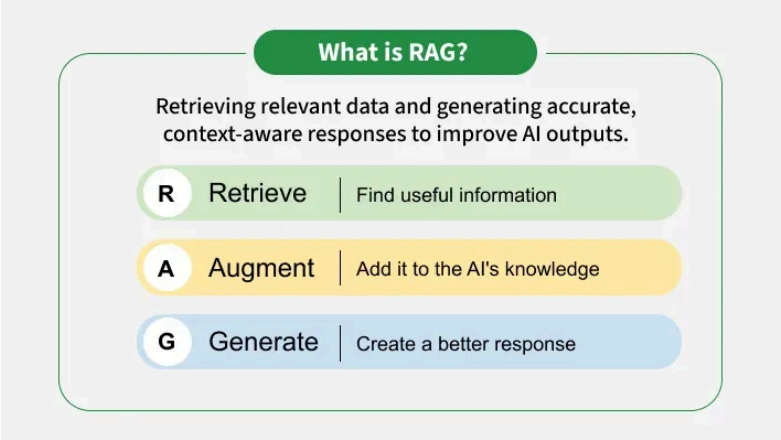
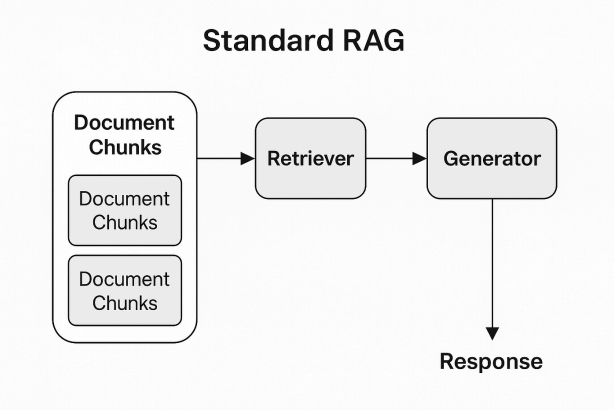
RAG : Retrieval-Augmented Generation



Retrieval-Augmented Generation (RAG) is an AI framework that enhances the output of **Large Language Models (LLMs)** by grounding their responses in external, authoritative knowledge sources. It essentially gives an LLM an "open book" to reference, overcoming limitations like relying solely on static training data and generating factually incorrect information (known as **hallucinations**).

How does RAG work?



RAG integrates two main components—a **retriever** and a **generator**—in a multi-step process that occurs at runtime when a user submits a query.

1. **Preparation Phase (Indexing):** Before the system is used, an external **knowledge base** is prepared:
    a. **External Data:** This is the authoritative, domain-specific, or proprietary information (documents, databases, web content) that the LLM needs access to.
    b. **Chunking & Embedding:** The external documents are broken down into smaller, semantically coherent pieces, or **chunks**. An **embedding model** then converts these text chunks into numerical representations called **vectors** (or embeddings).

Vectors capture the semantic meaning of the text, allowing for similarity comparisons.

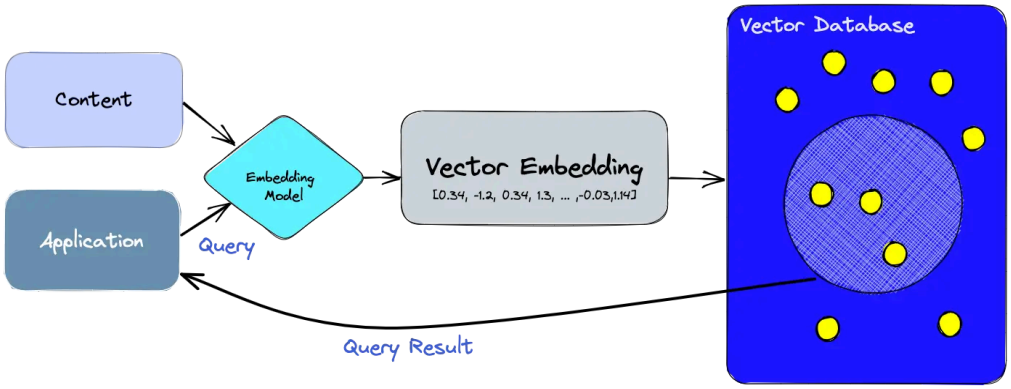
c. **Vector Database:** These vectors are stored in a specialized **vector database** (or vector index), which is optimized for fast and accurate "semantic search" based on vector similarity.

2. **Execution Phase (Query Time):** When a user submits a query (a question or prompt), the RAG system executes the following steps:
    a. **Query Encoding:** The user's query is also converted into a numerical **query vector** using the same embedding model.
    b. **Retrieval:** The system searches the vector database to find the **top-k** (most similar) document chunks whose vectors are closest to the query vector. This identifies the most relevant, contextually-matching pieces of information from the external knowledge base.
    c. **Augmentation (Prompt Engineering):** The retrieved text chunks are combined with the user's original query to create an **augmented prompt**. This enriched prompt provides the LLM with the necessary external context, effectively "stuffing" the facts into the model's context window.
    d. **Generation:** The augmented prompt is passed to the **Large Language Model (LLM)** (the generator). The LLM uses its pre-trained knowledge *plus* the new, contextually relevant information to generate a final, grounded, and coherent response.

Key Benefits:

1. Reduced Hallucinations as it grounds the LLM’s response in verifiable external facts, thereby decreasing the chances of getting incorrect data.
2. Allows LLM to access latest information even if it wasn’t in the original training data.
3. Enables LLMs to answer questions about proprietary, specialized, or internal company data.
4. It's much cheaper and faster than continuously **fine-tuning** or completely **retraining** a massive LLM every time new data is introduced.
5. RAG systems can cite the source documents used for generation, allowing users to verify the information and increasing trust in the output.

Vector DB



A **Vector Database (Vector DB)** is a specialized database designed to efficiently store, manage, and query **high-dimensional vectors**, which are numerical representations of unstructured data. Its core function is to enable **semantic search**—finding data that is **conceptually similar** to a query, rather than relying on exact keyword matches.

How does Vector DB work?

- **Vector Embeddings:** Before data enters the database, it's processed by an **Embedding Model** (a type of neural network). This model converts unstructured data (like text, images, or audio) into a fixed-length list of floating-point numbers called a **vector embedding** (e.g., `[0.2, -0.9, 1.5, ... ]`).
    - This numerical vector captures the **semantic meaning** or features of the data. In the resulting high-dimensional space, items with similar meaning are positioned closer together.
- **Indexing:** The Vector DB stores these embeddings and uses advanced algorithms (like HNSW or IVF) to **index** them. This indexing process is optimized for **Approximate Nearest Neighbor (ANN) search**, which allows for rapid lookups across billions of vectors.
- **Similarity Search:** When a user submits a query (e.g., "Find documents about electric cars"), the query is also converted into a query vector. The Vector DB then measures the distance (using metrics like cosine similarity) between the query vector and all stored vectors, instantly retrieving the **closest** (most relevant) vectors.

Key Uses:

1. Retrieval-Augmented Generation
2. Semantic Search
3. Recommendation Systems
4. Image/Video Recognition