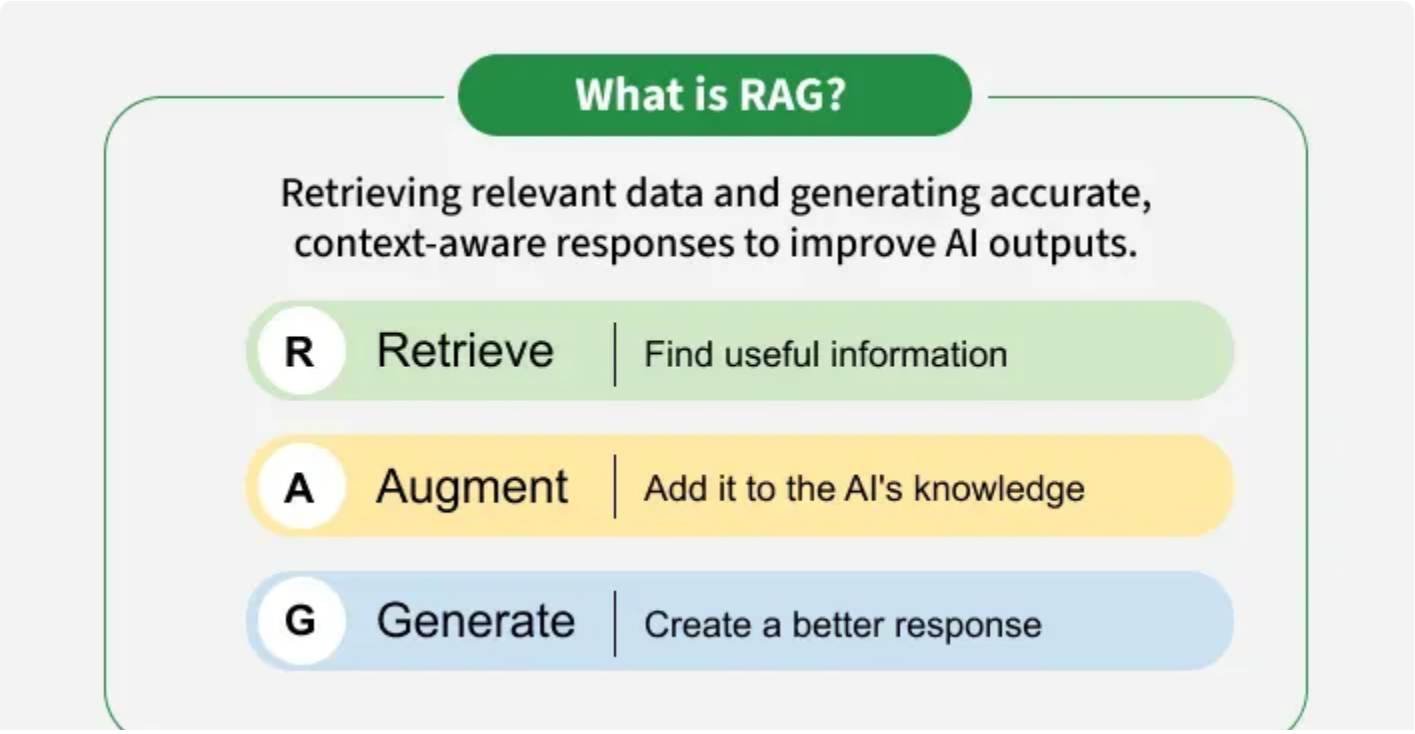
Retrieval-Augmented Generation (RAG)

KoustubhPK • 2026

1. What is Retrieval-Augmented Generation (RAG) ?

Retrieval-Augmented Generation (RAG) is a technique that enables language models to retrieve trusted external information before generating answers, instead of relying only on their training data.

It operates in a similar way to an open-book examination: relevant documents are retrieved first (retrieval) and then used by the language model to generate accurate and up-to-date responses (generation).



Key Components of RAG

RAG breaks down into two main phases Retrieval & Generation comprising four primary steps:

1. Data Ingestion: External data (PDFs, internal documents, websites) is broken into small, manageable pieces called "chunks".

2. Embedding: These chunks are converted into numerical representations called vectors (using embedding models) and stored in a specialized Vector Database.

3. Retrieval: When a user asks a question, the system converts that query into a vector and searches the vector database for the most relevant "chunks" of information.

4. Generation (Augmentation): The retrieved data and the user's original query are sent to the LLM. The LLM uses this context to generate an accurate, informed response

---

2. Why is RAG Important?

RAG (Retrieval-Augmented Generation) enables large language models to generate accurate, up-to-date and grounded responses by retrieving information from external and private data sources — without retraining the model — while significantly reducing hallucinations.

1. Solves LLM Hallucinations

LLMs generate text based on learned patterns rather than verified facts. Without external context, responses may appear confident but can be incorrect.

RAG grounds the model by injecting relevant, retrieved information before generation, significantly reducing hallucinations.

2. Enables Access to Fresh and Private Data

LLMs are trained on static and mostly public datasets and therefore cannot:

- access recent information
- query internal or proprietary documents

RAG enables models to:

- use up-to-date information

- answer questions over private datasets without retraining the model

3. Eliminates Costly Re-Training

Fine-tuning a model for every data update is:

- expensive
- time-consuming
- operationally difficult to maintain

RAG cleanly separates:

- **knowledge** (external data)
- **reasoning** (the language model)

This architecture is cheaper, faster to update and highly scalable.

4. Enables Source-Aware and Verifiable Answers

RAG allows systems to attach references to the retrieved documents used to generate a response.

This enables:

- answer verification
- traceability to original documents
- compliance, audit and review workflows

Source-aware generation significantly improves trust in enterprise deployments.

5. Makes LLMs Production-Ready

Without retrieval, LLMs are primarily:

- conversational
- creative
- risky for operational use

RAG transforms LLMs into:

- knowledge assistants
- internal search systems
- support agents
- decision-support tools

6. Improves Accuracy and Trust

Because answers are generated using retrieved context:

- responses are more accurate
- outputs are easier to audit
- reasoning is easier to explain
- source references can be included

This is essential for enterprise and regulated environments.

3. RAG vs. Fine-Tuning

RAG (Retrieval-Augmented Generation)

Connects the language model to external knowledge sources and retrieves relevant information at query time.

- Does **not** modify the model's internal parameters
- Uses external and private data dynamically
- Ideal for frequently changing knowledge

Analogy: Like giving a student an encyclopedia to look up facts.

Fine-Tuning

Trains the language model on new data, permanently updating its internal parameters.

- Changes the model's learned behavior
- Improves task-specific performance
- Requires training infrastructure and careful evaluation

Analogy: Like putting a student through a specialized training program.

4. RAG Architecture – Overview

RAG consists of two tightly coupled stages: **retrieval** and **generation**.
Together, they allow a language model to ground its responses in external and private knowledge while preserving fluent natural-language output.

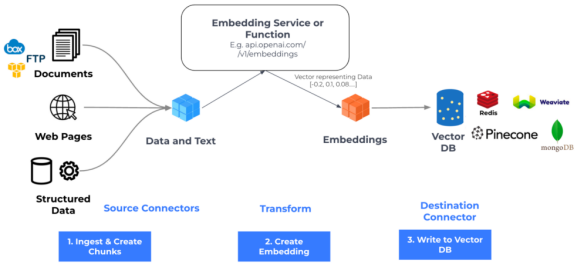
Retrieval stage

Generation stage

The retrieval stage is responsible for locating information that is relevant to the user's query from one or more external data sources.

During ingestion, source documents are converted into vector embeddings and stored inside a vector database.
At query time, the user question is embedded and matched against this index to retrieve the most relevant chunks.

This process enables targeted and semantically meaningful search instead of simple keyword matching.

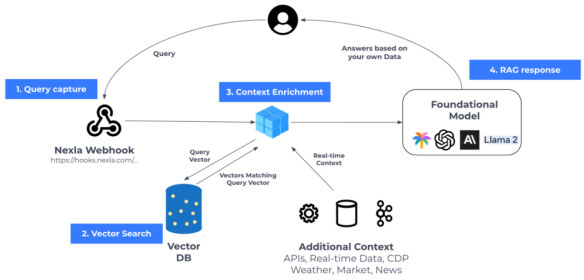


The high-level process of ingesting data into an LLM system

The generation stage produces the final response using both the user query and the retrieved contextual information.

Instead of directly repeating retrieved content, the language model integrates the external context with its internal knowledge to generate a coherent, accurate and task-specific answer.

This grounding step significantly improves factual accuracy and contextual relevance.



The high-level process of retrieval-augmented generation

5. Types of RAG Architectures

Retrieval-Augmented Generation (RAG) combines large language models with external knowledge retrieval systems to improve factual accuracy, coverage, and contextual relevance.
By injecting retrieved information into the generation pipeline, RAG enables models to produce responses that are grounded in authoritative and up-to-date data.

Understanding the major RAG architectural patterns is essential for selecting the right design based on scalability, complexity, and application requirements.
The three widely adopted architectural variants are described below.

5.1 Naive RAG

Naive RAG represents the foundational and most straightforward form of retrieval-augmented generation.
In this architecture, the system retrieves a set of relevant documents or text chunks for a given user query and directly provides them to the language model as contextual input.

Key characteristics

- **Retrieval mechanism** – Uses basic retrieval approaches such as keyword search or simple semantic similarity to fetch relevant chunks from a pre-indexed knowledge base.
- **Context integration** – Retrieved documents are concatenated with the user query and passed to the language model as a single prompt context.
- **Processing flow** – The pipeline follows a simple linear sequence: retrieve → concatenate → generate. Retrieved content is not refined or re-ranked before generation.

5.2 Advanced RAG

Advanced RAG extends the naive architecture by introducing more sophisticated retrieval and context-handling techniques.
Its primary goal is to improve both the relevance and usefulness of the information provided to the language model.

Key characteristics

- **Enhanced retrieval strategies** – Employs techniques such as query expansion and multi-stage or iterative retrieval to increase recall and precision of retrieved documents.
- **Context refinement** – Applies filtering, re-ranking, or selective attention mechanisms to emphasize the most relevant segments of the retrieved context.
- **Optimization mechanisms** – Uses relevance scoring, context compression, and

augmentation techniques to ensure that the final prompt contains high-quality and task-specific information.

5.3 Modular RAG

Modular RAG represents a flexible and extensible architectural paradigm in which the RAG pipeline is decomposed into independent, specialized components.
Each component can be developed, optimized, and replaced independently.

Key characteristics

- **Decoupled pipeline components** – The architecture separates major stages such as query rewriting, retrieval, re-ranking, context construction, and generation into individual modules.
- **High configurability** – Enables different algorithms, models, and heuristics to be plugged into each stage, allowing application-specific tuning and experimentation.
- **System extensibility** – Supports the integration of additional modules such as conversational memory, multi-source search connectors, and structured knowledge sources (for example, knowledge graphs or enterprise search systems).

This modular design enables organizations to build scalable and production-ready RAG systems that can evolve as new retrieval techniques and model capabilities emerge.

---

6. Explain Basic RAG Pipeline

The Basic Retrieval-Augmented Generation (RAG) Pipeline operates through two main phases:

- i. Data Indexing
- ii. Retrieval & Generation

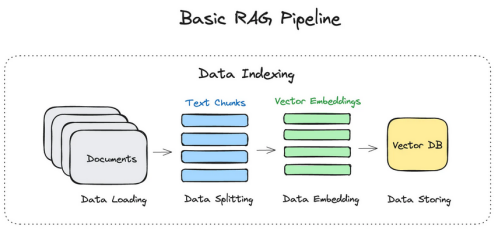
1. Data Indexing Process

Data Loading: This involves importing all the documents or information to be utilized.

Data Splitting: Large documents are divided into smaller pieces, for instance, sections of no more than 500 characters each.

Data Embedding: The data is converted into vector form using an embedding model, making it understandable for computers.

Data Storing: These vector embeddings are saved in a vector database, allowing them to be easily searched.



2. Retrieval and Generation Process

Retrieval: When a user asks a question:

- The user's input is first transformed into a vector (query vector) using the same embedding model from the Data Indexing phase.
- This query vector is then matched against all vectors in the vector database to find the most similar ones (e.g., using the Euclidean distance metric) that might contain the answer to the user's question. This step is about identifying relevant knowledge chunks.

Generation: The LLM model takes the user's question and the relevant information retrieved from the vector database to create a response. This process combines the question with the identified data to generate an answer.