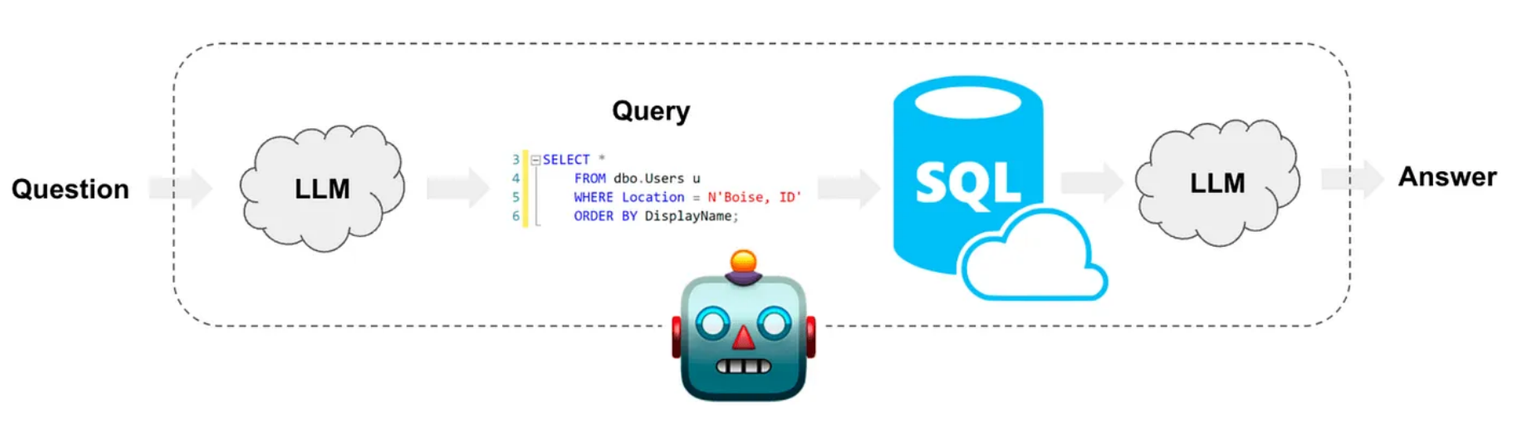
LLM-SQL Integration with LangChain: Building an AI Assistant for SQL Queries

Syahmisajid Follow 4 min read · Jan 31, 2025

In the world of modern data processing, querying databases efficiently is critical, and natural language interfaces (NLIs) have become essential tools to make this process more intuitive. Combining the power of Large Language Models (LLMs) and LangChain, an open-source framework for building language model applications, opens the door to transforming complex database interactions into simple human-like conversations.



In this article, we explore how to integrate an LLM, specifically *Ollama's LLama* model, with SQL queries using LangChain. We'll guide you through the process of setting up the environment, loading models, and querying databases directly from natural language. By the end of this article, you'll know how to harness the power of LLMs to execute MySQL queries seamlessly.

Prerequisites

Before we dive into the code, make sure you have the following installed and configured:

1. **LangChain**: For connecting with language models and creating chains for interaction.
2. **Ollama's LLaMA model**: An LLM capable of understanding and generating text.
3. **SQL Database (MySQL)**: We'll be using a MySQL database, so ensure you have one set up.
4. **Environment Setup**: Python environment with `pymysql`, `langchain`, and `dotenv`.

Setting Up the Environment

We'll begin by installing the necessary libraries and loading the environment variables.

```
!pip install pymysql
```

In your code, make sure to load the environment variables from `.env` using `dotenv`:

```
from dotenv import load_dotenv
import os

os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
load_dotenv()
```

Initializing LangChain and LLaMA

LangChain facilitates interactions between LLMs and different tools. We start by loading the LLaMA model, `llama3.2:3b`, and setting the base URL for connection.

```
from langchain_ollama import ChatOllama
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate

llm = ChatOllama(model='llama3.2:3b', base_url='http://localhost:11434')
llm.invoke('hi')  # Test connection
```

Connecting to the Database

Now, we connect to a MySQL database using `SQLDatabase` from LangChain's community utilities. Make sure the MySQL server is running and accessible.

```
from langchain_community.utilities import SQLDatabase
db = SQLDatabase.from_uri("mysql+pymysql://root:********@localhost/belajar_mysql
name_all_tables = db.get_table_names()
print(name_all_tables)
```

This fetches all table names from the connected database, giving you an overview of its structure.

Querying the Database

LangChain provides an easy way to build SQL query chains, and we can easily ask our LLM to query the database in plain language.

```
from langchain.chains import create_sql_query_chain

sql_chain = create_sql_query_chain(llm, db)
```

Question Answering with LLM

Now, let's create a chain that answers questions about our database by integrating the LLM into a conversational interface. We'll use a prompt template to structure the question-answer process.

```
from langchain_core.prompts import (SystemMessagePromptTemplate,
                                    HumanMessagePromptTemplate)

system = SystemMessagePromptTemplate.from_template("""You are helpful AI assista
prompt = """Answer the user's question based on the provided context ONLY! If yo
            ### Context:
            {context}
            ### Question:
            {question}
            ### Answer:"""
prompt = HumanMessagePromptTemplate.from_template(prompt)
messages = [system, prompt]
template = ChatPromptTemplate(messages)
qna_chain = template | llm | StrOutputParser()

def ask_llm(context, question):
    return qna_chain.invoke({'context': context, 'question': question})
```

Creating the SQL Query Chain

The next step is to integrate the database querying capability. Here, we will build a chain that generates an SQL query based on the user's question and context.

```
@chain
def get_correct_sql_query(input):
    context = input['context']
    question = input['question']

intruction = """
        Use the above context to fetch the correct SQL query for the following q
        {}
        Do not enclose the query in ```sql and do not write preamble or explanat
        You MUST return only a single SQL query.
    """.format(question)
    response = ask_llm(context=context, question=intruction)
    return response
```

Executing SQL Queries

To execute SQL queries generated by the LLM, we use `QuerySQLDataBaseTool` from LangChain's tools module.

```
from langchain_community.tools.sql_database.tool import QuerySQLDataBaseTool

execute_query = QuerySQLDataBaseTool(db=db)
sql_query = create_sql_query_chain(llm, db)
final_chain = (
    {'context': sql_query, 'question': RunnablePassthrough()}
    | get_correct_sql_query
    | execute_query
)
```

Example: Asking a Question

Now that everything is set up, let's ask a question. In this case, we'll ask about the number of transactions in the database.

```
question = "how many transactions are there? You MUST RETURN ONLY MYSQL QUERIES.
response = final_chain.invoke({'question': question})
print(response)
```

This will generate and execute the corresponding SQL query, giving us the result directly from the database.

Conclusion

In this article, we've built an integrated solution for querying MySQL databases using LangChain and LLaMA. The solution leverages natural language processing to generate SQL queries based on user input, making database interaction intuitive and efficient.

With this setup, you can create a smart assistant capable of interacting with databases, answering complex questions, and even executing SQL commands without needing to write queries manually. This integration opens up exciting possibilities for building conversational AI tools tailored to specific

Are you interested in trying out this solution? Find the complete source code on [GitHub]!