Tác động DB với LangChain và SQLModel

Giới thiệu

Trong bài này, chúng ta sẽ xây dựng một chatbot có khả năng tương tác với cơ sở dữ liệu SQLite thông qua LangChain và SQLModel. Chatbot sẽ có thể hiểu câu hỏi bằng tiếng Việt và tạo ra các câu truy vấc SQL tương ứng.

Các thư viện sử dụng

{ Alex Dev }

```
# LangChain components
from langchain_ollama import OllamaLLM  # Tích hợp mô hình Ollama
from langchain.chains.sql_database.query import
create_sql_query_chain  # Tạo chuỗi truy vấn SQL
from langchain.prompts import PromptTemplate  # Template cho prompt
from langchain_community.tools import QuerySQLDataBaseTool  # Tool
truy vấn DB
from langchain.sql_database import SQLDatabase  # Wrapper cho DB
from langchain_core.output_parsers import StrOutputParser  # Parser
output
from langchain_core.runnables import RunnablePassthrough  # Pipeline
processing

# SQLModel for ORM
from sqlmodel import Field, SQLModel, create_engine, Session, select

# Caching
from langchain.cache import InMemoryCache
```

```
from langchain.globals import set_llm_cache
```

Thiết lập Model và Database

1. Khởi tạo Language Model

```
{ Alex Dev }

llm = OllamaLLM(model="qwen2.5-coder:0.5b")
```

Qwen2.5-coder là một mô hình ngôn ngữ được tối ưu cho code, kích thước 0.5B parameters.

2. Định nghĩa Schema Database

```
{ Alex Dev }

class Blog(SQLModel, table=True):
    id: int = Field(primary_key=True)
    title: str
    content: str

sqlite_file = "blog.db"
engine = create_engine(f"sqlite:///{sqlite_file}")
```

3. Các hàm thao tác với Database

{ Alex Dev }

```
def create_db_and_tables():
    SQLModel.metadata.create_all(engine)

def create_blog(title: str, content: str):
    with Session(engine) as session:
        blog = Blog(title=title, content=content)
        session.add(blog);
        session.commit();
        session.refresh(blog);
        return blog;
```

Xây dựng Chain xử lý

1. Khởi tạo SQL Database wrapper

{ Alex Dev }

```
db = SQLDatabase(engine)
sql_chain = create_sql_query_chain(llm=llm, db=db)
```

2. Hàm chuẩn hóa câu truy vấn SQL

{ Alex Dev }

```
def standardize_query(query: str):
    return query.split("```sql")[1].split("```")[0].strip()
```

3. Xây dựng Chain xử lý hoàn chỉnh

{ Alex Dev }

```
llm_chain = (
    RunnablePassthrough.assign(
        question = lambda x: x["question"],
    ).assign(
        sql = lambda x : sql_chain.invoke({"question":
x["question"]})
    ).assign(
        standardize_query_sql = lambda x: standardize_query(x["sql"]
if '```sql' in x["sql"] else x["question"])
    ).assign(
        result = lambda x: db.run(x["standardize_query_sql"])
    ).assign(
        final_result = lambda x: [dict(row) for row in x["result"]]
    )
)
```

Cấu hình Generation Parameters

{ Alex Dev }

```
generation_params = {
    "temperature": 0.7,         # Điều chỉnh độ ngẫu nhiên (0-1)
    "top_k": 10,                # Giới hạn từ vựng top K tokens
    "top_p": 0.95,              # Ngưỡng nucleus sampling
    "num_ctx": 2048,            # Kích thước cửa sổ ngữ cảnh
    "num_thread": 1,            # Số luồng sử dụng
    "num_predict": 200,         # Số tokens tối đa dự đoán
    "repeat_last_n": 64,        # Số tokens cuối cùng xem xét lặp lại
    "repeat_penalty": 1.15      # Hệ số phạt lặp lại
```

```
}
```

Tài liệu tham khảo

1. LangChain Documentation
2. SQLModel Documentation
3. Ollama Documentation
4. SQLite Documentation

Giải thích luồng xử lý

1. Khi người dùng đặt câu hỏi, `llm_chain` sẽ:
    - Nhận câu hỏi đầu vào
    - Chuyển đổi thành câu truy vấn SQL thông qua LLM
    - Chuẩn hóa câu truy vấn SQL
    - Thực thi truy vấn trên database
    - Chuyển đổi kết quả thành định dạng dict
2. Cache được sử dụng để lưu trữ các kết quả truy vấn, giúp tăng tốc độ xử lý cho các câu hỏi lặp lại.
3. Generation parameters được tinh chỉnh để:
    - Đảm bảo tính đa dạng trong câu trả lời (temperature, top_k, top_p)
    - Kiểm soát độ dài output (num_predict)
    - Tránh lặp lại thông tin (repeat_penalty)