Week 3 Report: Learning Retrieval-Augmented Generation (RAG)

Understanding the RAG Pipeline

This week, I dove into the full Retrieval-Augmented Generation (RAG) pipeline, which is a clever way to make language models smarter by letting them "look up" information before answering. Instead of relying solely on what the model already knows, RAG helps the model pull in relevant facts from an external source to provide more accurate and up-to-date answers.

Here's how the pipeline works step-by-step:

1. **Chunking:** First, the source documents are broken down into smaller, manageable pieces called chunks. This makes it easier to find exactly the right bits of information later.
2. **Embedding:** Each chunk is then converted into a vector — think of it as a numerical summary that captures its meaning.
3. **Storing:** These vectors and their corresponding chunks are stored in a vector database, which is designed for quick and efficient searching.
4. **Retrieving:** When a user asks a question, the query is also turned into a vector, and the system searches the database to find the chunks most relevant to the question.
5. **Prompting:** Finally, the retrieved chunks are combined with the user's question to create a prompt that the language model uses to generate its answer.

This approach means the model's answers are grounded in real, retrievable information rather than just guesses.

How the System Fits Together: Architecture Overview

Imagine the system as a pipeline:

- It starts with raw documents, which go through chunking and embedding.
- The embeddings are stored in a vector database.
- When a question comes in, the system embeds the query and searches the database for relevant chunks.
- These chunks are then fed into the language model along with the question.

- The model generates an answer, often including references to the source chunks to show where the information came from.

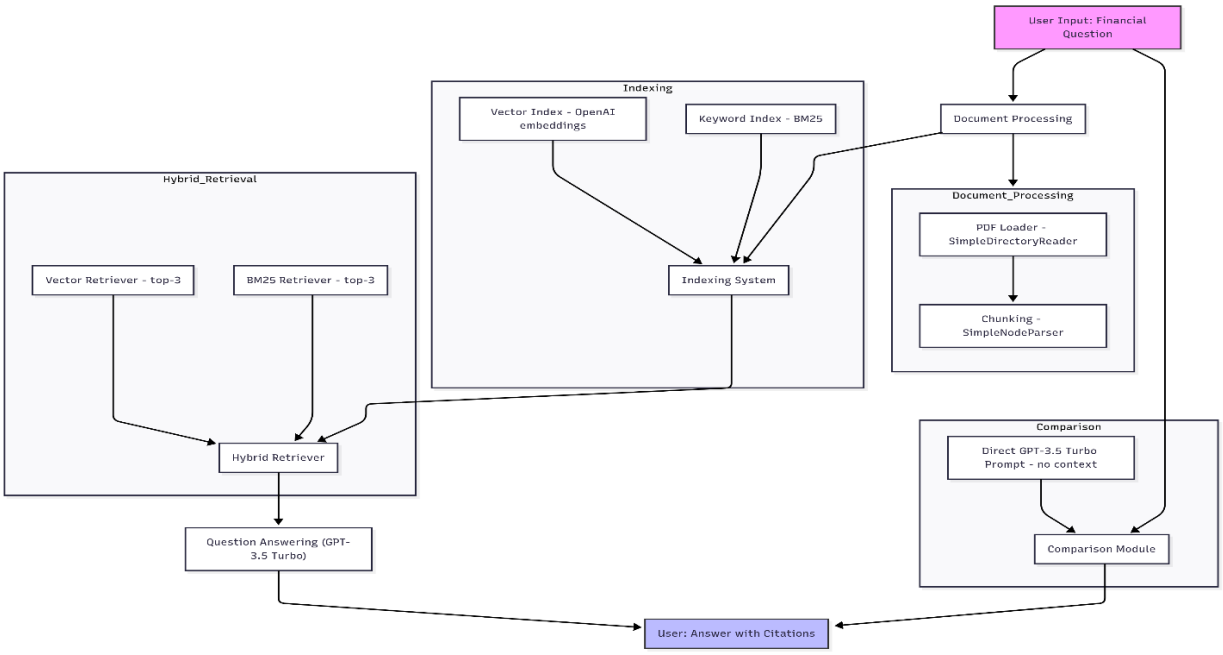
This flow ensures that the output is not only accurate but also transparent, since you can trace the answer back to its sources.

Building a Q&A Bot That Cites Its Sources

Using the RAG pipeline, I built a working question-and-answer Financial bot that doesn't just spit out answers but also tells you where it got the information. Here's the basic process I followed:

- I prepared a set of documents and split them into chunks.
- I created embeddings for each chunk and stored them in a vector database.
- When a user asks a question, the AI embeds the query and retrieves the most relevant chunks.
- It then constructs a prompt combining the user's question and the retrieved information.
- The language model generates an answer, and the bot includes citations pointing back to the original chunks.

This setup makes the bot's answers more trustworthy and grounded in actual data.



Comparing RAG with Pure Prompting

I also compared how well the bot performs when using retrieval-augmented prompts vs just relying on the language model's internal knowledge (pure prompting). Here's what I found:

- **Pure prompting** depends entirely on what the model learned during training. It can be quick and simple but sometimes gives outdated or inaccurate answers, especially on niche topics.
- **RAG**, on the other hand, enriches the model's responses with fresh, relevant information pulled from external documents. This leads to more precise and reliable answers, especially for specialized or recent information.

In short, retrieval-augmented prompting makes the model smarter and more trustworthy.

Trying Out Hybrid Search: BM25 Plus Vectors

To improve retrieval even further, I experimented with combining two search methods:

- **BM25**, a classic keyword-based search that's fast and good at finding exact matches.

- **Vector search**, which looks for semantic similarity, meaning it can find related concepts even if the exact words don't match.

By mixing both, the system benefits from the precision of keyword search and the flexibility of semantic search. This hybrid approach helped improve the relevance of retrieved documents, making the bot's answers even better.

Key Takeaways and Reflections

Working through RAG this week taught me a lot:

- The biggest strength of RAG is that it lets language models access external knowledge on the fly, which solves the problem of outdated or incomplete training data.
- Breaking documents into chunks and embedding them consistently is crucial for good retrieval.
- Retrieval-augmented prompts consistently outperform pure prompting, especially for detailed or domain-specific questions.
- Combining BM25 with vector search can significantly boost retrieval quality.
- Including source citations in answers builds user trust and makes the system's reasoning more transparent.
- Building a full RAG-powered Q&A bot requires careful integration of multiple components — chunking, embedding, storing, retrieving, and prompting — but the results are worth it.

Overall, this week's work showed me how retrieval-augmented generation can make AI assistants much more useful and reliable.