Gen AI: LLM Capabilities and Alignment

Introduction to RAG

Introduction to Retrieval-Augmented Generation (RAG)

Learning Objective

By the end of this lesson, you will be able to:

- **Explain** Retrieval-Augmented Generation (RAG) and how it enhances AI knowledge retrieval.
- **Describe** the key components and workflow of RAG.
- **Build** a simple RAG system using vector databases and embeddings.
- **Evaluate** RAG performance using retrieval and generation metrics.

What is RAG?

Retrieval-Augmented Generation (RAG) improves AI responses by retrieving relevant external knowledge instead of relying solely on a model's internal memory. This makes AI-generated responses more accurate, dynamic, and adaptable.

Why Use RAG?

- ✅ **Reduces hallucinations**: Fetches real-time, fact-based information.
- ✅ **No need for constant fine-tuning**: Updates knowledge without retraining the model.
- ✅ **Improves efficiency**: Uses external data sources dynamically.

How RAG Works

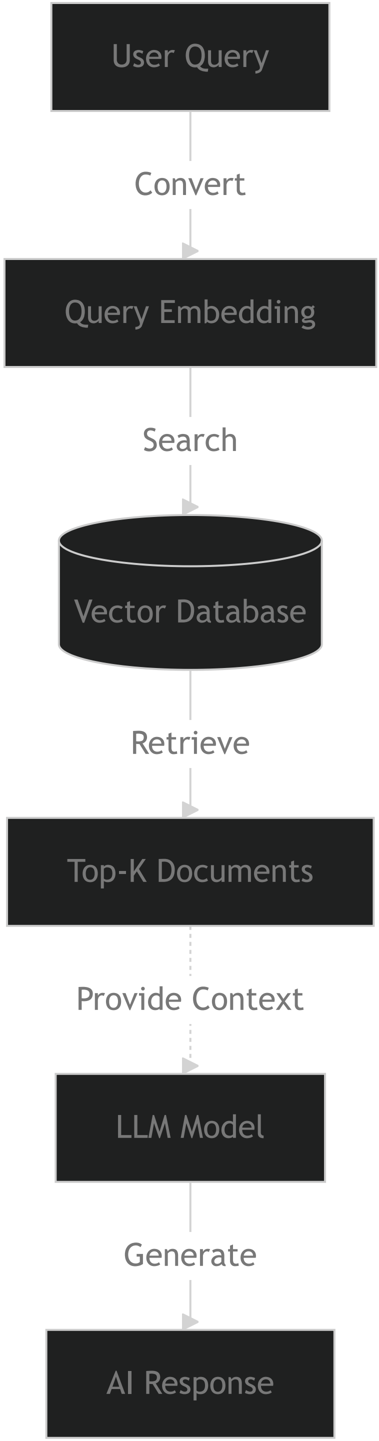
RAG has **two main steps**:

Step 1: Retrieval

- **Convert query to embedding** (numerical vector representation).
- **Search a vector database** for similar documents.
- **Select the most relevant results** to use as context.

Step 2: Generation

- The retrieved documents are **passed to an LLM** as context.
- The model **generates a response** based on both the user’s query and retrieved documents.



Hands-On: Build a Simple RAG System

We'll create a basic **RAG pipeline** using **Hugging Face embeddings** and **FAISS (a vector database)**.

Step 1: Install Dependencies

Copy

```
pip install faiss-cpu transformers datasets torch
```

Step 2: Load and Embed Documents

Copy

```
from transformers import AutoTokenizer, AutoModel
import torch
import faiss
import numpy as np

# Load embedding model
tokenizer = AutoTokenizer.from_pretrained("sentence-transformers/all-MiniLM-L6-v2
model = AutoModel.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")

# Example documents
documents = [
    "The Eiffel Tower is located in Paris, France.",
    "The capital of Germany is Berlin.",
    "The Great Wall of China is a famous historical landmark."
]

# Convert documents to embeddings
def embed_text(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    with torch.no_grad():
        embeddings = model(**inputs).last_hidden_state.mean(dim=1)
    return embeddings.squeeze().numpy()

# Create document embeddings
embeddings = np.array([embed_text(doc) for doc in documents])
```

Step 3: Store Embeddings in FAISS Vector Database

Copy

```
# Initialize FAISS index
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(embeddings)
```

Step 4: Retrieve Relevant Documents for a Query

Copy

```
query = "Where is the Eiffel Tower?"
query_embedding = embed_text(query).reshape(1, -1)
D, I = index.search(query_embedding, k=1)  # Retrieve top-1 match

print("Retrieved Document:", documents[I[0][0]])
```

Evaluating RAG Performance

Key evaluation methods:

- **Precision@K**: Measures how many of the top-k retrieved documents are relevant.
- **BLEU/ROUGE Scores**: Compares generated responses with reference answers.
- **Human Evaluation**: Checks factual accuracy and relevance.