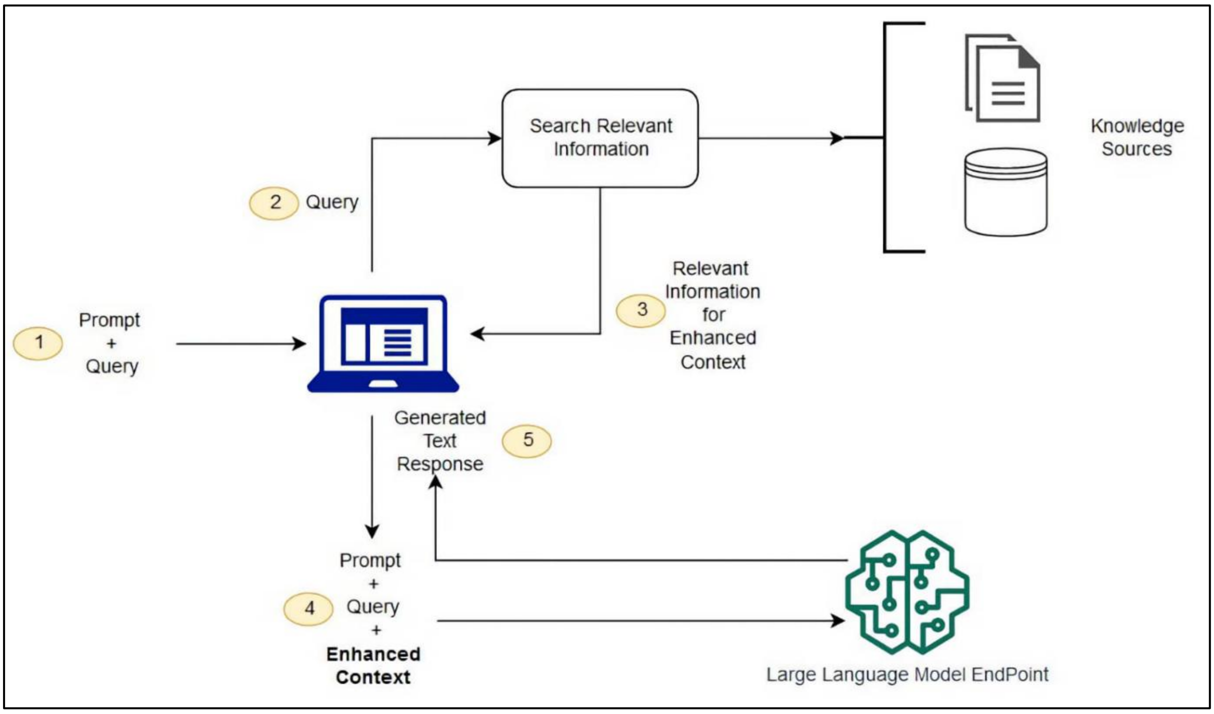
What is RAG (Retrieval-Augmented Generation)?

RAG is a hybrid approach that combines **retrieval-based** and **generation-based** methods to improve the performance of language models, especially in tasks requiring factual accuracy or domain-specific knowledge.

How RAG Works:

1. **Query Input**: A user provides a question or prompt.
2. **Retrieval Step**:
   - The system searches a **knowledge base** (often stored in a VectorDB) to find relevant documents or passages.
   - These documents are selected based on **semantic similarity** to the query.
3. **Augmentation**:
   - The retrieved documents are passed along with the original query to a **language model** (like GPT).
4. **Generation**:
   - The model uses both the query and the retrieved context to generate a more accurate and informed response.



RAG Architecture

RAG typically consists of two main components:

1. Retriever:

- Converts the query into a vector (embedding).
- Searches a **corpus of documents** stored in a VectorDB.
- Returns top-k relevant documents.

2. Generator:

- Takes the query and retrieved documents.
- Uses a sequence-to-sequence model (e.g., BART, T5, GPT) to generate a response.

This can be formalized as:

$P(y|x) = \sum_{d \in D} P(y|x, d) \cdot P(d|x)$

Where:

- x = input query
- d = retrieved document
- y = generated output
- D = set of top-k documents

Variants of RAG

- **RAG-Sequence**: Generates output token-by-token using one document at a time.
- **RAG-Token**: Considers all documents jointly for each token generation.

Benefits of RAG:

- **Improves factual accuracy** by grounding responses in external data.
- **Scales knowledge** without retraining the model.
- **Keeps responses up-to-date** by retrieving from dynamic sources.

What is a VectorDB (Vector Database)?

A **VectorDB** is a specialized database designed to store and search **vector embeddings**—numerical representations of data (like text, images, or audio) in high-dimensional space.

Mathematical Foundation

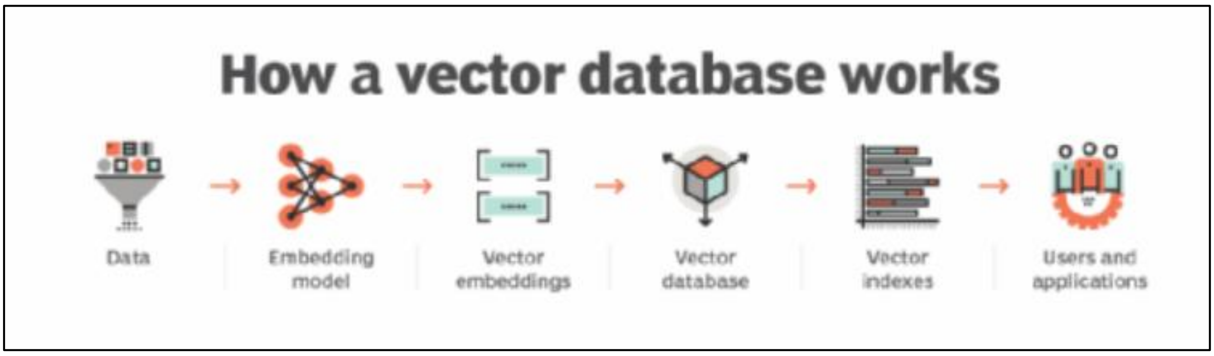
Given a query vector qqq and a set of document vectors {d1,d2,…,dn}\{d_1, d_2, ..., d_n\}{d1,d2,…,dn}, the goal is to find the top-k documents that minimize the **distance** (or maximize similarity) to qqq.

Common similarity metrics:

- **Cosine similarity**:

$sim(q,d) = q \cdot d / (\|q\|\|d\|)$

- **Euclidean distance**
- **Dot product**



1. **Embedding Creation**:
    - Text (e.g., documents, queries) is converted into **vectors** using models like BERT, OpenAI embeddings, etc.
2. **Storage**:
    - These vectors are stored in the VectorDB along with metadata (e.g., source, tags).
3. **Similarity Search**:
    - When a query is embedded into a vector, the VectorDB performs a **nearest neighbor search** to find the most similar vectors (i.e., relevant documents).
    - This is often done using algorithms like **FAISS**, **Annoy**, or **HNSW**.

Popular VectorDBs:

- **Pinecone**
- **Weaviate**
- **Milvus**
- **Qdrant**
- **Chroma**

Use Cases:

- Semantic search
- Recommendation systems
- RAG pipelines
- Image and audio similarity search

How RAG and VectorDB Work Together

In a typical RAG pipeline:

1. A query is embedded.
2. The embedding is used to search a VectorDB for relevant documents.
3. Retrieved documents are fed into the LLM.
4. The LLM generates a response using both the query and the retrieved context.

This combination allows LLMs to **access external knowledge** dynamically, making them more powerful and reliable.