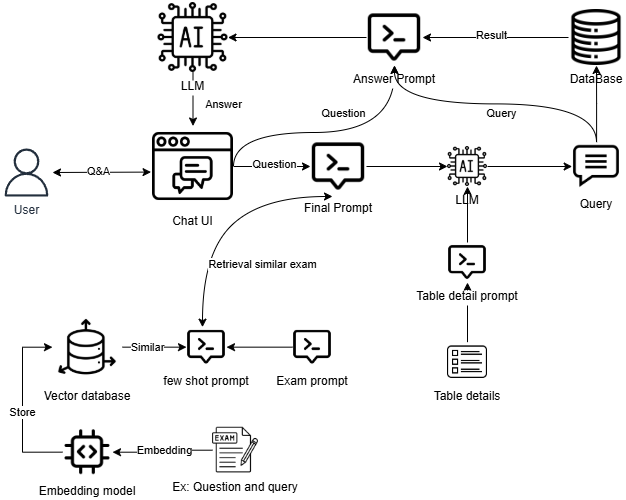
# Mô hình tổng quan



# Các bước thực hiện

Ứng dụng được triển khai trên nền tảng pycharm

## Install, Import các gói cần thiết và khai báo các API

Các gói cần install: langchain, pandas, pydantic, langchain\_openai, streamlit, langchain\_community, langchain\_core, faiss,… (Phiên bản mới nhất 20/12/2024)

Tạo file .env và khai báo các key:

|  |
| --- |
| OPENAI\_API\_KEY= your\_openai\_api\_key LANGCHAIN\_API\_KEY=your\_langchain\_api\_key  LANGCHAIN\_TRACING\_V2=true |

Có thể thay thế openai bằng gemini

Lấy key gemini ở <https://aistudio.google.com/apikey>

Lấy key langchain ở <https://smith.langchain.com/>

Tạo file chính main.py và import các thư viện sau:

|  |
| --- |
| import os from operator import itemgetter from typing import List  from dotenv import load\_dotenv from pydantic import BaseModel, Field import pandas as pd import streamlit as st from langchain\_openai import ChatOpenAI, OpenAIEmbeddings from langchain\_community.utilities import SQLDatabase from langchain\_community.tools import QuerySQLDatabaseTool from langchain\_core.output\_parsers import StrOutputParser from langchain\_core.runnables import RunnablePassthrough from langchain\_core.prompts import ChatPromptTemplate, MessagesPlaceholder, FewShotChatMessagePromptTemplate, PromptTemplate from langchain.vectorstores import FAISS from langchain\_core.example\_selectors import SemanticSimilarityExampleSelector from langchain.chains import create\_sql\_query\_chain from langchain.chains.openai\_tools import create\_extraction\_chain\_pydantic from langchain\_community.chat\_message\_histories import ChatMessageHistory |

## Kết nối database (MS SQL server, ….) và cấu hình LLM

Thông tin sơ bộ về database dùng để xây dựng.

A screenshot of a computer

Description automatically generated

Cấu hình các API đã khai báo. Kết nối tới sql server(local). Thay thế các thông tin bên dưới để sữ dụng

User: usera

Pass: 123456

Database: Demo

Driver: ODBC+Driver+17+for+SQL+Server

|  |
| --- |
| load\_dotenv() working\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_)) os.environ["KMP\_DUPLICATE\_LIB\_OK"]="TRUE"  # Kết nối db connection\_string = f"mssql+pyodbc://usera:123456@localhost/Demo?driver=ODBC+Driver+17+for+SQL+Server" db = SQLDatabase.from\_uri(connection\_string) |

Cấu hình LLM

|  |
| --- |
| llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0) |

Có thể dùng các LLM miễn phí khác như gemini, TogetherAI,…

## Xây dựng “Dynamic Few-Shot Example Selection”

Mục đích của việc xây dựng “Dynamic Few-Shot Example Selection”: Đối với những câu hỏi phức tạp ta có thể dùng phương pháp này để cải thiện độ chính xác.

|  |
| --- |
| #Dynamic Few-Shot Example Selection  examples = [  {  "input": "Chi tiết danh sách khách hàng?",  "query": "SELECT \* FROM custome"  },  {  "input": "Danh sách khách hàng?",  "query": "SELECT [customer\_name] FROM customer"  }  ]  example\_prompt = ChatPromptTemplate.from\_messages(  [  ("human", "{input}\nSQLQuery:"),  ("ai", "{query}"),  ]  )  example\_selector = SemanticSimilarityExampleSelector.from\_examples(  examples,  OpenAIEmbeddings(),  FAISS,  k=1,  input\_keys=["input"],  )  few\_shot\_prompt = FewShotChatMessagePromptTemplate(  example\_prompt=example\_prompt,  example\_selector=example\_selector,  input\_variables=["input","top\_k"],  ) |

Xây dựng Few\_shot\_prompt:

* Example\_prompt: Xác định cách thức biểu diễn và chứa các các ví dụ ở trên.
* example\_selector (SemanticSimilarityExampleSelector):
  + Dùng FAISS và OpenAIEmbeddings để tính toán mức độ tương đồng ngữ nghĩa giữa câu hỏi của người dùng và các ví dụ trong examples. Ngoài OpenAI embedding ra có thể thay thế bằng các embedding model khác như Hugging Face, …
  + Chọn ra k=1 (ví dụ phù hợp nhất) dựa trên nội dung câu hỏi đầu vào
* Input đầu vào hỗ trợ chọn các ví dụ phù hợp

## Xây dựng “Dynamic Relevant Table Selection”\

Mục đích của việc xây dựng “Dynamic Relavant Table Selection” là giúp mô hình xác định chính xác các bảng SQL cần sử dụng dựa trên câu hỏi của người dùng và mô tả bảng. Giúp giảm thiểu tài nguyên và tốc độ khi truy vấn đối với những database có số lượng bảng lớn.

Cần tạo 1 file csv chứa thông tin mô tả các bảng:

A screenshot of a computer

Description automatically generated

|  |
| --- |
| def get\_table\_details():  table\_description = pd.read\_csv("data\_table\_description.csv")  table\_docs = []   # Duyệt qua các hàng của DataFrame để tạo đối tượng Document  table\_details = ""  for index, row in table\_description.iterrows():  table\_details = table\_details + "Table Name:" + row['Table'] + "\n" + "Table Description:" + row['description'] + "\n\n"   return table\_details   class Table(BaseModel):  *"""Table in SQL database."""* name: str = Field(description="Name of table in SQL database.")  table\_details = get\_table\_details()\  table\_details\_prompt = f"""Return the names of ALL the SQL tables that MIGHT be relevant to the user question. \ The tables are: {table\_details} Remember to include ALL POTENTIALLY RELEVANT tables, even if you're not sure that they're needed."""  def get\_tables(tables: List[Table]) -> List[str]:  tables = [table.name for table in tables]  return tables  select\_table = {"input": itemgetter("question")} | create\_extraction\_chain\_pydantic(Table, llm, system\_message=table\_details\_prompt) | get\_tables |

* Hàm get\_table\_details() đọc file csv và trả về thông tin tổng hợp dưới dạng chuỗi.
* Định nghĩa model Table bằng pydantic
* Table\_details\_prompt: Yêu cầu mô hình liệt kê các bảng có liên quan đến câu hỏi người dùng.
* Table\_chain tạo chain(chuỗi) dùng LLM xử lí prompt và trích xuất các bảng liên quan
* Get\_table() hàm dùng để chuyển danh sách Table thành danh sách tên bảng.
* select\_table: Tự động ánh xạ câu hỏi người dùng qua các bước

## Xây dựng Main Chain(chuỗi chính)

Khai báo history dùng để lưu trữ lịch sử hội thoại dùng để giữ ngữ cảnh trong suốt cuộc trò chuyện

|  |
| --- |
| history = ChatMessageHistory() |

final\_prompt: Kết hợp thông tin bảng SQL ({table\_info}), các ví dụ few-shot (few\_shot\_prompt), và câu hỏi người dùng ({input}) để hướng dẫn LLM tạo truy vấn SQL chính xác.

|  |
| --- |
| final\_prompt = ChatPromptTemplate.from\_messages(  [  ("system",  """  "Bạn là một chuyên gia MySQL. Dựa trên câu hỏi đầu vào, hãy tạo một truy vấn MySQL cú pháp chính xác để thực thi. Trừ khi có yêu cầu cụ thể khác.  Giữ nguyên các giá trị được truy vấn trong SQL  Dưới đây là thông tin về bảng liên quan: {table\_info}  Dưới đây là một số ví dụ về câu hỏi và truy vấn SQL tương ứng.  """  ),  few\_shot\_prompt,  MessagesPlaceholder(variable\_name="messages"),  ("human", "{input}"),  ] ) |

generate\_query: Chuỗi xử lý được tạo với create\_sql\_query\_chain, kết hợp LLM, cơ sở dữ liệu, và final\_prompt để chuyển đổi câu hỏi ngôn ngữ tự nhiên thành câu truy vấn SQL.

Excute\_query (QuerySQLDatabaseTool) dùng để thực thi câu truy vấn SQL trên cơ sở dữ liệu.

|  |
| --- |
| #Tạo một chain -> SQL query với final\_prompt generate\_query = create\_sql\_query\_chain(llm, db, final\_prompt) #truy vấn SQl trên csdl execute\_query = QuerySQLDatabaseTool(db=db) |

repharase\_answer: định dạng câu trả lời bằng cách tạo một template để kết hợp câu hỏi, truy vấn SQL, và kết quả truy vấn. Sau đó đưa cho LLM xử lí đưa ra câu trả lời.

|  |
| --- |
| answer\_prompt = PromptTemplate.from\_template(  """  Question: {question}  SQL Query: {query}  SQL Result: {result}  Answer: """ ) rephrase\_answer = answer\_prompt | llm | StrOutputParser() |

Xây dựng main chain:

* select\_table: xác định các bảng liên quan
* generate\_query: tạo truy vấn sql
* execute\_query: thực thi truy vấn
* rephrase\_anwser: định dạng câu trả lời.

## Xây dựng giao diện chat

Xây dựng giao diện chat với streamlit.

|  |
| --- |
| #streamlit  if "chat\_history" not in st.session\_state:  st.session\_state.chat\_history = []  # Hiển thị lịch sử trò chuyện for message in st.session\_state.chat\_history:  with st.chat\_message(message["role"]):  st.markdown(message["content"])  # Nhập câu hỏi từ người dùng user\_input = st.chat\_input("Nhập câu hỏi của bạn...")  if user\_input:  # Lưu câu hỏi của người dùng vào lịch sử  st.session\_state.chat\_history.append({"role": "user", "content": user\_input})   # Hiển thị câu hỏi của người dùng  with st.chat\_message("user"):  st.markdown(user\_input)   # Xử lý câu hỏi và trả lời  with st.chat\_message("assistant"):  try:  # Gọi chain để xử lý câu hỏi  response = chain.invoke({"question": user\_input, "messages": history.messages})  assistant\_response = response # Kết quả trả lời  except Exception as e:  assistant\_response = f"Lỗi: {str(e)}"   # Hiển thị câu trả lời  st.markdown(assistant\_response)  st.session\_state.chat\_history.append({"role": "assistant", "content": assistant\_response}) |

# Kết quả (Demo)

A screenshot of a chat

Description automatically generated

Src: <https://github.com/K-18123/Q-A-chatbot-v1>