Lesson 2 Demo 1

LangGraph Augmented LLM: Smart Search Optimization

Objective: To showcase how to use LangGraph and an LLM to enhance web search optimization

Traditional web search queries often lack optimization, leading to irrelevant or inefficient search results. Users may struggle to formulate precise queries that capture their intent, requiring multiple refinements. Augmenting search queries with an LLM can improve search accuracy and efficiency by generating optimized queries and providing contextual reasoning.

Prerequisites:

1. Create a virtual environment
2. Activate the virtual environment
3. Install the libraries in `requirements.txt`

Steps to be followed:

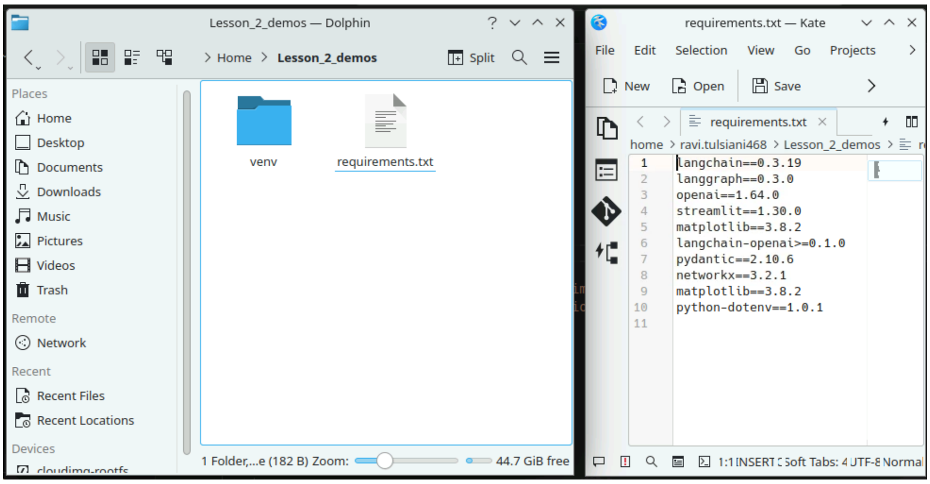
1. Set up the environment
2. Import the required libraries
3. Define a structured schema using Pydantic
4. Build a Streamlit UI
5. Run the webapp

Step 1: Set up the environment

1.1 Open command prompt and create a new folder named "**Lesson_2_demos**" and go to the respective folder using the command given below:
mkdir Lesson_2_demos
cd Lesson_2_demos

1.2 After this, install and activate the virtual environment using the command below:
python3 -m venv venv
source venv/bin/activate

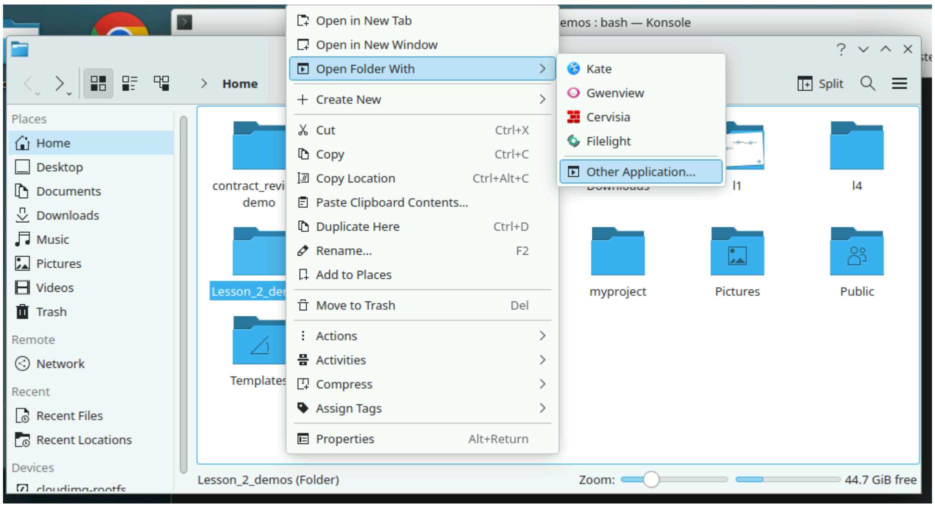
1.3 Now, create a `requirements.txt` file inside the folder with required libraries:



```
langchain==0.3.19
langgraph==0.3.0
openai==1.64.0
streamlit==1.30.0
matplotlib==3.8.2
langchain-openai>=0.1.0
pydantic==2.10.6
networkx==3.2.1
matplotlib==3.8.2
python-dotenv==1.0.1
```

1.4 Install all the required libraries using the command below:
pip install -r requirements.txt

1.5 Now, open the folder using VS Code editor:



1.6 After this, open a new Python file using the "**New File**" option and name it as "**Demo1**".

Step 2: Import the required libraries

2.1 Import necessary libraries to build the UI, process user input, and interact with the LLM.

2.2 Build the client with the required parameters and their respective values.

```
# Import required libraries

from dotenv import load_dotenv
import os
import streamlit as st
from pydantic import BaseModel
import openai

client = openai.AzureOpenAI(

api_key="2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAAB
ACOGi3sC",
    api_version="2023-12-01-preview",
    #azure_endpoint="https://openai-api-management-gw.azure-api.net/"
    azure_endpoint="https://openai-api-management-gw.azure-api.net/deployments/gpt-
4o-mini/chat/completions?api-version=2023-12-01-preview"
)
```

Step 3: Define a structured schema using Pydantic

5.1 To ensure LLM output consistency, define a structured schema using Pydantic. This schema will validate and format the response by organizing it into two key components:
- the optimized search query
- the justification for that query

```
class WebSearchPrompt(BaseModel):
    search_query: str
    justification: str
```

Step 4: Build a Streamlit UI

4.1 Create an interactive user interface using Streamlit to allow users to input their search queries.

4.2 Send the input to a local LLM backend.

4.3 Display the response returned by the local API using the Pydantic model.

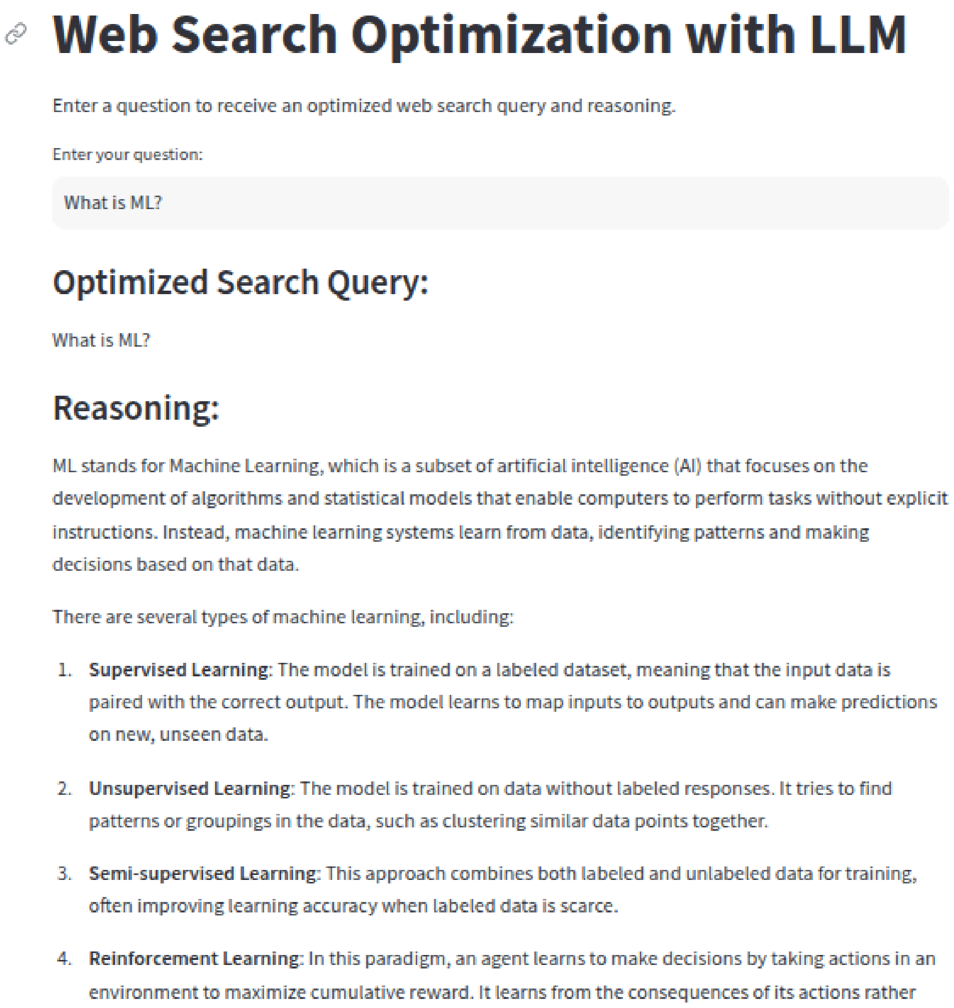
```
# Use Streamlit to create a simple UI where users can input their question and get
a structured response.
st.title("Web Search Optimization with LLM")
st.write("Enter a question to receive an optimized web search query and
reasoning.")

# Step 4: Create input field for the user's question
user_query = st.text_input("Enter your question:") # ask question



# Step 5: Process the input query and display the result
if user_query:
    # Invoke the LLM with the user query
    # Extract relevant parts of the response
    response = client.chat.completions.create(model="gpt-4o-mini",
                                          messages=[{"role": "user", "content":
user_query}],
                                          temperature=0.3)
    response_content = response.choices[0].message.content

    # Structure the output using the pydantic model
    formatted_response = WebSearchPrompt(search_query=user_query,
justification=response_content)


    # Display the structured response to the user
    st.subheader("Optimized Search Query:")
    st.write(formatted_response.search_query)  # Display the optimized search query
    st.subheader("Reasoning:")
    st.write(formatted_response.justification)   # Display the reasoning behind the
query
```

Step 5: Run the webapp

5.1 Save the file and then run the streamlit webapp from command prompt using the command given below:

```
streamlit run Demo1.py
```

Output:



Web Search Optimization with LLM

Enter a question to receive an optimized web search query and reasoning.

Enter your question:

What is ML?

Optimized Search Query:

What is ML?

Reasoning:

ML stands for Machine Learning, which is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform tasks without explicit instructions. Instead, machine learning systems learn from data, identifying patterns and making decisions based on that data.

There are several types of machine learning, including:

1. **Supervised Learning**: The model is trained on a labeled dataset, meaning that the input data is paired with the correct output. The model learns to map inputs to outputs and can make predictions on new, unseen data.
2. **Unsupervised Learning**: The model is trained on data without labeled responses. It tries to find patterns or groupings in the data, such as clustering similar data points together.
3. **Semi-supervised Learning**: This approach combines both labeled and unlabeled data for training, often improving learning accuracy when labeled data is scarce.
4. **Reinforcement Learning**: In this paradigm, an agent learns to make decisions by taking actions in an environment to maximize cumulative reward. It learns from the consequences of its actions rather

By following the above-mentioned steps, you have successfully used SmartQuery to enhance web searches by refining user queries with AI, ensuring more accurate and relevant results. It structures responses for better understanding using LangGraph, OpenAI LLM, Pydantic, and Streamlit. Future improvements could include real-time search integration and context-aware refinement, making searches smarter and more efficient.