Retrieval-Augmented Generation (RAG) — A Basic Introduction

Retrieval-Augmented Generation (RAG) is an innovative approach in natural language processing that combines the power of pre-trained language models with external knowledge retrieval systems. Traditional language models rely solely on the information learned during their training. However, they may sometimes generate incorrect or incomplete answers when the knowledge required is outside their training data. RAG addresses this limitation by retrieving relevant documents or pieces of information from a large external database before generating a response.

In RAG, when a user inputs a query, the system first searches a knowledge base — which can be documents, PDFs, or other data — to find the most relevant passages. These passages are then fed into a language model, such as GPT, which uses them to generate a more accurate and context-aware response. This method improves factual correctness and allows the system to handle queries about recent or specialized information not contained in the original model's training data.

One of the key components of a RAG system is the vector database, where documents are stored as embeddings — numerical representations of text. When a query is received, it is also converted into an embedding, and the system retrieves documents with embeddings closest to the query embedding. This retrieval step ensures that only relevant content is provided to the language model.

RAG systems have wide applications, including question answering, customer support, and document summarization. They are particularly useful in domains where up-to-date or specialized knowledge is crucial.

To build a simple RAG system, you typically need to perform the following steps:

1. **Document ingestion:** Load and preprocess documents (e.g., PDFs) into manageable chunks.
2. **Embedding:** Convert chunks into vector embeddings using models like OpenAI's text-embedding-3-large.
3. **Storage:** Store embeddings in a vector database such as Chroma or FAISS.
4. **Retrieval:** When a query is asked, convert it into an embedding and retrieve the most relevant chunks.
5. **Generation:** Pass the retrieved chunks and the query to a language model to generate a response.

This approach enhances the model's ability to provide grounded, relevant, and up-to-date answers.