Dokumentation des RAG-Systems

Architektur

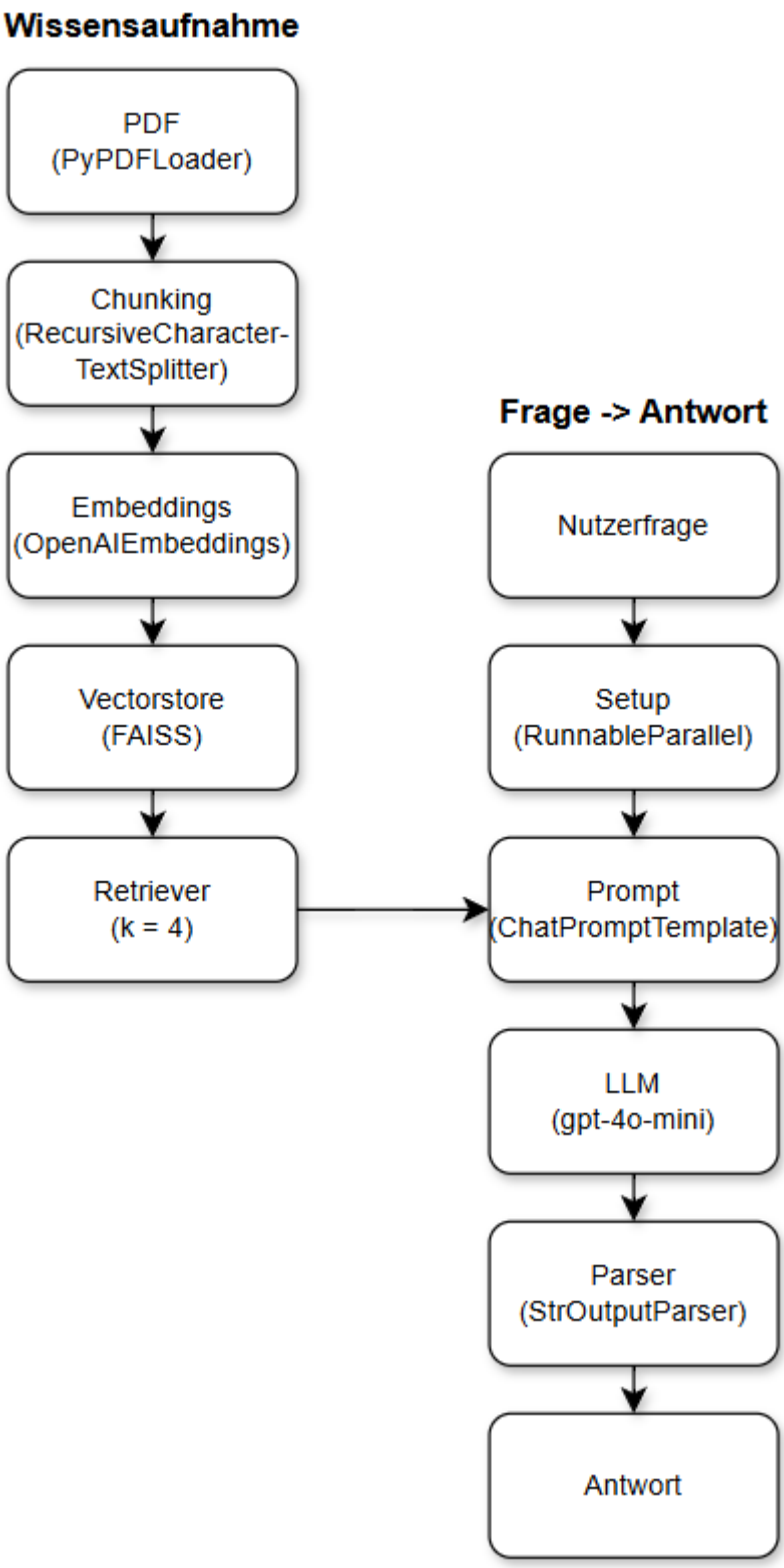
Die Architektur des RAG-Systems lässt sich grob in 2 Bereiche aufteilen.

In der Wissensaufnahme wird zunächst die PDF durch den PyPDFLoader geladen. Anschließend werden die Seiten durch den RecursiveCharacterTextSplitter in überlappende Chunks geteilt. Durch OpenAIEmbeddings (text-embedding-3-small) werden die Chunks in Vektoren umgewandelt. Der FAISS Vectorstore indexiert dies dann für eine schnelle Ähnlichkeitssuche. Der Retriever kapselt die Suche und liefert zu einer Frage die ähnlichsten Chunks. In diesem RAG-Systems sind es die 4 ähnlichsten.

In dem Bereich von der Nutzerfrage bis hin zur Antwort wird zunächst in einem Setup (RunnableParallel) die Eingabe in zwei Stränge aufgeteilt: Context bezieht der Retriever aus dem FAISS-Index, und question wird via RunnablePassthrough aus der Eingabe unverändert weitergereicht. Diese beiden Werte füllt das ChatPromptTemplate in den Prompt ein. Anschließend erzeugt das LLM die Antwort auf Basis des Prompts, und der StrOutputParser extrahiert daraus den reinen Text und liefert somit die Antwort.

Modellkonfiguration

Für die Antwortgenerierung wird gpt-4o-mini mit einer Temperature von 0,1 genutzt, um präzise und wenig zufällige Antworten zu erhalten. Die semantische Suche basiert auf text-embedding-3-small und die Wissensquelle wird per PyPDFLoader eingelesen und auf maximal 10 Seiten begrenzt. Der Inhalt wird mit Chunk Size 800 und Chunk Overlap 120 zu überlappenden Textstücken segmentiert, damit Zusammenhänge erhalten bleiben. Die Einbettungen werden in einem FAISS-Vectorstore indexiert, und der Retriever liefert für jede Frage die vier passendsten Chunks (k=4), die anschließend in den Prompt gegeben werden.


Wissensaufnahme
PDF
(PyPDFLoader)
Chunking
(RecursiveCharacter-
TextSplitter)
Embeddings
(OpenAIEmbeddings)
Vectorstore
(FAISS)
Retriever
(k = 4)
Frage -> Antwort
Nutzerfrage
Setup
(RunnableParallel)
Prompt
(ChatPromptTemplate)
LLM
(gpt-4o-mini)
Parser
(StrOutputParser)
Antwort