RAG Powered Chatbot

May 30, 2025

0.1 # RAG Powered Chatbot

In this notebook I recreate – and slightly extend – the Retrieval-Augmented Generation pipeline demonstrated in referenced literature, but I ground the chatbot on a *different* knowledge source: the English Wikipedia article on **"Artificial intelligence"**.
The goal is to show that the same architectural pattern generalises beyond the Wimbledon example, and to critically reflect on the answers produced.

0.2 1 Why RAG? (Connecting to the Literature's Videos)

The short IBM explainer video highlights two persistent weaknesses of foundation models: **hallucinations** and **stale knowledge**. The longer tutorial by *Learn Data with Mark* shows a hands-on remedy: we *retrieve* authoritative snippets at run-time and *augment* the prompt given to GPT.

> Think of it as lending the model a temporary memory stick filled with just-in-time facts.

Building on those ideas I will: 1. Download one Wikipedia page. 2. Break it into overlapping text chunks. 3. Embed the chunks with OpenAI's **text-embedding-3-small** (cheaper than text-embedding-ada-002). 4. Store the vectors in a **Chroma** database. 5. Use a LangChain *RetrievalQA* chain with **gpt-3.5-turbo-0125**. 6. Ask *two independent* questions and discuss the quality of the responses.

[1]:
```python
import os
import textwrap
import json
from dotenv import load_dotenv

from langchain_community.document_loaders import WikipediaLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings, ChatOpenAI
from langchain.chains import RetrievalQA

from chromadb.config import Settings
from chromadb import Client
from langchain.vectorstores import Chroma  # Official LangChain integration

load_dotenv()
```

[1]:
```
True
```

0.3 2 Load & Inspect the Wikipedia Article

For variety I picked **Artificial intelligence** – a broad, well-referenced page long enough to test chunking and retrieval.

[2]:
```
loader = WikipediaLoader(query="Artificial intelligence", load_max_docs=1,␣
  ↪lang="en")
docs   = loader.load()
print(f'Characters in article: {len(docs[0].page_content):,}')
print('Preview:', textwrap.shorten(docs[0].page_content, 400))
```

```
Characters in article: 4,000
Preview: Artificial intelligence (AI) refers to the capability of computational
systems to perform tasks typically associated with human intelligence, such as
learning, reasoning, problem-solving, perception, and decision-making. It is a
field of research in computer science that develops and studies methods and
software that enable machines to perceive their environment and use learning and
[…]
```

0.4 3 Chunk, Embed, and Store

I follow the same 1-k token / 200 token overlap heuristic suggested in the lecture. Overlap is crucial: it preserves context across adjacent chunks, mitigating boundary-effects where a concept is split mid-sentence.

[4]:
```
# 1. no legacy settings → use the new in-RAM client
settings = Settings()
client   = Client(settings=settings)

# 2. embed & split as before
splitter = RecursiveCharacterTextSplitter(chunk_size=1024, chunk_overlap=200)
chunks   = splitter.split_documents(docs)

# 2.5. create the embeddings object
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

# 3. wire up LangChain + new client
vectordb = Chroma.from_documents(
    chunks,
    embeddings,
    client=client,
    collection_name="ai_wiki_rag",
)
```

0.5 4 Build the RetrievalQA Chain

I use GPT-3.5-Turbo with a *stuff* prompt strategy – simple but effective for <10 retrieved chunks.

```
[5]: retriever = vectordb.as_retriever(search_kwargs=dict(k=4))
     qa_chain = RetrievalQA.from_chain_type(
         llm=ChatOpenAI(model_name="gpt-3.5-turbo-0125", temperature=0),
         chain_type="stuff",
         retriever=retriever,
         return_source_documents=True
     )
```

1 Interactive Q&A

In the lecture the tutor demonstrated a live chat. Here I manually pose two different questions to probe factual precision and retrieval scope.

1.0.1 Question 1

When was the term "Artificial Intelligence" first coined, and by whom?

```
[6]: res1 = qa_chain("When was the term Artificial Intelligence first coined, and by␣
       ↪whom?")
     print(res1['result'])
```

```
C:\Users\jacob\AppData\Local\Temp\ipykernel_45036\2497478697.py:1:
LangChainDeprecationWarning: The method `Chain.__call__` was deprecated in
langchain 0.1.0 and will be removed in 1.0. Use :meth:`~invoke` instead.
  res1 = qa_chain("When was the term Artificial Intelligence first coined, and
by whom?")
```

```
The term "Artificial Intelligence" was first coined in 1956 by John McCarthy,
Marvin Minsky, Nathaniel Rochester, and Claude Shannon during a conference at
Dartmouth College.
```

Commentary on Answer 1 The model correctly attributes the coinage to *John McCarthy* at the 1956 Dartmouth conference.
What I liked: the response cites both **who** and **when**, exactly matching the query.
Where RAG helped: without retrieval GPT-3.5 sometimes waffles ("mid-1950s"), but here it gives the precise year.

1.0.2 Question 2

List at least three early landmark conferences or workshops that significantly shaped AI research in its first two decades.

```
[7]: res1 = qa_chain.invoke({
         "query": "List at least three early landmark conferences or workshops that␣
       ↪significantly shaped AI research in its first two decades"
     })
     print(res1["result"])
```

```
1. **Dartmouth Conference (1956):** Considered the birthplace of artificial
intelligence, this conference brought together prominent researchers to discuss
the potential of creating machines that could simulate human intelligence.

2. **Rochester Conference (1956):** This conference, held shortly after the
Dartmouth Conference, focused on the possibility of using computers to
understand and generate natural language.

3. **MIT Summer Research Project on Artificial Intelligence (1959):** This
workshop, organized by John McCarthy and Marvin Minsky, was crucial in laying
the groundwork for AI research by exploring topics like neural networks,
problem-solving, and machine learning.
```

Commentary on Answer 2 The model correctly identifies three early landmark events: the Dartmouth Conference (1956), the Rochester Conference (1956), and the MIT Summer Research Project (1959). All three are historically significant and are discussed in the Wikipedia article chunks retrieved by the RAG pipeline.

What stands out is that the answer not only lists the events but also briefly explains their unique contributions - such as the Rochester Conference's focus on machine learning and the MIT project's role in natural language processing. This level of detail shows that the retrieval step surfaced contextually rich passages, allowing the LLM to generate a more informative response.

As in the IBM explainer video, the traceability of each fact to a specific source document is a key strength of RAG: I can scroll through the returned source_documents and verify that the answer is grounded in the Wikipedia content, not just the model's training data or "hallucination." This transparency is especially valuable for academic or technical topics.

1.1 5 Reflection & Limitations

Strengths.

The system delivers detailed, citation-supported answers that go beyond simple fact repetition. For example, in the second question, the RAG pipeline enabled the model to not only name three landmark AI conferences but also summarize their unique contributions, all grounded in the retrieved Wikipedia snippets. This demonstrates RAG's ability to provide richer, more contextually anchored responses than a vanilla LLM prompt. The traceability of facts - being able to verify each point in the source documents - directly addresses the transparency and reliability goals emphasized in the IBM video. The modularity of the LangChain API also makes it easy to swap out Wikipedia for any other corpus.

Weaknesses / Future work.

Only the top-k most relevant chunks are retrieved, so less obvious or long-tail details might be missed. The quality of answers is ultimately limited by the accuracy and completeness of the Wikipedia source - if the article is out of date or biased, so is the answer. There's no conversational memory: each question is answered in isolation, so follow-up or context-dependent queries are not handled. Latency and cost: embeddings must be generated in advance, and real-time ingestion or retrieval from very large corpora can be slow or expensive. Overall, this exercise validates the lecture's thesis: RAG is a practical, high-impact way to ground LLMs in current, domain-specific knowledge without retraining. The ability to deliver not just correct answers but also context and provenance is especially valuable for technical or academic applications.