به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

تمرین ششم

خرداد ماه ۱۴۰۳

فهرست

سوال اول

در این تمرین، یک چت بات پاسخ‌گو در زمینه علوم کامپیوتر با تخصص ویژه در پردازش زبان‌های طبیعی را پیاده سازی خواهیم کرد. ابزارهای مورد استفاده، جدیدترین ابزارهای این حوزه هستند و از قدرتمندترین مدل‌های وزن باز[۱] برای این منظور استفاده خواهیم کرد. در نظر داشته باشید که نگاه ما به مدل‌های زبانی بزرگ به عنوان هسته کار در این تمرین، نگاه جعبه سیاهی خواهد بود. با فرض داشتن مدلی قدرتمند که در تمرین‌های قبل به تنظیم دقیق[۲] و پیاده‌سازی آن‌ها پرداخته شد، سعی خواهیم کرد کاربردهای جذابی را با استفاده از مدل مرکزی پیاده کنیم.

روند کلی چت بات ما به این گونه است که با گرفتن پیام کاربر، بین سه مسیر زیر یکی را انتخاب می‌کند:

۱. جست‌وجو در موتور جست‌وجو

۲. جست‌وجو و بازیابی اسناد مرتبط از منبع داخلی اسناد (کتاب مرجع)

۳. پاسخ عمومی

پس از انتخاب بهترین مسیر، مراحل مربوط به آن اجرا شده و در نهایت منجر به تولید پاسخ مناسب برای کاربر خواهد شد. جزئیات هر مسیر را در بخش مربوطه خواهیم دید. سعی شده است روند تمرین به کوچک‌ترین قدم‌های معنادار شکسته شود تا حل آن به آسان‌ترین شکل ممکن باشد.

ابزارهای اصلی استفاده شده در این تمرین، LangChain و LangGraph خواهند بود و طی این مسیر از موارد مختلفی مانند FAISS، HuggingFaceEmbeddings، TavilySearch، BM۲۵Retriever و TogetherAI استفاده خواهیم کرد. تنها زبان برنامه نویسی مجاز برای این پروژه python است. توضیحات مربوط به ابزارهای اصلی در کارگاه مکمل این تمرین داده خواهد شد. هم‌چنین یک نسخه کوچک‌تر از همین تمرین را در کنار هم پیاده خواهیم کرد و خروجی کارگاه می‌تواند نقطه شروعی برای انجام تمرین باشد.

---

[۱] open weight

[۲] fine-tuning

دادگان

دادگان مورد استفاده در این تمرین، فصل‌های کتاب مرجع درس خواهند بود که از سایت استنفورد[۳] قابل دسترس هستند.

بخش اول – دریافت و آماده سازی دادگان

در این بخش قصد داریم داده‌های مربوط به کتاب مرجع درس را دریافت کرده، به اندازه‌های مناسب تقسیم کنیم و در پایگاه داده بُرداری محلی برنامه ذخیره کنیم.

الف) در ابتدا به این آدرس مراجعه کنید. باید به کمک از روشی مبتنی بر کد، لینک‌های فصل‌ها را پیدا کرده و در یک لیست ذخیره کنید.

ب) پس از آن تمام اعضای این لیست را که مربوط به فصل‌های مختلف کتاب هستند توسط PdfLoader موجود در کتابخانه LangChain به فرمت مناسب تبدیل کنید.

پ) در مرحله بعد با استفاده از RecursiveCharacterTextSplitter و مقادیر مناسب برای chunk_size و chunk_overlap، اسناد را به اندازه‌های مناسب تبدیل کنید.

ت) توضیح دهید در این مرحله دقیقا چه عملی انجام می‌شود (نحوه عملکرد کلاسی که برای تکه تکه کردن متن استفاده کرده‌اید) و چرا لازم است این عملیات (تبدیل متن به بخش‌های کوچک‌تر) را انجام دهیم؟

ث) در مورد اهمیت مقدار مناسب برای chunk_size و chunk_overlap توضیح دهید و بیان کنید در صورت انتخاب مقادیر اشتباه برای این پارامترها، به چه مشکلاتی بر خواهیم خورد؟

مقادیر پیشنهادی برای این پارامترها، به ترتیب ۱۰۲۴ و ۶۴ است.

(انواع مختلفی از Splitter ها در کتابخانه LangChain پیاده شده است که هرکدام شهود و نوآوری‌های مربوط به خود را دارند. مطالعه این موارد می‌تواند دید بسیار خوبی به شما بدهد که با توجه به کاربرد، از ابزار مناسب استفاده کنید)

---

[۳] https://stanford.edu/~jurafsky/slp۳/

بخش دوم – تولید بازنمایی و پایگاه داده بُرداری

الف) با استفاده از Embedder مناسب از HuggingFaceEmbeddings و ابزار FAISS از شرکت متا، داده‌های مرحله پیش را ذخیره کنید. Embedder پیش فرض خود کلاس هاگینگ فیس برای استفاده مناسب است.

ب) در مورد اهمیت استفاده از Embedder مناسب توضیح دهید. برای مثال اگر برای به دست آوردن بازنمایی جملات فارسی از مدلی استفاده کنیم که به طور کلی داده‌های فارسی را ندیده است، به چه مشکلاتی بر خواهیم خورد؟

راهنمایی: برای استفاده بهینه از GPU های رایگان در اختیار در پلتفرم‌های آنلاین، خوب است که یک بار بازنمایی‌ها را حساب کرده و بارهای بعد از موارد ذخیره شده استفاده کنیم. با این دیدگاه و به منظور استفاده از Cache ، می‌توانید از CacheBackedEmbeddings در LangChain استفاده کنید. این قابلیت به شما این امکان را می‌دهد که اسناد را یک بار Embed کرده و ذخیره کنید و برای دفعات بعد، از بازنمایی‌های ذخیره شده استفاده کنید. قابل توجه است که نوع حافظه مورد استفاده برای ذخیره بازنمایی‌ها حائز اهمیت خواهد بود؛ در صورت زیاد بودن تعداد اسناد، استفاده از GPU برای محاسبه مجدد بازنمایی‌ها می‌تواند سریع‌تر از خواندن بازنمایی‌های ذخیره شده از روی دیسک باشد. در صورتی که هم‌چنان از نظر سرعت محاسبه بازنمایی‌ها مشکل دارید، می‌توانید تعداد محدودتری فصل را برای پایگاه داده برداری استفاده کنید (برای مثال چند فصل مرتبط با این تمرین). در صورت انتخاب تعدادی از فصل‌ها، موضوعات فصل‌ها را در ارزیابی و پرس‌وجوهای آینده در نظر بگیرید. به طور کلی هدف این تمرین، یادگیری روند و پیاده‌سازی درست با حفظ کاربرد تا حد ممکن است.

(مدل‌های بسیار متنوعی را برای Embed کردن متن‌ها در هاگینگ فیس و به ویژه ابزار sentence-transformers خواهید یافت. بررسی این مدل‌ها در کنار سایر راه‌های معرفی شده برای Embed کردن در LangChain دید بسیار خوبی در مورد استفاده از مدل مناسب به شما خواهد داد. به طور کلی در راهکارهای مبتنی بر RAG و به ویژه در زبان‌های کم منبع مانند زبان فارسی، یکی از بزرگترین چالش‌ها، چالش داشتن مدل Embedder با کیفیت مناسب برای کاربرد مدنظر است و داشتن دانش در این مورد به شما آزادی عمل بسیاری در صنعت و پژوهش خواهد داد)

بخش سوم – پیاده‌سازی بازیاب معنایی

بخش بعد (امتیازی)، کامل شده همین بخش است. در صورت انجام آن لازم نیست این بخش را انجام دهید.

الف) با استفاده از ابزار FAISS و رابط پیاده شده برای آن در LangChain، یک بازیاب را با استفاده از داده‌هایی که در بخش قبل تکه تکه کردید پیاده کنید.

ب) در نهایت با دادن سه پرس و جو به بازیاب خود، عملکرد آن‌را با بررسی اسناد بازیابی شده بررسی کنید. یکی از پرس و جوها مربوط به مباحث کتاب مرجع، دیگری مربوط به علوم کامپیوتر و خارج از زمینه پردازش زبان‌های طبیعی (مانند درخت جستجوی دودویی) و دیگری به طور کلی خارج از حوزه عملکرد چت بات ما (مانند رئیس جمهور بولیوی کیست) باشد.

بخش سوم – پیاده‌سازی بازیاب ترکیبی (امتیازی)

در این بخش به پیاده سازی بازیاب مناسب برای اسناد مرتبط خواهیم پرداخت. برای این موضوع لازم است که از EnsembleRetriever در کتابخانه LangChain و BM۲۵Retriever و FAISS به عنوان بازیاب‌های Lexical و Semantic استفاده شود.

الف) تفاوت بین این دو نوع بازیاب را توضیح دهید. (لازم نیست به جزئیات عملکرد هر الگوریتم بپردازید، به صورت کلی تفاوت‌های اصلی دو گروه روش بازیابی در چیست و اهمیت هرکدام در چه نوع کاربردی مشخص می‌شود)

ب) با استفاده از کلاس‌های معرفی شده، بازیاب ترکیبی را پیاده‌سازی کنید.

پ) مقادیر مناسب برای ضریب تأثیر هر نوع بازیاب را در بازیاب نهایی تعیین کنید. برای این منظور می‌توانید از چند مثال استفاده کنید و تأثیر در اسناد بازیابی شده را با مقادیر ضریب تأثیر افراطی برای هر نوع بازیاب بررسی کنید.

راهنمایی: به طور عمومی، در زبان‌هایی مانند انگلیسی که Embedder ها از قدرت بالایی برخوردار هستند، وزن بیشتر برای بازیاب Semantic عموما به عملکرد نهایی بهتر منجر می‌شود. نیاز نیست برای این بخش زمان زیادی اختصاص دهید، گرفتن حس از تأثیر ضرایب در عملکرد نهایی بازیاب مدنظر است.

ت) در نهایت با دادن سه پرس و جو به بازیاب خود با تنظیمات نهایی، عملکرد آن‌را با بررسی اسناد بازیابی شده بررسی کنید. یکی از پرس و جوها مربوط به مباحث کتاب مرجع، دیگری مربوط به علوم کامپیوتر و خارج از زمینه پردازش زبان‌های طبیعی (مانند درخت جستجوی دودویی) و دیگری به طور کلی خارج از حوزه عملکرد چت بات ما (مانند رئیس جمهور بولیوی کیست) باشد.

راهنمایی: در نظر داشته باشید که سرعت عملکرد بازیاب‌ها، زمانی که از GPU استفاده می‌کنید به شدت افزایش می‌یابد.

(در این بخش با توجه به ابعاد تمرین و درس، به نوع سوم بازیاب‌ها که از خانواده ColBERT هستند اشاره نشد. این خانواده نیز ویژگی‌های جذابی در خود دارند 😊)

بخش راهنما – راهنمای کلی پیاده‌سازی زنجیرها[۴]

زنجیرها یک مفهوم منطقی در LangChain هستند که همانطور که از اسمشان پیداست، به کمک یک گراف جهت‌دار بدون دور، خروجی هر بخش را به ورودی بخش بعد متصل می‌کنند. به طور کلی زنجیرهای مورد استفاده ما سه بخش اصلی دارند:

۱. ورودی مدل‌زبانی بزرگ که معمولا یک رشته معمولی است که تعدادی متغیر در آن در نظر گرفته می‌شود. برای مثال:

```
[6]  sample_prompt_template = (
         "You are a firendly chatbot, Here is the user query: \n"
         "{query}"
     )
```

شکل ۱. مثال پرامپت

پرس‌وجوی کاربر به عنوان ورودی زنجیر ما باید در محل قرار داده شده برای متغیر query قرار گیرد. برای این منظور و رفتن به دنیای استاندارد LangChain باید از ChatPromptTemplate استفاده کنیم. در مثال زیر رشته‌ی "Salam Bacheha" را در پرامپت مربوطه قرار داده‌ایم:

```
from langchain.prompts import ChatPromptTemplate

sample_prompt = ChatPromptTemplate.from_template(sample_prompt_template)

print(sample_prompt.invoke({"query":"Salam Bacheha!"}))
```

```
messages=[HumanMessage(content='You are a firendly chatbot, Here is the user query: \nSalam Bacheha!')]
```

شکل ۲. پرامپت مثال با قرار گرفتن پرس‌وجو

۲. خود مدل‌های زبانی که پرامپت را به عنوان ورودی گرفته و خروجی را تولید می‌کنند.

[۴] chains

۳. پس پردازش‌گر [۵] که وظیفه ارزیابی خروجی مدل زبانی بزرگ را دارد. در طی این تمرین ما از PydanticOutputParser استفاده خواهیم کرد. عملکرد این مدل به این صورت است که یک کلاس را با ساختار دلخواهمان تعریف می‌کنیم و خروجی مدل زبانی باید از آن ساختار پیروی کند.

```
class SampleParser(BaseModel):
  tool_name: Literal["Value1", "Value2", "Value3"] = Field(
      description="we have a sample parser that only has three valid values, Value1, Value2 and Value3")

sample_parser = PydanticOutputParser(pydantic_object=SampleParser)
```

شکل ۳. مثال استفاده از PydanticOutputParser

با استفاده از این سه بلوک، زنجیر نهایی ما به صورت زیر تعریف خواهد شد:

```
sample_chain = sample_prompt | sample_llm | sample_parser
```

بخش چهارم – پیاده‌سازی ROUTER CHAIN

الف) با مراجعه به سایت TogetherAI و ثبت نام، API Key رایگان خود را دریافت کنید. این سایت تا ۵ دلار به شما اعتبار اولیه رایگان داده و امکان استفاده رایگان از تمام مدل‌های مطرح وزن باز یا متن باز برای شما فراهم خواهد بود.

ب) مدل خود را با استفاده از کلاس ChatTogether در LangChain روی مدل meta-llama/Llama-۳-۷۰b-chat-hf با میزان Temperature برابر صفر تعریف کنید.

پ) حال باید یک Chain تعریف کنید که پرس‌وجوی کاربر را به عنوان ورودی گرفته و ابزار مورد استفاده را به عنوان خروجی انتخاب کند. وظیفه این زنجیر آن است که با گرفتن ورودی کاربر و به کمک prompt زدن به llm، تعیین کند که این پرس‌وجو مربوط به پردازش زبان طبیعی، علوم کامپیوتر و یا خارج از حوزه کاری چت بات ماست. این زنجیر شامل سه بخش است:

۱. بخش مربوط به تولید پرامپت ورودی که باید با استفاده از ChatPromptTemplate پیاده شود.

---

[۵] parser

۲. بخش مربوط به مدل زبانی بزرگ که در بخش ب تعریف شد.

۳. بخش مربوط به پردازش خروجی مدل زبانی بزرگ که در بخش ت پیاده خواهد شد.

ورودی را به کمک ChatPromptTemplate پیاده کرده و به مدل تعریف شده در بخش ب متصل کنید.

ت) در این مرحله Parser ای بنویسید که با استفاده از کلاس‌های Pydantic ، خروجی llm را ارزیابی کند و مطمئن شود که یکی از مقادیر VectorStore ، SearchEngine یا None است. ارزیاب نوشته شده را به زنجیر اضافه کرده و نام این زنجیر را router_chain در نظر بگیرید.

ث) در مورد اینکه چرا برای این کاربرد میزان ۰ را برای temperature استفاده کرده‌ایم توضیح دهید.

(قابلیت تصمیم‌گیری و استفاده از ابزارهای مختلف توسط مدل زبانی بزرگ، Function Calling نام دارد که توسط خود Together نیز روی سه مدل عرضه می‌شود. با بررسی‌های انجام شده، این مدل‌ها کیفیت لازم را ندارند و به همین منظور، این روش برای انجام کار انتخاب شد. با فراگیری این روش دیگر محدود به پشتیبانی مدل زبانی بزرگ از Function Calling نخواهید بود و می‌توانید از هر مدلی استفاده کنید. لازم به ذکر است که Function Calling کاربردها و ویژگی‌های دیگری نیز دارد که به دلیل ایجاد پیچیدگی از پیاده‌سازی دستی آن در این تمرین خودداری شد. بررسی این موارد می‌تواند لبه بسیار ارزشمندی به شما در پژوهش و صنعت بدهد 😊)

بخش پنجم – پیاده‌سازی SEARCH ENGINE CHAIN

الف) با ثبت نام در پلتفرم Tavily، API Key رایگان خود را دریافت کنید. این ابزار، ابزاری آسان برای بازیابی اسناد برای استفاده در LLM هاست. این پلتفرم به عنوان SearchEngine ما مورد استفاده قرار خواهد گرفت.

ب) با تعریف این ابزار در فضای LangChain، یک پرس‌وجو را به عنوان ورودی بدهید و خروجی را جهت آشنایی با شکل خروجی بررسی کنید.

ت) حال باید یک زنجیر بنویسید که پرس‌وجوی کاربر را به عنوان ورودی بگیرد و حداکثر ۵ سند مرتبط را بازیابی کند. هر سند بازیابی شده باید محتوای سند را به عنوان page_content و url سایت مربوطه را به عنوان metadata تحت تعریف Document های LangChain داشته باشد. برای تعریف اسناد از این کلاس استفاده کنید. نام این زنجیر را search_engine_chain در نظر بگیرید. این زنجیر دو بخش اصلی خواهد داشت:

۱. پلتفرم جستجوی Tavily

۲. پس پردازش‌گر با هدف تبدیل داده‌های خروجی جستجو به Document های استاندارد LangChain

بخش ششم – پیاده‌سازی RELEVANCY CHECK CHAIN (امتیازی)

هدف این بخش پیاده‌سازی یک زنجیر به منظور بررسی مرتبط بودن سند به پرس‌وجوی کاربر است. ورودی این زنجیر یک سند به همراه پرس‌وجوی کاربر بوده و خروجی آن یکی از کلمات relevant و یا irrelevant خواهد بود. سند ورودی به این زنجیر می‌تواند خروجی پایگاه داده برداری و یا موتور جست‌وجوی ما باشد و به همین منظور در بخش قبل خروجی‌های موتور جست‌وجو را به فرم استاندارد LangChain در آوردیم. زنجیر نهایی این بخش هم شامل سه بخش پرامپت ، مدل زبانی بزرگ و پس پردازش‌گر مربوطه خواهد بود.

الف) پرامپتی برای ارزیابی ارتباط طراحی کنید. در این پرامپت به مدل زبانی بزرگ بگویید که یک پرس‌وجو و یک سند به او داده شده است و وظیفه دارد که مرتبط بودن یا نبودن سند به پرس‌وجو را مورد بررسی قرار دهد. خروجی‌های مجاز برای مدل عبارت‌های relevant و irrelevant خواهند بود.

ب) مدل را با استفاده از ChatTogether تعریف کنید.

پ) پس پردازشگر مربوطه را برای پردازش خروجی مدل زبانی بزرگ تعریف کنید.

ب) به نظر شما چرا به چنین زنجیری نیاز داریم؟ با یک مثال بیان کنید.

نام این زنجیر را relevancy_check_chain در نظر بگیرید.

بخش هفتم – پیاده‌سازی FALLBACK CHAIN

Router ما سه عنوان مجاز برای خروجی داست. VectorStore، SearchEngine و Fallback (None). دو مورد قبل را در بخش‌های قبلی پیاده کردیم و تنها مورد آخر باقی مانده است. زنجیر پیاده شده در این بخش سابقه چت کاربر را در کنار پرس و جوی او به عنوان ورودی دارد. ورودی‌ها در این زنجیر به مدل زبانی بزرگ داده خواهند شد و با پرامپت مناسب، از مدل خواسته می‌شود تا عدم توانایی خود را در پاسخگویی به این سوال کاربر ابراز کند.

الف) یک پرامپت آماده کنید که حیطه کاربرد مدل زبانی بزرگ برای آن مشخص می‌شود و متغیرهایی برای سابقه گفت‌وگو و پرس‌وجوی کاربر در نظر گرفته شده است. در نهایت از مدل بخواهید که پاسخ مناسب را با در نظر گرفتن محدودیت‌هایش تولید کند.

ب) یک مدل زبانی با استفاده از تنظیمات Together تعریف کنید و پرامپت را به آن وصل کنید. می‌توانید مقادیر بالاتر temperature را برای مدل مورد ارزیابی قرار دهید.

پ) در نهایت خروجی مدل را به یک StrOutputParser وصل کنید.

راهنمایی: تبدیلات مربوط به تبدیل سابقه گفت‌وگو از چارچوب LangChain به متن، در کارگاه پیاده سازی خواهد شد و می‌توانید از همان منطق استفاده کنید.

نام این زنجیر را fallback_chain در نظر بگیرید.

بخش هشتم – پیاده‌سازی GENERATE WITH CONTEXT CHAIN

این آخرین زنجیر ماست. وظیفه این زنجیر گرفتن پرس‌وجوی کاربر به همراه اسناد مرتبط به عنوان ورودی است و پاسخ نهایی را به عنوان خروجی خواهد داشت.

الف) پرامپت این بخش شامل توضیحاتی در این مورد است که سوال و یک مجموعه از جواب‌های محتمل به مدل داده شده است و مدل باید پاسخ را با استفاده از اسناد داده شده تولید کند.

ب) مدل همانند بخش‌های قبل است.

پ) پس پردازش‌گر نیز StrOutputParser خواهد بود.

نام این زنجیر را generate_with_context_chain در نظر بگیرید.

بخش نهم – آماده سازی گراف با استفاده از LANGGRAPH

حال تمام گره‌ها و زنجیرهای لازم را برای استفاده از ابزار LangGraph داریم.

State گراف ما با تعریف زیر خواهد بود:

```
class AgentSate(TypedDict):
    """The dictionary keeps track of the data required by the various
nodes in the graph"""

    query: str
    chat_history:list[BaseMessage]
    generation: str
    documents: list[Document]
```

گراف ما گره‌های زیر را با اهداف تعریف شده خواهد داشت:

الف) router_node : این گره، گره شروع کار گراف است و پرس‌وجوی کاربر را به عنوان ورودی گرفته و یکی از سه یال خروجی (ابزارهای در دسترس) را با استفاده از router_chain انتخاب می‌کند.

ب) vector_store : این گره وظیفه بازیابی اسناد مرتبط با پرس‌وجو را به کمک retriever_chain دارد.

پ) search_engine: این گره پرس‌وجوی کاربر را از state گرفته و اسناد مرتبط را با استفاده از search_engine_chain بازمی‌گرداند.

ت) filter_docs (امتیازی): این گره وظیفه بررسی ارتباط اسناد بازگردانده شده از یکی از دو گره بخش ب و پ را با پرس‌وجوی کاربر دارد. در این گره ارتباط بین پرس‌وجو از state با تمام اسناد موجود در state سنجیده و یک لیست فیلترشده از اسناد مرتبط را به عنوان documents بازمی‌گرداند. این کار به کمک check_relevancy_chain انجام خواهد شد.

ث) fallback : این گره سابقه چت را در کنار پرس‌وجوی کاربر از state برداشته و با استفاده از fallback_chain پاسخ نهایی را برای کاربر تولید کرده و به پایان گراف متصل است.

ج) generate_with_context : وظیفه این گره، تولید خروجی نهایی به کمک اسناد مرتبط است. تنها ورودی این گره، گره filter_docs بوده و از رنجیر generate_with_context_chain استفاده می‌کند. در صورت انجام ندادن بخش‌های امتیازی، vector_store و search_engine به این گره دسترسی مستقیم خواهند داشت.

روند کار گراف به صورت زیر خواهد بود:

۱. کاربر پرس‌وجوی خود را به گراف می‌دهد. ورودی گراف، گره مربوط به تصمیم‌گیری (router_node) خواهد بود.
۲. router_node سه یال به سه گره search_engine ، vector_store و fallback دارد. با توجه به خروجی این گره، یکی از یال‌ها فعال شده و گره مرتبط به آن انتخاب خواهد شد. یکی از سه حالت زیر را خواهیم داشت:

حالت اول) خروجی روتر SearchEngine باشد: گره search_engine فعال می‌شود و خروجی‌های آن وارد گره مربوط به شناسایی ارتباط اسناد به نام filter_docs می‌شود. اگر خروجی filter_docs یک لیست خالی نبود، گره generate_with_context فراخوانی می‌شود و کار ما تمام خواهد شد.

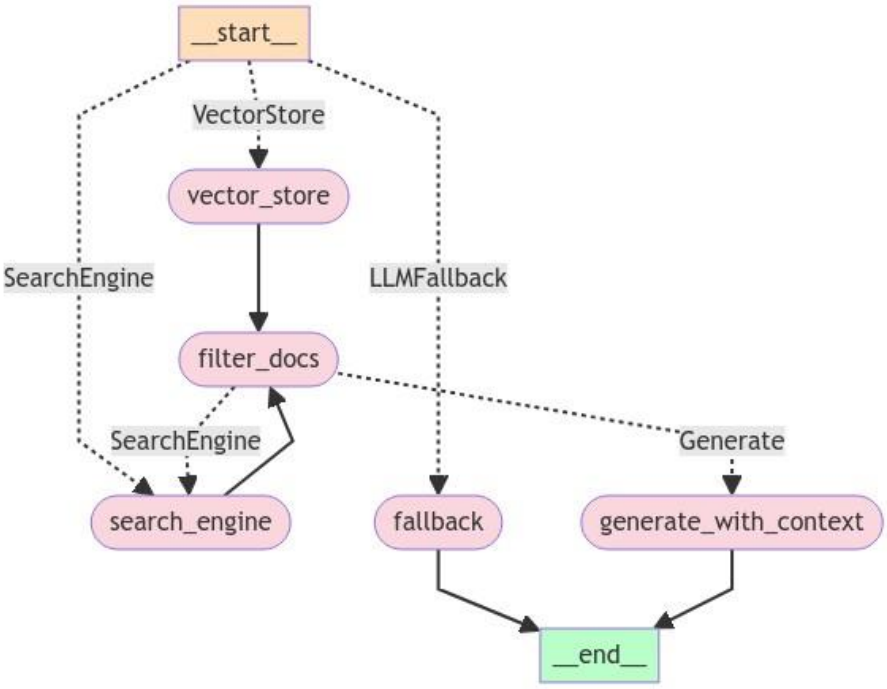
در صورتی که بخش امتیازی مربوط به گره filter_docs را انجام نداده‌اید، اسناد خروجی این بخش به طور مستقیم به گره generate_with_context فرستاده می‌شوند.

حالت دوم) خروجی روتر VectorStore باشد: در این حالت اسناد مرتبط از پایگاه داده محلی بازیابی شده و به filter_docs داده می‌شوند. اگر لیست خروجی خالی بود گره search_engine فراخوانی شده و اگر خالی نبود گره generate_with_context فراخوانی خواهد شد و به انتهای روند کار گراف می‌رسیم.

در این بخش نیز مانند حالت اول، انجام بخش امتیازی دخیل خواهد بود.

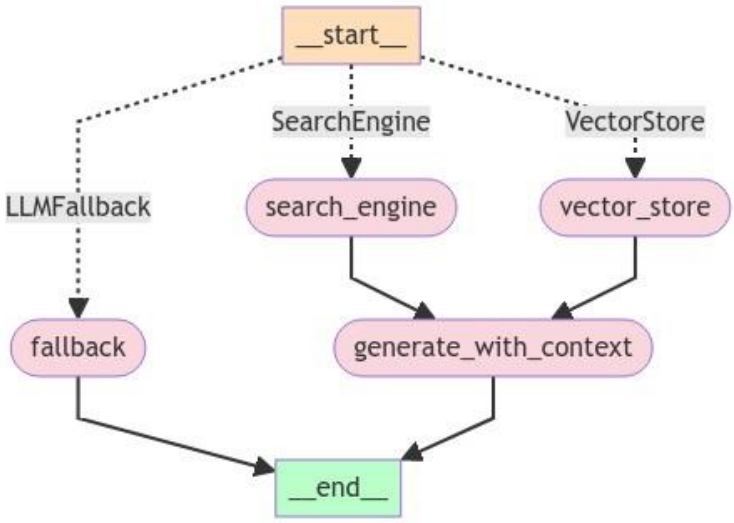
حالت سوم) خروجی روتر Fallback باشد: در این حالت مستقیما گره fallback فراخوانی خواهد شد و روند گفتگوی ما با چت بات به پایان خواهد رسید.

خروجی گراف تولیدی باید به شکل صفحه بعد باشد:



شکل ۴. گراف نهایی مورد انتظار با بخش‌های امتیازی

در صورتی که بخش‌های امتیازی را انجام نداده‌اید، خروجی گراف شما باید به شکل زیر باشد:



شکل ۵. گراف نهایی مورد انتظار بدون بخش‌های امتیازی

اگر اینجایید کار مهمی انجام دادید 😊 برای چت بات خودتون یه اسم بذارید و از صحبت کردن باهاش لذت ببرید!

ملاحظات (حتما مطالعه شود)

تمامی نتایج شما باید در یک فایل فشرده با عنوان NLP_CA۶_StudentID تحویل داده شود.

- خوانایی و دقت بررسی‌ها در گزارش نهایی از اهمیت ویژه‌ای برخوردار است. به تمرین‌هایی که به صورت کاغذی تحویل داده شوند یا به صورت عکس در سایت بارگذاری شوند، ترتیب اثری داده نخواهد شد.
- کدهای نوشته شده برای هر بخش را با نام مناسب مشخص کرده و به همراه گزارش تکلیف ارسال کنید. همه‌ی کدهای پیوست گزارش بایستی قابلیت اجرای مجدد داشته باشند. در صورتی که برای اجرا مجدد آنها نیاز به تنظیمات خاصی می‌باشد بایستی تنظیمات مورد نیاز را نیز در گزارش خود ذکر کنید.
- این تمرین امکان ارسال با تاخیر نخواهد داشت.
- توجه کنید این تمرین باید به صورت تک نفره انجام شود و پاسخ‌های ارئه شده باید نتیجه فعالیت فرد نویسنده باشد (همفکری و به اتفاق هم نوشتن تمرین نیز ممنوع است). در صورت مشاهده تشابه به همه افراد مشارکت کننده، نمره تمرین صفر و به استاد نیز گزارش می‌گردد.
- در صورت بروز هرگونه مشکل با ایمیل زیر در ارتباط باشید:

mailto:vahyd@live.com

وحید رحیم زاده

مهلت تحویل: ۱۹ تیرماه ۱۴۰۳