Material : **Generative AI**

Topic : **RAG (Retrieval Augmented Generation)**

Daniel
danielgenai77@gmail.com

Gen AI – RAG (Retrieval-Augmented Generation)

Gen AI – RAG (Retrieval-Augmented Generation)

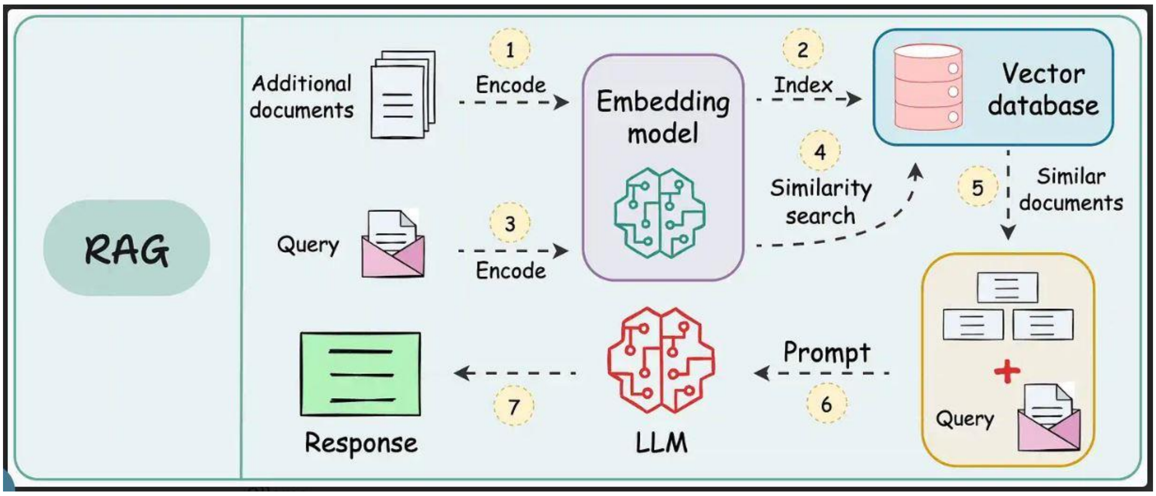
1. RAG

- ✓ **RAG** stands for Retrieval-Augmented Generation.
- ✓ RAG is an NLP approach that:
    - **Retrieves** relevant info from external sources.
    - **Augments** the model input with that info.
    - **Generates** more accurate and factual responses using a language model.
- ✓ It helps LLMs handle large or constantly changing knowledge bases more effectively.
- ✓ RAG in GenAI = **Search + Generate**.
    - It gives large language models "open-book access" to relevant information, making them smarter, safer, and more adaptable.

2. Why RAG?

- ✓ LLMs often lack up-to-date or specific knowledge.
- ✓ RAG solves this by retrieving relevant information from external sources and adding it to the prompt.
- ✓ This allows the LLM to generate more accurate, grounded, and context-aware responses.
- ✓ Here RAG helps.

3. RAG Architecture: Step by step



3.1. Encode Docs

- ✓ Convert documents into embeddings using an embedding model.

3.2. Index

- ✓ Store those embeddings in a vector database.

3.3. Encode Query

- ✓ Convert the user query into an embedding.

3.4. Similarity Search:

- ✓ Search the vector DB for documents similar to the query.

3.5. Retrieve:

- ✓ Get the top matching documents.

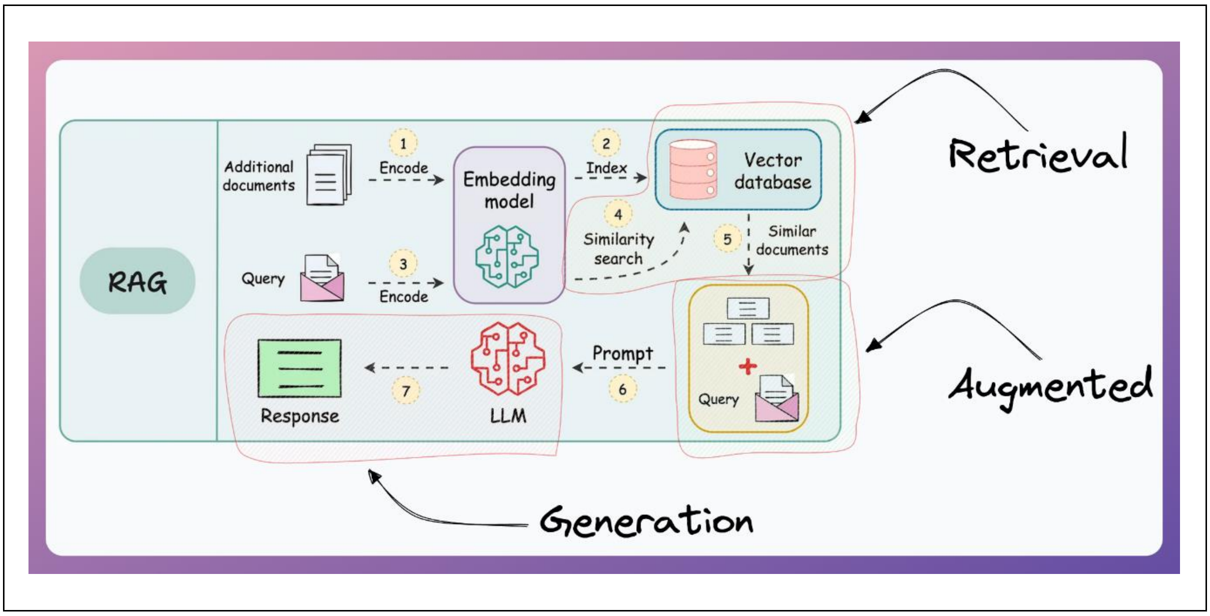
3.6. Prompt LLM:

- ✓ Combine the retrieved documents with the query and send to the language model.

3.7. Generate Response:

- ✓ The LLM generates a final answer based on the augmented input.

4. RAG: Retrieval Augmented Generation



4.1. Retrieval

- ✓ Fetching relevant info from a source (e.g., database).

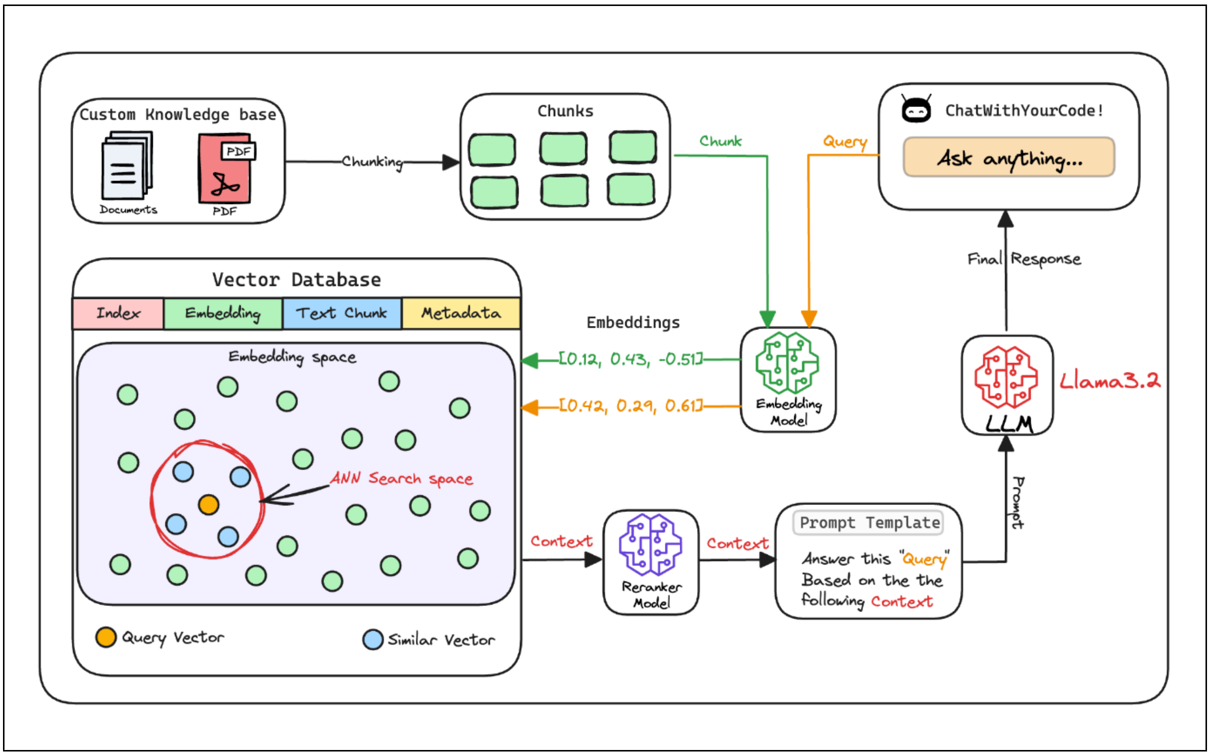
4.2. Augmented

- ✓ Adding extra context to improve the process (e.g., text generation).

4.3. Generation

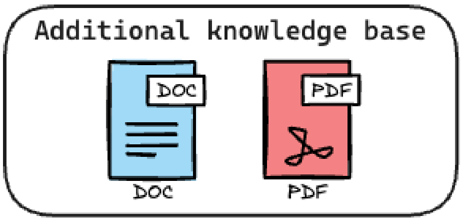
- ✓ Creating or producing something, like generating text.

5. Workflow of a RAG System



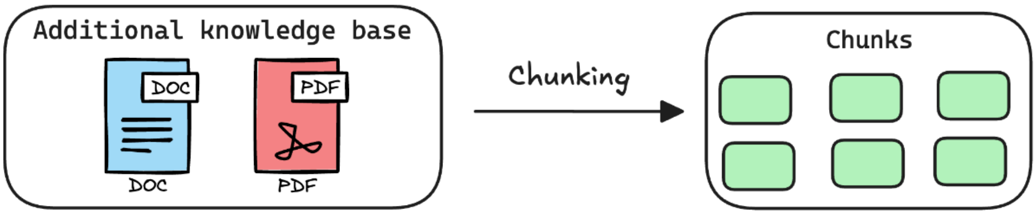
5.1. Addition knowledge base

- ✓ We start with some external knowledge that wasn't seen during training, and we want to enhance the LLM.



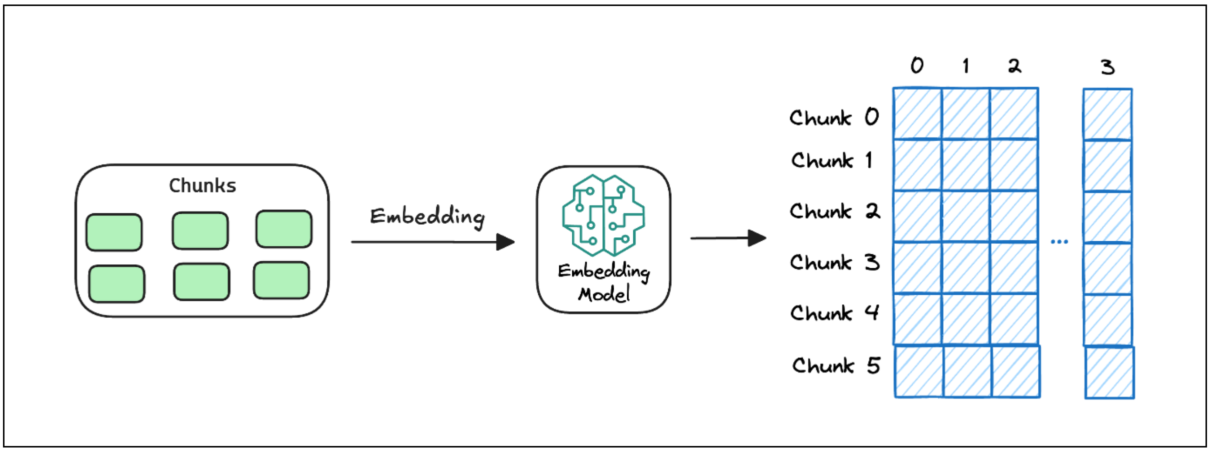
5.2. Create Chunks

- ✓ Before storing knowledge, it's broken into smaller, meaningful chunks.
- ✓ This improves retrieval accuracy and ensures each piece of information is easily searchable during query time.



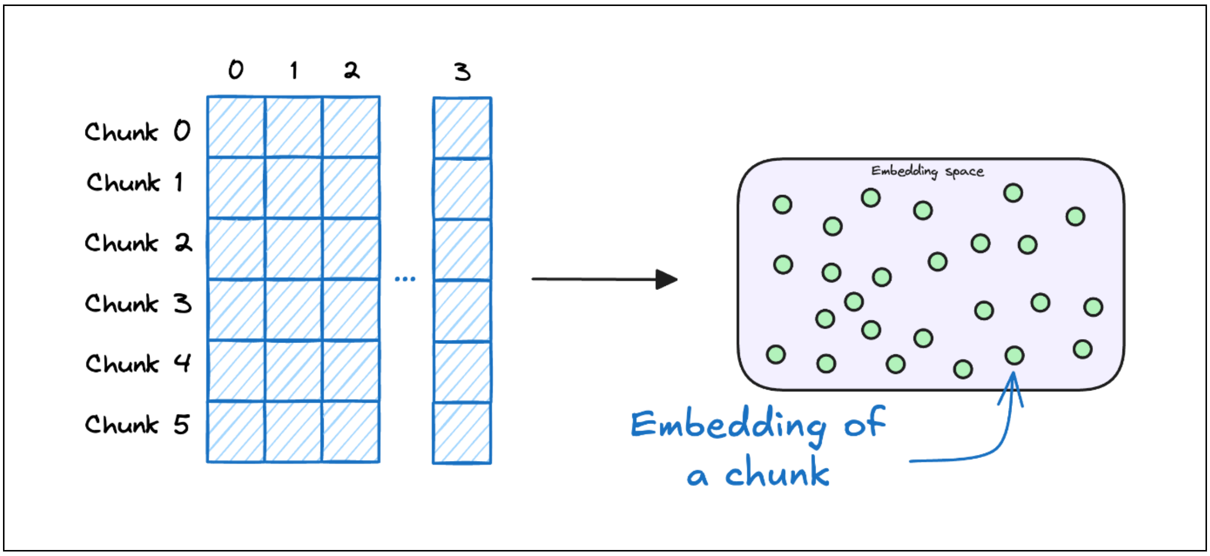
5.3. Generate embeddings

✓ After chunking, we embed the chunks using an embedding model.



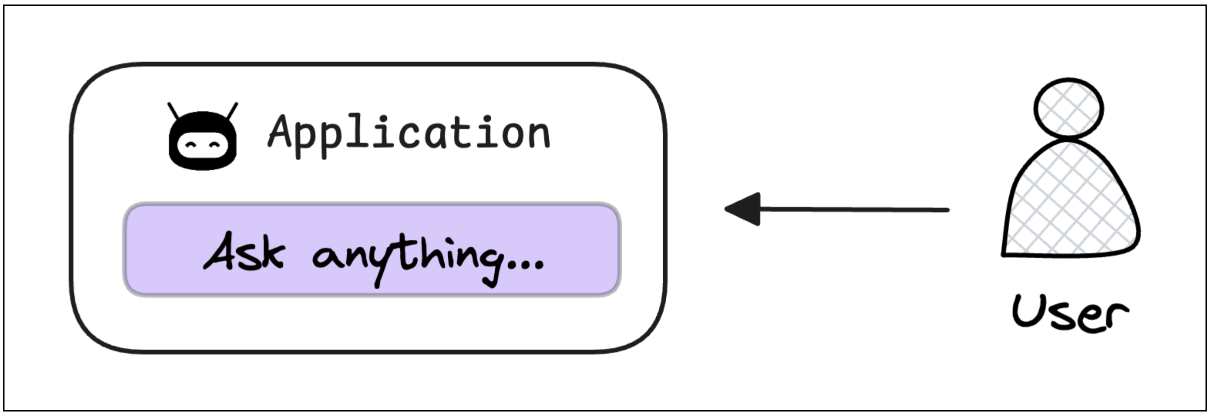
5.4. Store embeddings in a vector database

✓ These embeddings are then stored in the vector database



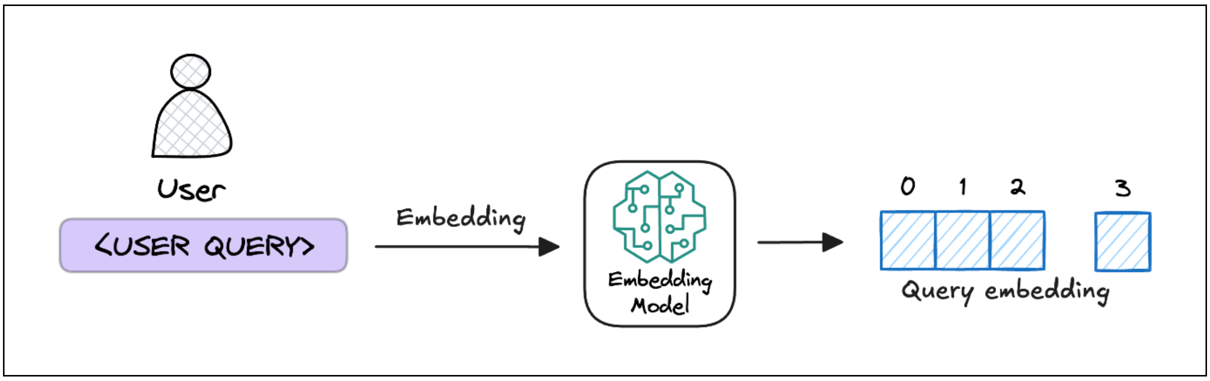
5.5. User input query

- ✓ Next, the user inputs a query, a string representing the information they're seeking.



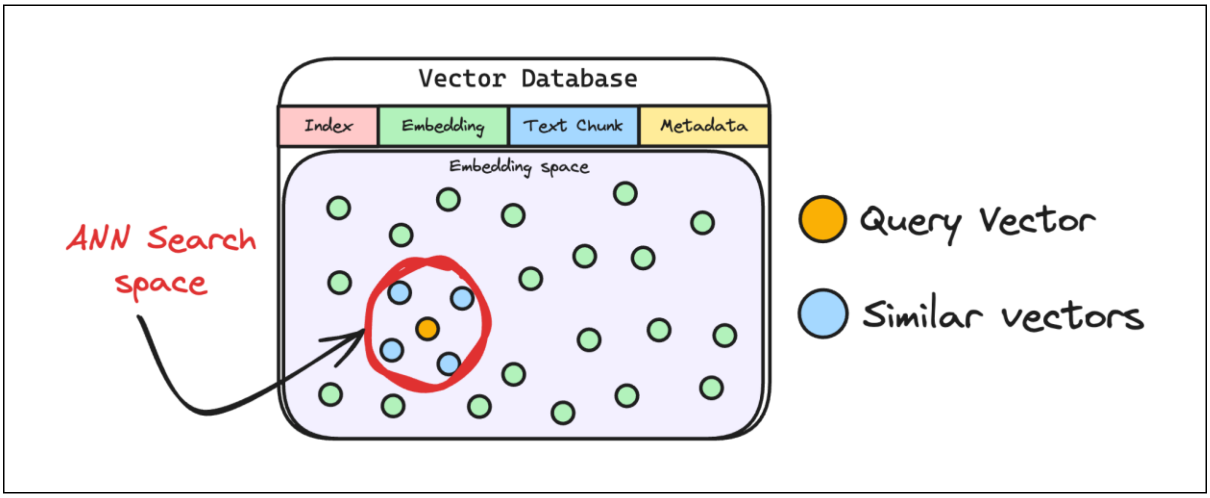
5.6. Embed the query

- ✓ This query is transformed into a vector using the same embedding model we used to embed the chunks earlier in Step 2.



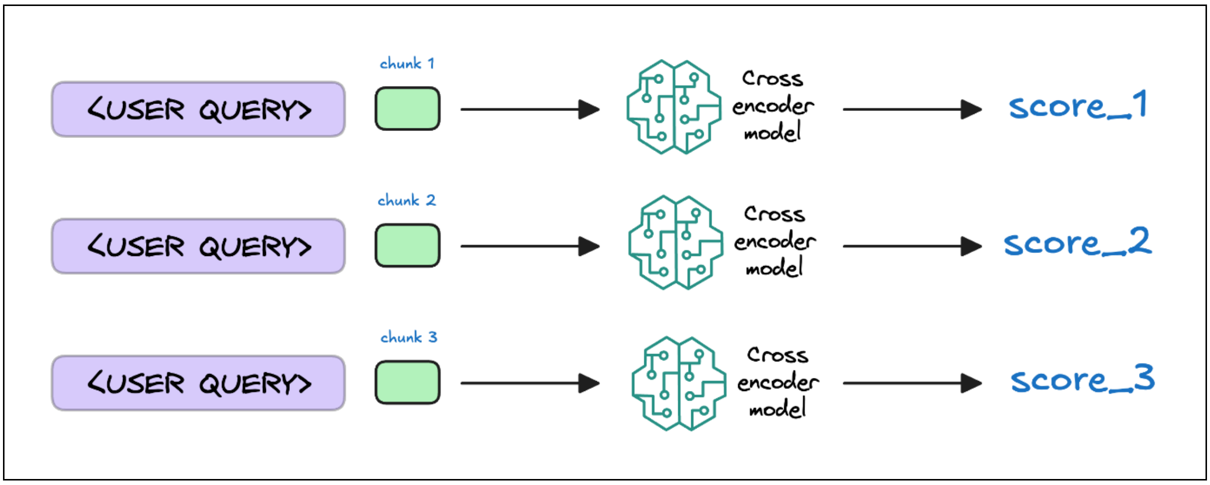
5.7. Retrieve similar chunks

✓ The vectorized query is then compared against our existing vectors in the database to find the most similar information.



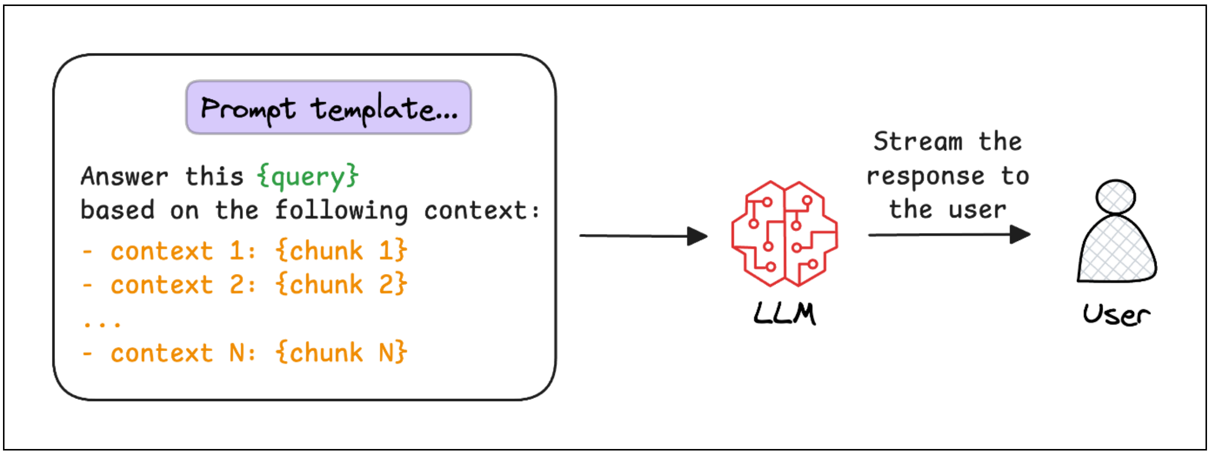
5.8. Re-rank Chunks

✓ After retrieval, a cross-encoder re-scores the chunks to prioritize the most relevant ones based on the query.



5.9. Generate Final Response

- ✓ The top-ranked chunks and the user query are combined into a prompt and sent to the LLM, which generates a final, informed response.



6. Tool Stack for Building a RAG System

6.1. LlamaIndex

- ✓ Simplifies connecting LLMs with external data sources for indexing and querying.

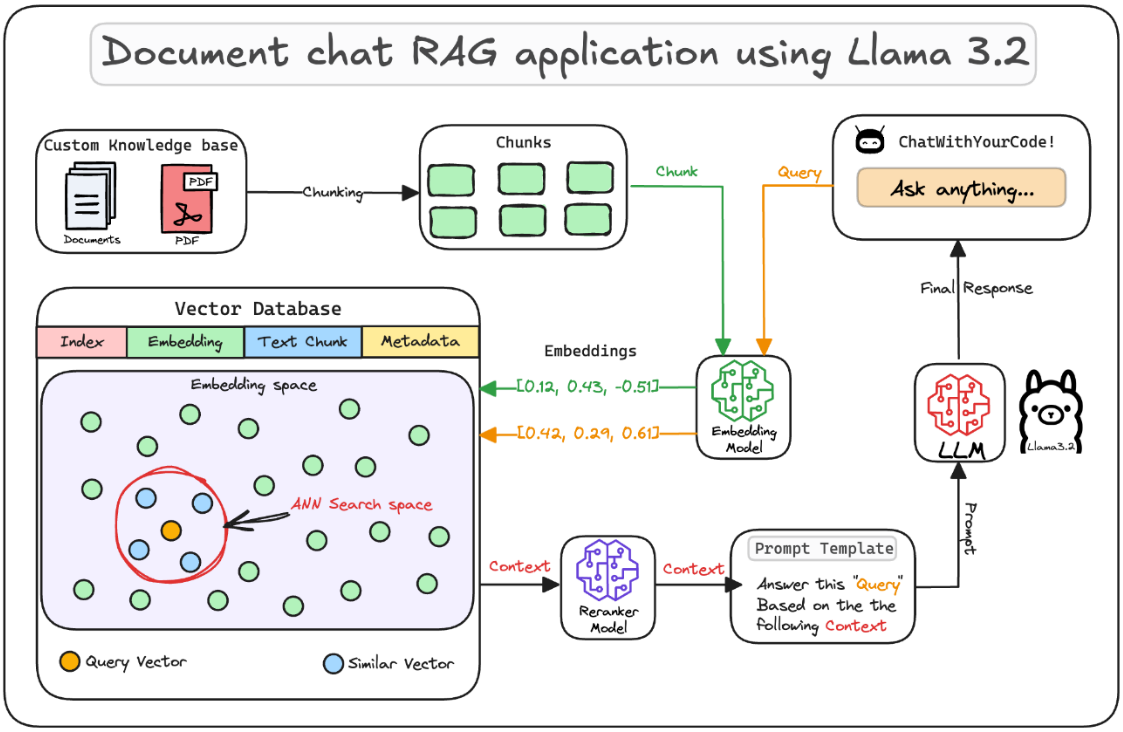
6.2. Qdrant

- ✓ Open-source vector database for fast, filtered similarity search at scale.

6.3. Ollama

- ✓ Runs LLMs locally, ideal for privacy-focused applications

7. RAG application using Llama 3.2



- ✓ **Custom Knowledge Base**: Input documents (e.g., PDFs, text files)
- ✓ **Chunking**: Break documents into smaller pieces (chunks)
- ✓ **Embedding**: Convert chunks into vector embeddings using an Embedding Model
- ✓ **Vector Database**: Store embeddings and related metadata. Perform ANN (Approximate Nearest Neighbor) search to find similar vectors to the user's query
- ✓ **Query Processing**: User submits a question. Convert query to a vector. Retrieve top matching chunks
- ✓ **Re-ranking** (Optional): Use a Re-ranker Model to refine and prioritize the most relevant chunks
- ✓ **Prompt Construction**: Fill a prompt template with the query and retrieved chunks
- ✓ **LLM Generation** (Llama 3.2): Final response is generated and shown to the user