Hands-On Guide: Integrating Neo4j with LLMs Using Streamlit and Together API : Hamza SAFRI

Introduction

In this hands-on tutorial, we'll build an interactive web application that leverages the power of Neo4j, a graph database, and a Large Language Model (LLM) accessed via the Together API. The application will allow users to query a movie database and receive responses generated by the LLM based on the retrieved data. This project demonstrates the principles of Retrieval-Augmented Generation (RAG) using a knowledge graph.

Objectives

- Understand how to connect and query a Neo4j database.
- Learn to use Streamlit to create an interactive web application.
- Integrate an LLM using the Together API to generate natural language responses.

- Implement RAG by combining real-time data retrieval and language model generation.

Prerequisites

Before we start, make sure you have the following:

1. **Neo4j Database**: Installed and running locally or via Docker with a dataset containing movies.
2. **Streamlit**: Installed in your Python environment.
3. **Together API Key**: Sign up at Together to get an API key.
4. **Python Packages**: Install the necessary packages using pip:

   ```
   pip install neo4j streamlit together-ai
   ```

5. **Python Environment**: Ensure you are using Python 3.7 or higher.

Step 0: Obtain Together.ai API Token

Together.ai is a company that develops open-source tools and platforms for AI, aimed at making AI development more accessible, scalable, and collaborative. Their offerings include infrastructure for large-scale AI projects, tools for collaborative model training, and a focus on ethical, transparent AI development. By fostering a community-driven ecosystem, Together.ai enables developers, researchers, and organizations to build and deploy AI models efficiently while promoting open collaboration and innovation.

To get the API key from Together.ai, follow these steps based on the provided image:

1. **Login**: Sign in to your Together.ai account.
2. **Access Settings**: On the left-hand side menu, you will see an option labeled "SETTINGS". Click on it.
3. **Navigate to API Keys**: In the settings menu, select the "API KEYS" option.
4. **View API Key**: Once you're on the "API KEYS" page, you should see your API key displayed. It's partially hidden for security, but you can copy it by clicking

the clipboard icon next to the key.

5. **Copy the Key**: Click the clipboard icon to copy the API key to your clipboard.
6. **Use the API Key**: You can now use this API key in your applications or integrations that require access to Together.ai's services.

Make sure to keep your API key secure and avoid sharing it publicly.

Step 1: Set Up Neo4j with Docker

To easily manage and deploy the Neo4j database, you can use Docker. Follow these steps to set up Neo4j using Docker:

Dockerfile for Neo4j

Create a `Dockerfile` in your project directory to define the Neo4j environment.

```
# Use the official Neo4j image from the Docker Hub
FROM neo4j:latest

# Set environment variables for Neo4j
ENV NEO4J_AUTH=neo4j/your_password_here

# Expose the default Neo4j port
EXPOSE 7474 7687
```

Build and Run the Neo4j Container

1. **Save the Dockerfile**: Ensure the `Dockerfile` is saved in your project directory.
2. **Build the Docker Image**: Open your terminal, navigate to the directory containing the Dockerfile, and run the following command:

   ```
   docker build -t my-neo4j-image .
   ```

3. **Run the Neo4j Container**: Once the image is built, run the container using:

```
docker run -d --name my-neo4j-container -p 7474:7474 -p 76
87:7687 my-neo4j-image
```

This command will start Neo4j and map the container ports `7474` (HTTP) and `7687` (Bolt) to your local machine.

Access the Neo4j Browser

You can now access the Neo4j Browser by navigating to `http://localhost:7474` in your web browser. Log in using the username `neo4j` and the password you set in the `Dockerfile`.

Import a Sample Dataset

1. Start your Neo4j server.
2. Open Neo4j Browser at `http://localhost:7474`.
3. Use the following Cypher query to create a simple movie dataset:

```
CREATE
// Movie Nodes
(TheMatrix:Movie {title: "The Matrix", tagline: "Welcome t
o the Real World", released: 1999}),
(Inception:Movie {title: "Inception", tagline: "Your mind
is the scene of the crime", released: 2010}),
(TheDarkKnight:Movie {title: "The Dark Knight", tagline:
"Why So Serious?", released: 2008}),
(Interstellar:Movie {title: "Interstellar", tagline: "Mank
ind was born on Earth. It was never meant to die here.", r
eleased: 2014}),
(FightClub:Movie {title: "Fight Club", tagline: "Mischief.
Mayhem. Soap.", released: 1999}),
(ThePrestige:Movie {title: "The Prestige", tagline: "Are y
ou watching closely?", released: 2006}),
(Gladiator:Movie {title: "Gladiator", tagline: "A Hero Wil
l Rise.", released: 2000}),
(Titanic:Movie {title: "Titanic", tagline: "Nothing on Ear
```

```
th could come between them.", released: 1997}),
(TheGodfather:Movie {title: "The Godfather", tagline: "An
offer you can't refuse.", released: 1972}),
(PulpFiction:Movie {title: "Pulp Fiction", tagline: "Just
because you are a character doesn't mean you have characte
r.", released: 1994}),
(ForrestGump:Movie {title: "Forrest Gump", tagline: "Life
is like a box of chocolates.", released: 1994}),
(StarWars:Movie {title: "Star Wars", tagline: "A long time
ago in a galaxy far, far away...", released: 1977}),

// Actor Nodes
(Keanu:Actor {name: "Keanu Reeves"}),
(Carrie:Actor {name: "Carrie-Anne Moss"}),
(Laurence:Actor {name: "Laurence Fishburne"}),
(Leonardo:Actor {name: "Leonardo DiCaprio"}),
(Joseph:Actor {name: "Joseph Gordon-Levitt"}),
(Ellen:Actor {name: "Ellen Page"}),
(Christian:Actor {name: "Christian Bale"}),
(Heath:Actor {name: "Heath Ledger"}),
(Matthew:Actor {name: "Matthew McConaughey"}),
(Anne:Actor {name: "Anne Hathaway"}),
(Edward:Actor {name: "Edward Norton"}),
(Brad:Actor {name: "Brad Pitt"}),
(Russell:Actor {name: "Russell Crowe"}),
(Joaquin:Actor {name: "Joaquin Phoenix"}),
(Kate:Actor {name: "Kate Winslet"}),
(Marlon:Actor {name: "Marlon Brando"}),
(Al:Actor {name: "Al Pacino"}),
(John:Actor {name: "John Travolta"}),
(Samuel:Actor {name: "Samuel L. Jackson"}),
(Tom:Actor {name: "Tom Hanks"}),
(Harrison:Actor {name: "Harrison Ford"}),

// Relationships
(Keanu)-[:ACTED_IN]->(TheMatrix),
```

```
(Carrie)-[:ACTED_IN]->(TheMatrix),
(Laurence)-[:ACTED_IN]->(TheMatrix),
(Leonardo)-[:ACTED_IN]->(Inception),
(Joseph)-[:ACTED_IN]->(Inception),
(Ellen)-[:ACTED_IN]->(Inception),
(Christian)-[:ACTED_IN]->(TheDarkKnight),
(Heath)-[:ACTED_IN]->(TheDarkKnight),
(Matthew)-[:ACTED_IN]->(Interstellar),
(Anne)-[:ACTED_IN]->(Interstellar),
(Edward)-[:ACTED_IN]->(FightClub),
(Brad)-[:ACTED_IN]->(FightClub),
(Russell)-[:ACTED_IN]->(Gladiator),
(Joaquin)-[:ACTED_IN]->(Gladiator),
(Leonardo)-[:ACTED_IN]->(Titanic),
(Kate)-[:ACTED_IN]->(Titanic),
(Marlon)-[:ACTED_IN]->(TheGodfather),
(Al)-[:ACTED_IN]->(TheGodfather),
(John)-[:ACTED_IN]->(PulpFiction),
(Samuel)-[:ACTED_IN]->(PulpFiction),
(Tom)-[:ACTED_IN]->(ForrestGump),
(Harrison)-[:ACTED_IN]->(StarWars),
```

Step 2: Build the Streamlit Application

Create a new Python file, e.g., `movie_app.py`, and begin by importing the necessary libraries and setting up the Neo4j connection.

```
import os
import streamlit as st
from neo4j import GraphDatabase
from together import Together

# Neo4j connection details
uri = "bolt://localhost:7687"
user = "neo4j"
password = "your_password_here"
```

```
# Create a Neo4j driver instance
driver = GraphDatabase.driver(uri, auth=(user, password))

# Initialize the Together API client
client = Together(api_key=os.environ.get("TOGETHER_API_KEY"))
```

Explanation:

- We use Neo4j's `GraphDatabase.driver` to connect to the local database.
- The `Together` client is initialized using an API key stored in environment variables for security.

Step 3: Define the Core Functions

These functions will handle querying the database, creating prompts for the LLM, and generating responses.

3.1 Query Neo4j

Define a function to execute Cypher queries on Neo4j:

```
def query_neo4j(tx, query, params=None):
    try:
        result = tx.run(query, params or {})
        return [record for record in result]
    except Exception as e:
        st.error(f"Neo4j query error: {str(e)}")
        return []
```

Purpose:

This function runs queries on the Neo4j database, handling any exceptions that occur during the execution.

3.2 Retrieve Information

Create a function to retrieve movie data from Neo4j:

```
def retrieve_information():
    query = """
    MATCH (m:Movie)
    RETURN m.title AS title, m.tagline AS tagline, m.released
AS released
    LIMIT 30
    """

    with driver.session() as session:
        results = session.read_transaction(query_neo4j, quer
y)

    if results:
        return "\\\\n".join([f"Title: {r['title']}, Tagline:
{r['tagline']}, Released: {r['released']}" for r in results])
    else:
        return "No movies found in the database."
```

Purpose:

This function retrieves movie data from Neo4j and formats it for display or use in prompts.

3.3 Create Prompt

Define a function to create a prompt for the LLM:

```
def create_prompt(retrieved_info, user_query):
    prompt = f"Based on the following movie information:\\\\n
{retrieved_info}\\\\n\\\\nAnswer the question: {user_query}"
    return prompt
```

Purpose:

This function combines retrieved movie data and the user's question into a single prompt for the language model. This step is crucial for RAG (Retrieval-Augmented

Generation), where the prompt acts as a bridge between the factual data from the Neo4j database and the language generation capabilities of the LLM.

3.4 Generate Response

Create a function to generate a response using the Together API:

```
def generate_response(prompt):
    try:
        stream = client.chat.completions.create(
            model="meta-llama/Meta-Llama-3.1-8B-Instruct-Turb
o",
            messages=[{"role": "user", "content": prompt}],
            stream=True,
            max_tokens=1000,
            temperature=0.1,
            top_p=0.9,
            top_k=50,
        )

        generated_text = ''
        for chunk in stream:
            if chunk.choices:
                content = chunk.choices[0].delta.content
                if content:
                    generated_text += content
                    yield content

        if not generated_text:
            yield "No response generated from the model."
    except Exception as e:
        yield f"Error with model: {str(e)}"
```

Purpose:

This function sends the prompt to the Together API and streams the language model's response, allowing for real-time feedback. The LLM processes the

augmented prompt to provide a natural language response, demonstrating the integration of real-world data into LLM capabilities

.

Step 4: Build the Streamlit Interface

Use Streamlit to create the user interface:

```
def main():
    st.title("Movie Knowledge Graph with LLM")
    st.write("Ask anything about the movies in the databas
e!")

    user_query = st.text_input("Enter your question:")
    if st.button("Submit"):
        with st.spinner("Retrieving data from Neo4j..."):
            retrieved_info = retrieve_information()

        with st.spinner("Generating response..."):
            prompt = create_prompt(retrieved_info, user_quer
y)
            response = generate_response(prompt)
            for r in response:
                st.write(r)

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

Explanation:

- **UI Elements**: The title, text input, and button form the main interactive components.
- **User Input**: When the user submits a query, the application retrieves data from Neo4j, creates a prompt, and then generates a response using the LLM.
- **Real-Time Feedback**: The `st.spinner` provides visual feedback while the data is being processed and the response is being generated.

Step 5: Run the Application

To run the application, use the following command in your terminal:

```
streamlit run movie_app.py
```

This will start the Streamlit server, and you'll be able to interact with your application by navigating to the provided local URL in your browser.
