llm-chat_demo

January 9, 2024

1 LLM Developement

1.1 Einfache Chat Completions mit OpenAI

OpenAI-Dokumentation https://platform.openai.com/docs/quickstart?context=python

[ ]:
```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "user", "content": "Schreibe ein Gedicht über einen␣
 ↪Softwareentwickler und seinen KI-Copiloten."}
  ]
)

print(completion.choices[0].message)
```

1.2 Langchain

Langchain ist ein Framework für Python und JavaScript zum entwickeln von Anwendungen die LLMs nutzen.

- Vereinheitlichung von Prompting und OutputParsing unabhängig des Sprachmodelles
- Einfache Definition von Chains zur Integration von Retrievern und Agenten
- Anknüfungspunkt zu Tools für das Deployment, Debugging und Überwachen von LLM-Applikationen

Langchain-Dokumentation https://python.langchain.com/docs/get_started/quickstart

1.3 Import der benötigten Packages

[ ]:
```
import os

from langchain_mistralai.chat_models import ChatMistralAI
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
```

```
from langchain_core.runnables import ConfigurableField
from operator import itemgetter
from langchain.prompts import HumanMessagePromptTemplate
```

1.4 Model Initialisierung mit Fallback

Ausfallsicherheit bewerkstelligen, mithilfe von Fallbacks. Wenn chat_openai nicht erreichbar ist wird stattdessen chat_mistral verwendet.

MistralAI https://mistral.ai/

[ ]:
```
chat_openai = ChatOpenAI(model="gpt-3.5-turbo")
chat_mistal = ChatMistralAI(model="mistral-small")

model = (
    chat_openai
    .with_fallbacks([chat_mistal])
)
```

1.5 Langchain Templates für Prompts

Erzeugung von Prompt-Templates mit Platzhaltern, um unabhängig der Nutzereingabe bestimmtes Verhalten zu erzwingen.

[ ]:
```
template = """
Verfügbare Kurse:
- Digital Trainer
- Fit fürs Studium
- KI im Alltag

Empfehle einen Kurs basierend auf der vorangegangenen Liste von Kursen zum␣
 ↪thema {topic}.
"""
prompt = ChatPromptTemplate.from_template(template)
```

1.6 Langchain Expression Language für das einfache Erstellen von LLM-Aktionen

Definition einer Kette von Aktionen. Ergebniss des Prompts wird an das Modell übergeben, das Ergebnis des Modells an den OutputParser.

[ ]:
```
chain = (
    prompt
    | model
    | StrOutputParser()
)
```

1.7 Chain ausführen

```
[ ]: user_input = input("Für welches Thema interessieren Sie sich?")
     chain.invoke({"topic": user_input})
```

1.8 RAG - Retrieval Augmented Generation

Sodass echte FLL Kurse vorgeschlagen werden und Halluzinationen verringert werden, soll dem Modell eine dynamische Wissensbasis zur Verfügung gestellt werden, um darauf basierend Antworten zu generieren. Dazu benötigen wir einen Retriever. Ein Objekt, das auf Basis eines Inputs und einer Datengrundlage, zum Beispiel eine Datenbank oder einen Knowledge Graph, einen passenden Kontext generiert, der in den Prompt integriert werden kann.

Für diesen Fall, wollen wir eine Vektordatenbank mit den FLL-Kursen befüllen. Eine Vektordatenbank ist darauf optimiert, mehrdimensionale Vektoren zu speichern und durchsuchbar zu machen. Auf Basis der Berechnung der Nähe von Vektoren im mehrdimensionalen Raum, können so semantisch ähnliche Dokuemente identifiziert werden. Zur Erzeugung von Vektoren, die die Kurse darstellen, benötigen wir eine Spezialisiertes Sprachmodell. Diese Vektoren werden auch Embeddings genannt.

1.9 Inititlaisiere Embedding Funktion

Als Embedding-Modell nutzen wir hier ein das Modell Instructor-Large, das über den HugginFace-Hub bereitsgestellt wird und lokal auf unserem Gerät ausgeführt werden kann. Für die Kompatibilität mit Langchain benutzen wir dazu das HuggingFaceInstructEmbeddings-Package.

```
[ ]: from langchain_community.embeddings import HuggingFaceInstructEmbeddings
     embedding_function = HuggingFaceInstructEmbeddings(
         # model_name="hkunlp/instructor-large",
         query_instruction="Represent the course for retrieval: "
     )
```

1.10 Import der zuvor heruntergeladenen Kurse im csv Format

```
[ ]: # get array of courses from courses.csv with columns name,description,url
     import csv

     courses = []
     with open("local/courses.csv", newline="", encoding="utf-8") as csvfile:
         reader = csv.DictReader(csvfile)
         for row in reader:
             courses.append(row)
```

1.11 Initialisierung einer Vektordatenbank, in der die Kurse als Documents gespeichert werden

Wir nutzen in diesem Beispiel ChromaDB. Es gibt aber auch viele andere Alternativen wie FAISS, MongoDB Atlas, Pinecone, Redis, Postgres Embedding…

ChromaDB-Dokumentation https://github.com/chroma-core/chroma

[ ]:
```python
from langchain.docstore.document import Document

# Pro kurs wird ein Document erstellt. Dieses enthält Metadaten sowie einen␣
 ↪page_content.
# Der Inhalt von page_content wird embedded und so für die sucher verwendet.
docs = []
for course in courses:
    # Remove html from description
    import re
    course["description"] = re.sub("<[^<]+?>", "", course["description"])
    # Create document.
    doc = Document(
        page_content=course["name"] + " " + course["description"],
        metadata={
            "name": course["name"],
            "description": course["description"],
            "url": course["url"],
        },
    )
    docs.append(doc)

# Create a vector store from the documents with the predefined embedding␣
 ↪function.
from langchain_community.vectorstores import Chroma
db = Chroma.from_documents(docs, embedding_function)
# Retriever will search for the top_5 most similar documents to the query.
retriever = db.as_retriever(search_kwargs={"k": 5})
```

1.12 Visualisierung der eingebetteten Kurs-Dokumente

[ ]:
```python
# Run pip install git+https://github.com/wyatt/chromaviz/ to install chromaviz␣
 ↪or skipt this step.
from chromaviz import visualize_collection
visualize_collection(db._collection)
```

1.13 Anspassung des Templates für RAG

Die hart codierten Kurse ersetzen wir durche einen Platzhalter, der durch die Ergebnisse des Retrievers, mit zum thema passenden Kursen ausgefüllt wird.

[ ]:
```python
# Complete RAG Chain
template = """
Verfügbare Kurse:
{context}

Empfehle einen oder bis zu 3 Kurse basierend auf der vorangegangenen Liste von␣
 ↪Kursen, die gut zum Thema {topic} passen.
```

```
Wenn möglich biete Links zu den Kursen an, über die Nutzende die Kurse␣
 ↪erreichen können.
"""
prompt = ChatPromptTemplate.from_template(template)
```

1.14 Alternative Erstellung des Prompts mit System-Message

[ ]:
```
from langchain.prompts import HumanMessagePromptTemplate
from langchain_core.messages import SystemMessage

prompt = ChatPromptTemplate.from_messages(
    [
        SystemMessage(
            content=(
                "Du bist ein hilfreicher Assistent der Nutzenden dabei␣
 ↪unterstützt, passende Kurse auf der Kursplattform FututreLearnLab zu finden.␣
 ↪Du kannst auf Fragen antworten und Empfehlungen aussprechen."
                "Antworte immer auf deutsch. Wenn keine gut passenden Kurse␣
 ↪gefunden werden können, dann gib eine entsprechende Antwort aus."
            )
        ),
        HumanMessagePromptTemplate.from_template(template),
    ]
)
```

1.15 Ergänzung der Chain um weitere Inputwerte

[ ]:
```
chain = (
    {"context": retriever, "topic": RunnablePassthrough()}
    | prompt
    | model
    | StrOutputParser()
)
```

1.16 Chain ausführen

[ ]:
```
user_input = input("Für welches Thema interessieren Sie sich?")
chain.invoke(user_input)
```

1.16.1 Glückwunsch!

Wenn alles geklappt hat solltest du nun individualisierte Kursempfehlungen auf Basis der echten FLL-Kurse erhalten. Wenn du noch Lust hast experimentiere etwas mit der Chain! Füge eine andere Datenbasis ein, passe den Prompt an oder baue einen kompletten Chatbot mit Interface und allem drum und dran. Oder noch besser, lass das alles deinen Assistenten tun.

Weiterführende Links:

- Erstelle und deploye eigene AI/ML-Apps mit Gradio und HuggingFace
- Die besten Open-Source LLMs https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
- Installiere LLMs lokal mit Ollama oder LM-Studio
- Werde selbst zum Coding-Assistenten mit GPT-Pilot
- Multi-Agenten Systeme entwickeln mit autogen oder crew.ai