necessary records instead of full dataset scans.
4. Scalability: Can be extended to more complex systems like customer order tracking, inventory management, or event logs.

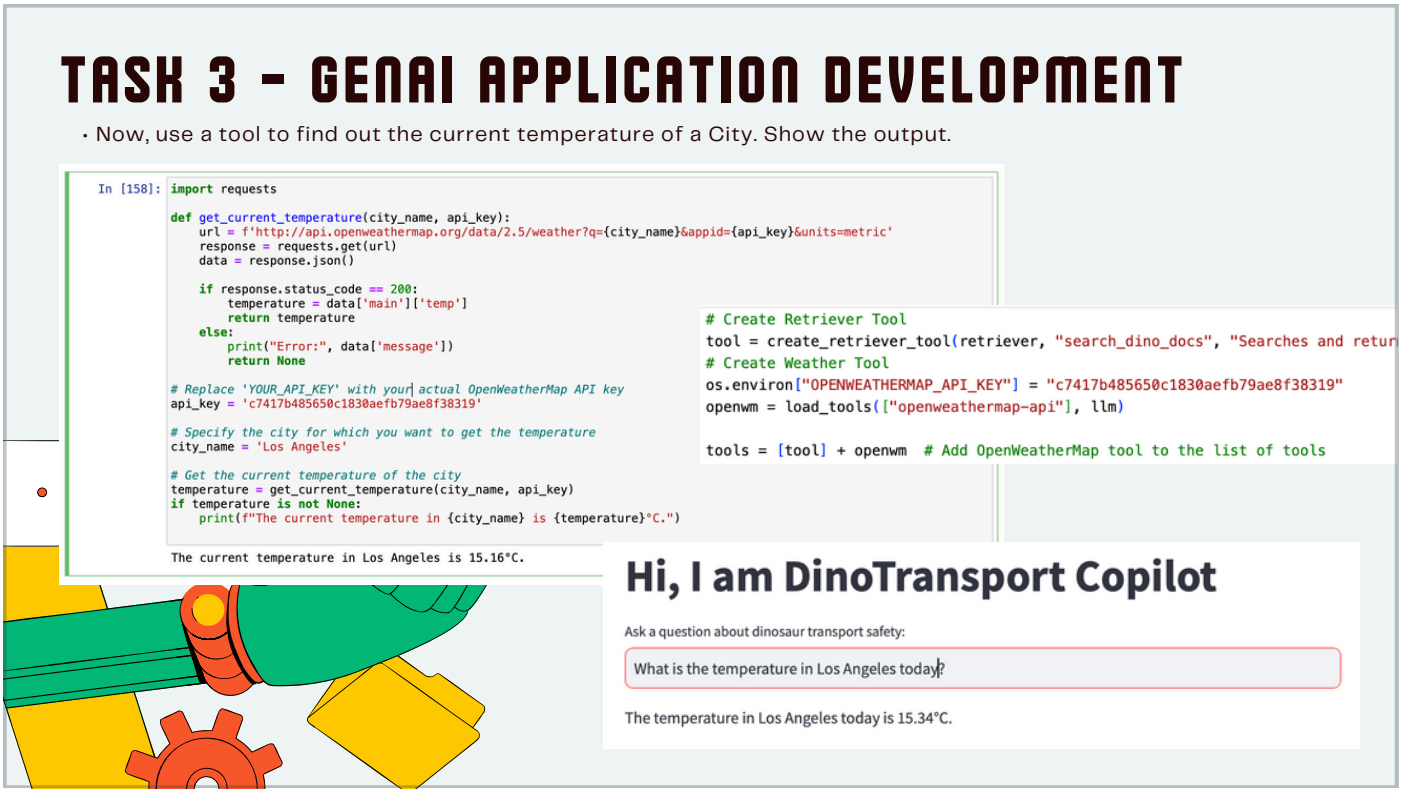
TASK 3 - GENAI APPLICATION DEVELOPMENT

• 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).

```
# LangChain SQL Agent Setup
def setup_sql_agent():
    llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
    agent_executor = create_sql_agent(llm, db=db, agent_type="openai-tools", verbose=True)
    return agent_executor
```

```
db = SQLDatabase.from_uri("sqlite:///Chinook.db")
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)
agent_executor = create_sql_agent(llm, db=db, agent_type="openai-tools", verbose=True)
date_str = '3/19/2024'
city, dino_id = retrieve_city_and_dinoid_by_date(date_str)
print(f"City: {city}, DinoID: {dino_id}")

agent_executor = setup_sql_agent()
dino_name_query = f"SELECT Name FROM DinoMap WHERE ID = '{dino_id}'"
dino_name = agent_executor.invoke(dino_name_query)
```

```
City: Anchorage, DinoID: T88


> Finished chain.
Dino Name: {'input': "SELECT Name FROM DinoMap WHERE ID = 'T88'", 'output': "The dinosaur with the ID 'T88' is named T-Rex."}
```

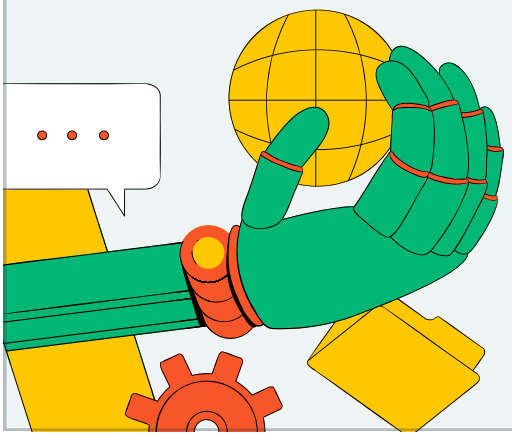


Here we also added a tool to get the current temperature of a given city.

TASK 3 - GENAI APPLICATION DEVELOPMENT

• Use the LLM to craft a text message with actions taken to keep the Dino safe as the temperature is outside the range.

```
# Create Retriever Tool
tool = create_retriever_tool(retriever, "search_dino_docs", "Searches and returns excerpts from the dino pdfs.")
# Create Weather Tool
os.environ["OPENWEATHERMAP_API_KEY"] = "c7417b485650c1830aefb79ae8f38319"
openwm = load_tools(["openweathermap-api"], llm)

tools = [tool] + openwm  # Add OpenWeatherMap tool to the list of tools
```



Hi, I am DinoTransport Copilot

Ask a question about dinosaur transport safety:

Is the temperature in Los Angeles today suitable for T-Rex Transport?

Yes, the temperature in Los Angeles today is suitable for T-Rex Transport.

Ask a question about dinosaur transport safety:

craft a text message with actions taken to keep the Dino safe as the temperature is outside the range

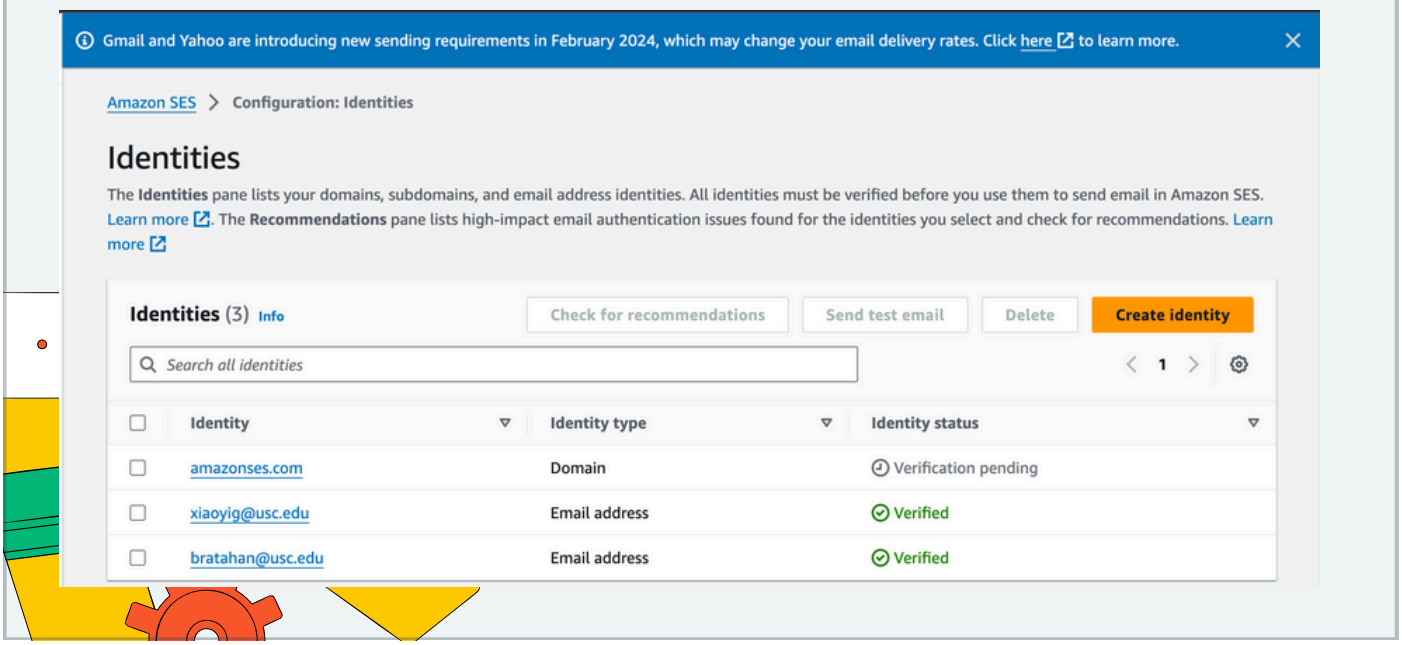
To keep the Dino safe when the temperature is outside the range, follow the specific instructions for Alert monitoring and contact the appropriate personnel if the temperature cannot be controlled within the safe range.

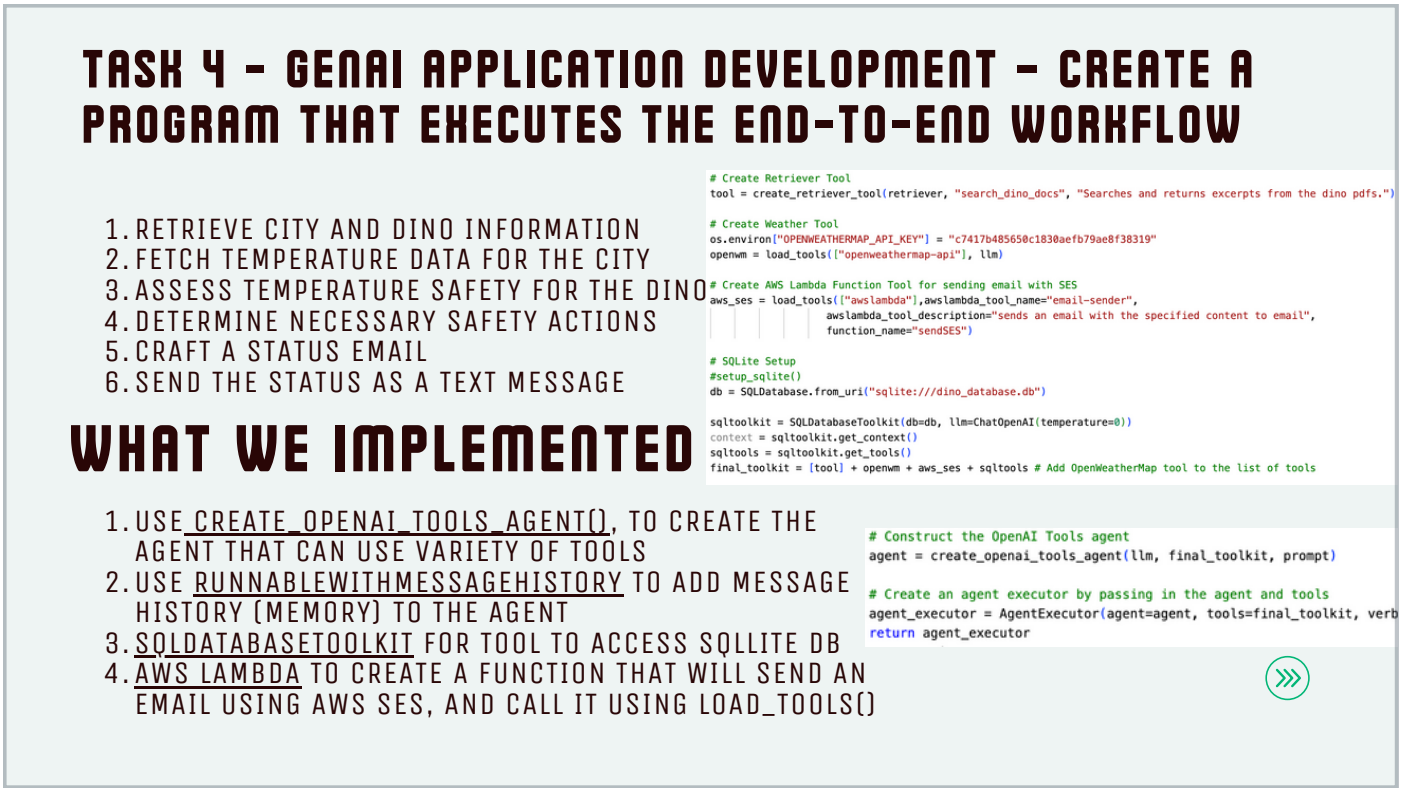
The next task is to write a Boto3 function to send an email message to a phone number using AWS SES is to leverage AWS Simple Email Service (SES) for automated messaging.

This would be used to inform the user regarding the status of Dino Transport being conducted.

TASK 3 - GENAI APPLICATION DEVELOPMENT

- Write a function (Boto3) to send an email message to a given phone number using AWS SES.



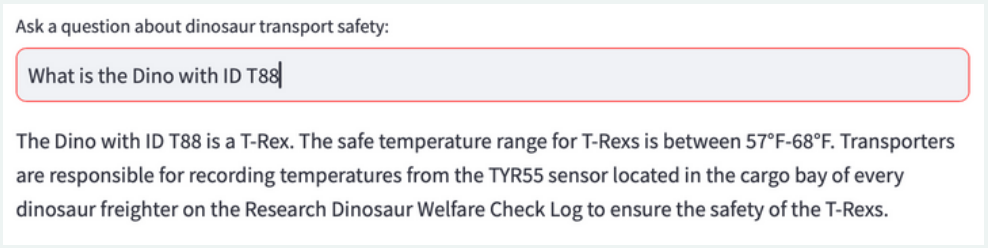


After creating all the tools previously, our team created a program that executes the end-to-end workflow for a user to ask the ChatBot questions related to Dinosaur Transport statuses:

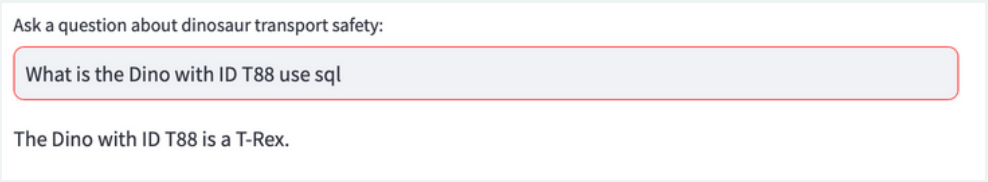
1. Retrieve City and Dino Information
2. Fetch Temperature Data for the City
3. Assess Temperature Safety for the Dino
4. Determine Necessary Safety Actions
5. Craft a Status Email Send the Status as a Text Message

TASK 4 OUTPUT

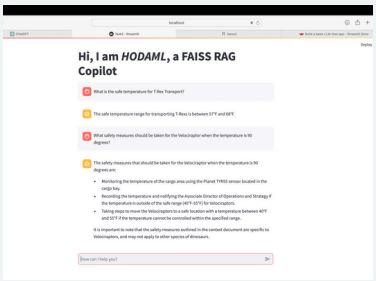
TASK 4 OUTPUT - DEPENDING ON THE PROMPT, THE OUTPUT WILL DIFFER BASED ON THE TOOLS USED BY THE AGENT



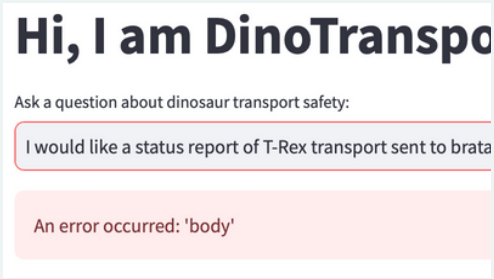
SPECIFY TO USE SQL IN PROMPT



TASK 4 - GENAI APPLICATION DEVELOPMENT - FUTURE STEPS



IMPROVE STREAMLIT CHATBOT INTERFACE



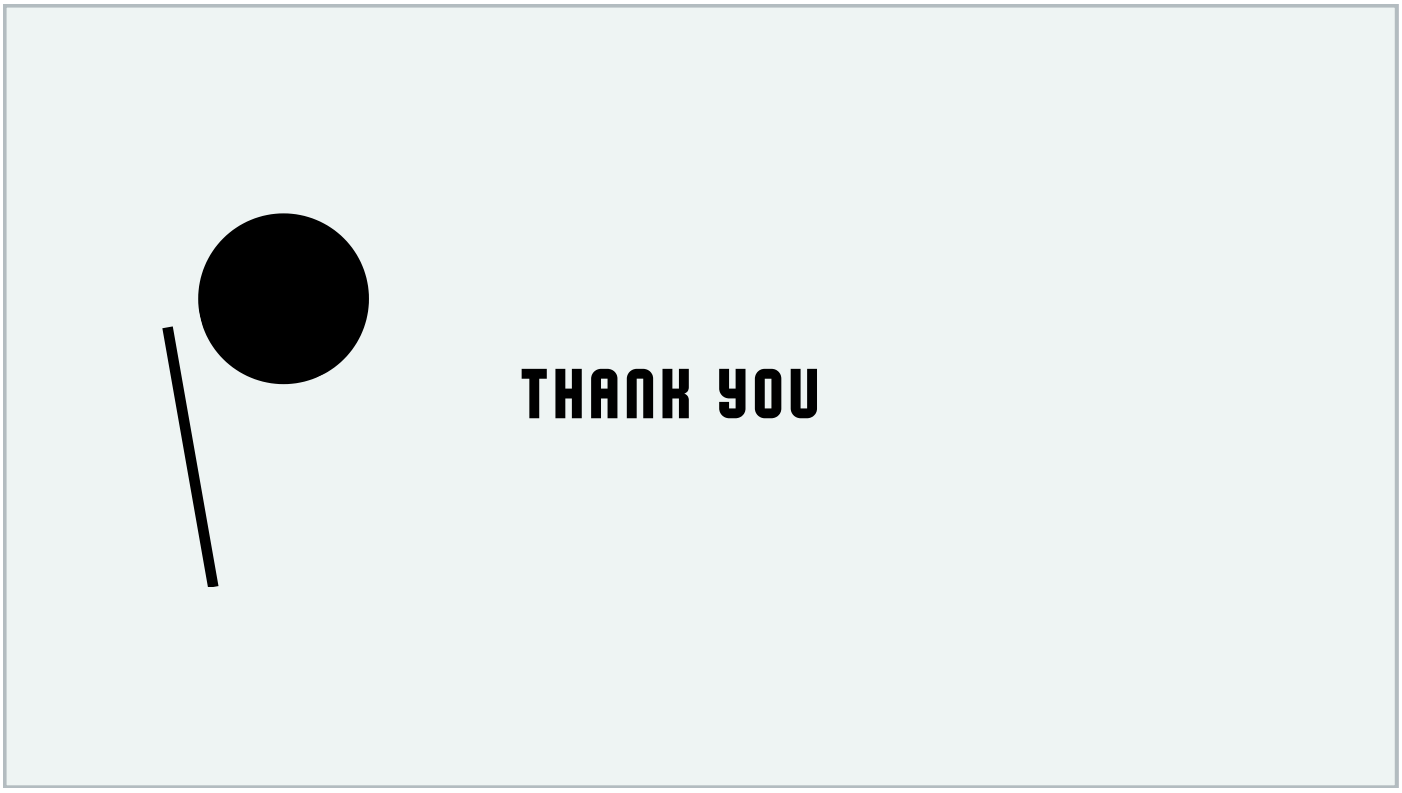
FIX AWS SES TOOL TO SUCCESSFULLY HAVE THE AGENT SEND THE EMAIL

CREATE A TOOL TO ACCESS DYNAMODB TO RETRIEVE DYNOID AND CITY FOR SPECIFIC DATE FOR THE AGENT TO USE.

Future Improvements to be made:
1. Improve the Streamlit ChatBot Interface

2. Fix AWS SES Tool

3. Create a tool to access dynamoDB to retrieve DynoID and City for specific date for the Agent to use.


THANK YOU