RAG

Retrieval Augmented Generation

LLM Limitations

Knowledge Cutoff Limitation

- LLMs are trained on fixed datasets and can’t access new or domain-specific information beyond their training cutoff, leading to outdated answers.

Hallucination Problem

- Without external factual grounding, LLMs may confidently generate incorrect or fabricated outputs.

Data Privacy and Customization needs

- Enterprises often have private, proprietary data that can’t be shared for model retraining.

LLMs are resource intensive to both run and finetune on enterprise data.

Grounding LLMs with Factual Context

```
You: who won Asia cup 2025?
Gemini: The Asia Cup 2025 has not happened yet, so there is no winner.

The tournament is scheduled to take place in **2025**. We'll know the winner after it concludes!
```

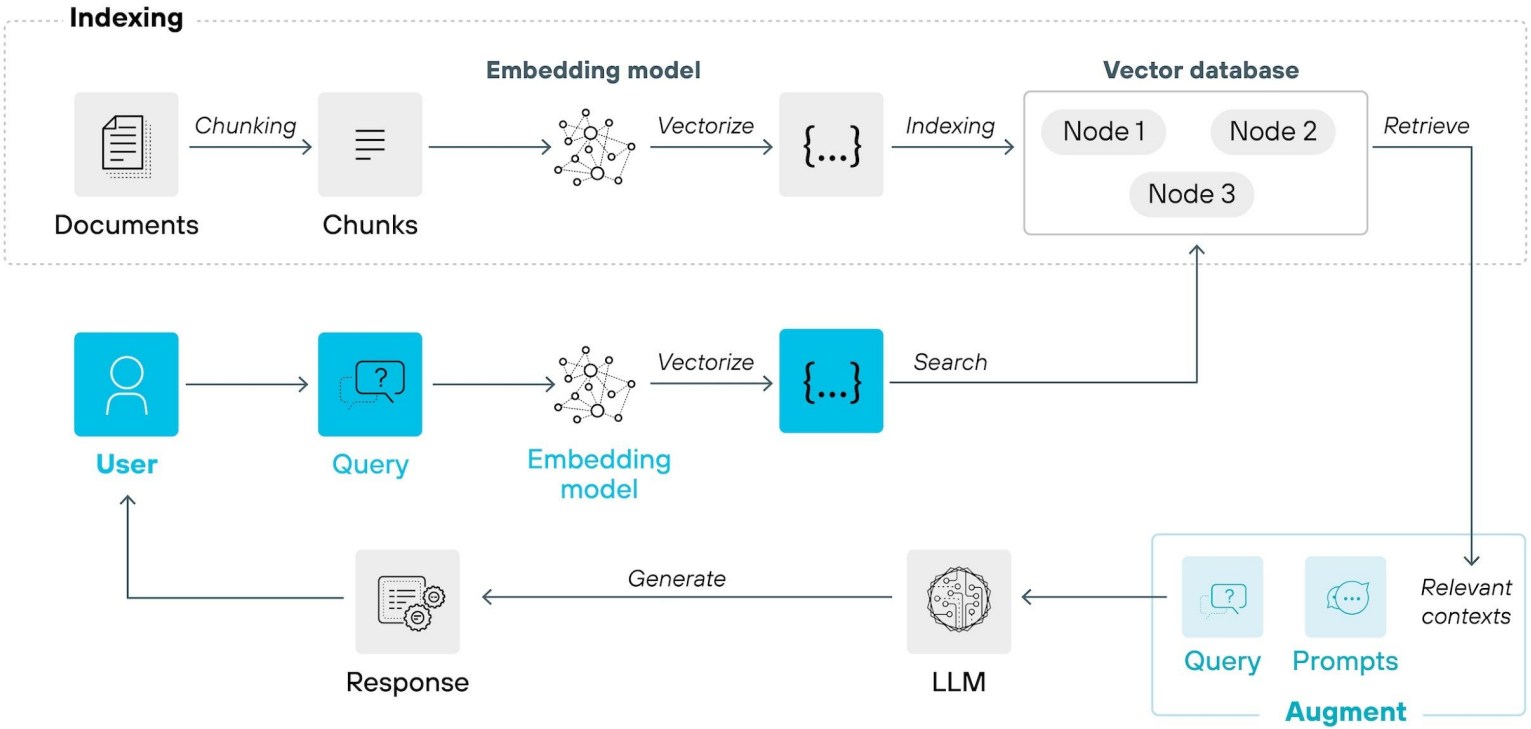
```
Context:  The 2025 Men's Asia Cup was the 17th edition of the Asia Cup cricket tournament.
        It took place in the United Arab Emirates (UAE) from 9 to 28 September 2025.
        In the final, India defeated Pakistan by 5 wickets to win their ninth title, and thus retained the title

You: Who won Asia cup 2025?
Gemini: Based on the text you provided, **India** won the 2025 Men's Asia Cup by defeating Pakistan by 5 wickets in the final.
```

But is this approach feasible for huge enterprise context ?

RAG — Scaling Factual Grounding

- **From Manual to Automated Grounding**
  - Instead of manually injecting context into prompts, RAG automates the retrieval of relevant information from vast knowledge sources — at query time.
- **Scalable Knowledge Integration**
  - RAG connects LLMs to dynamic, ever-growing datasets (internal documents, databases, web data), allowing factual grounding across millions of information pieces — without retraining.
- **Continuous Adaptation**
  - As data evolves, RAG automatically fetches the latest facts, ensuring responses remain current, domain-specific, and verifiable at scale.

RAG Architecture



RAG - Building Blocks

Indexing

- process of organizing a vast amount of text data in a way that allows the RAG system to quickly find the most relevant pieces of information for a given query

Retrieval

- Fetching the most relevant context based in the user query

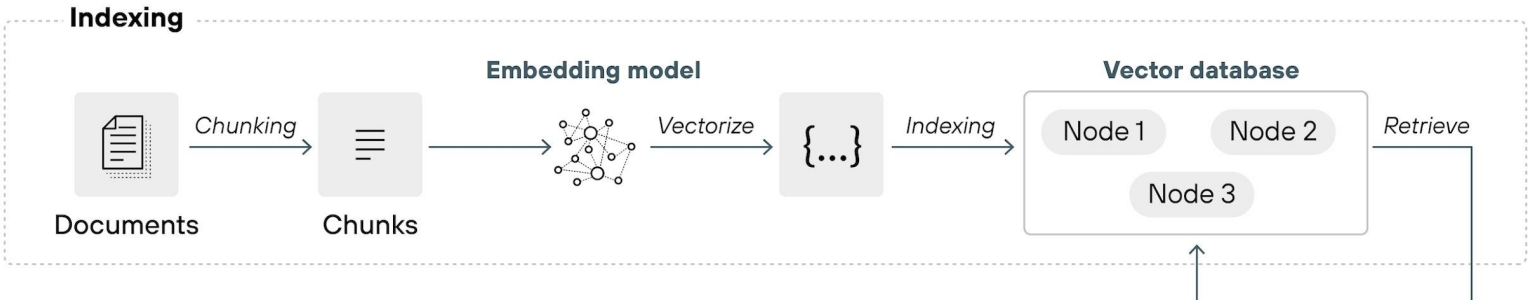
Augmentation

- the retrieved context and the user query are combined into a prompt to provide the model with context for the generation step

Generation

- the model generates output from the augmented prompt, using the context to drive a more accurate and relevant response.

Indexing


Indexing
Embedding model
Vector database
Chunking
Vectorize
Indexing
Retrieve
Node 1
Node 2
Node 3
Documents
Chunks

Indexing

What is it

- Indexing is the process of organizing and structuring data (e.g., document embeddings) so that similarity search can be performed quickly and accurately.

How does it help

- **Efficient Retrieval:** Retrieves relevant documents instantly instead of scanning all data manually, much like using a catalog in a library
- **Improved Accuracy:** Enables better matching between queries and documents, ensuring contextually relevant results for the language model
- **Scalability:** Efficiently manages large-scale datasets by using advanced indexing and search techniques
- **Reduced Latency:** Speeds up the overall RAG pipeline, making responses nearly real-time while maintaining quality.

Indexing - Building Blocks

Vectors — The Coordinates of Meaning

Each embedding is a vector, a list of numbers representing the position of a piece of text in multi-dimensional space.

Example:

Text	Vector (simplified)
“cat on mat”	[0.12, 0.98, -0.45, …]
“dog on rug”	[0.10, 0.95, -0.47, …]

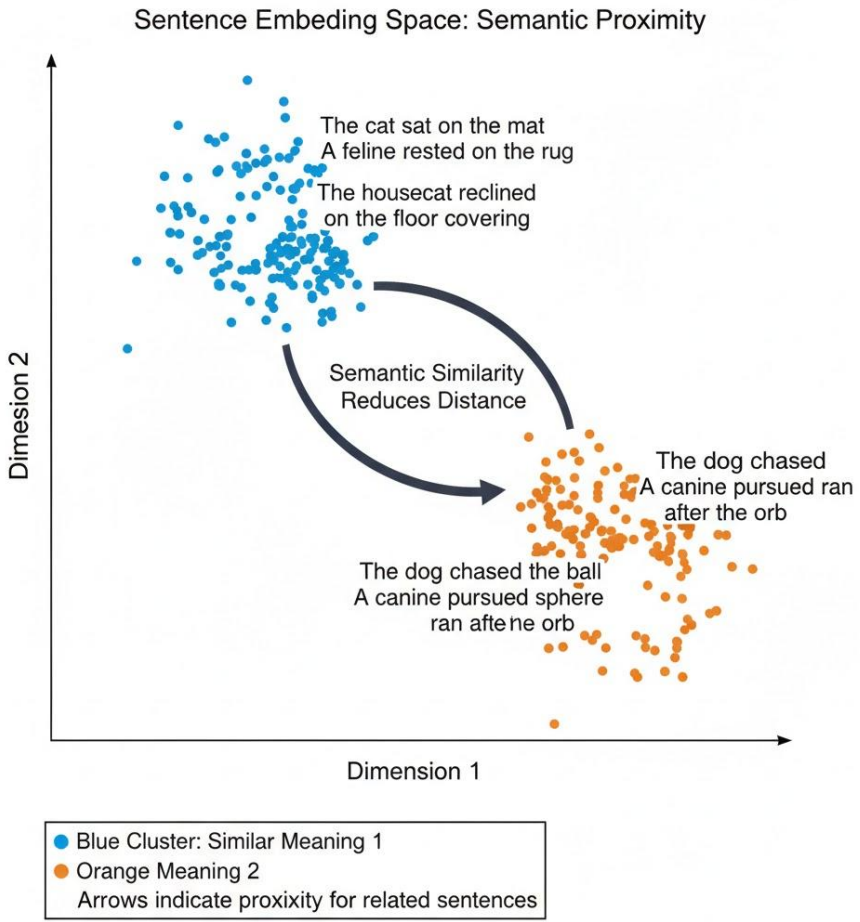
Indexing - Building Blocks

Embeddings - Turning Text into Numbers

- Embeddings are dense numerical representations of text that capture semantic meaning rather than exact words.
- Words or sentences with similar meanings have similar embeddings.
- Example: “car” and “automobile” → close in vector space
- Created using models like OpenAI Embeddings, Sentence-BERT, or E5.

Purpose: They make **text machine-understandable** for similarity search.

Text to Embeddings



Indexing - Building Blocks

Vector Databases

It is a specialized system that stores and manages vector embeddings—numerical representations of text, images, or other data—enabling fast similarity searches.

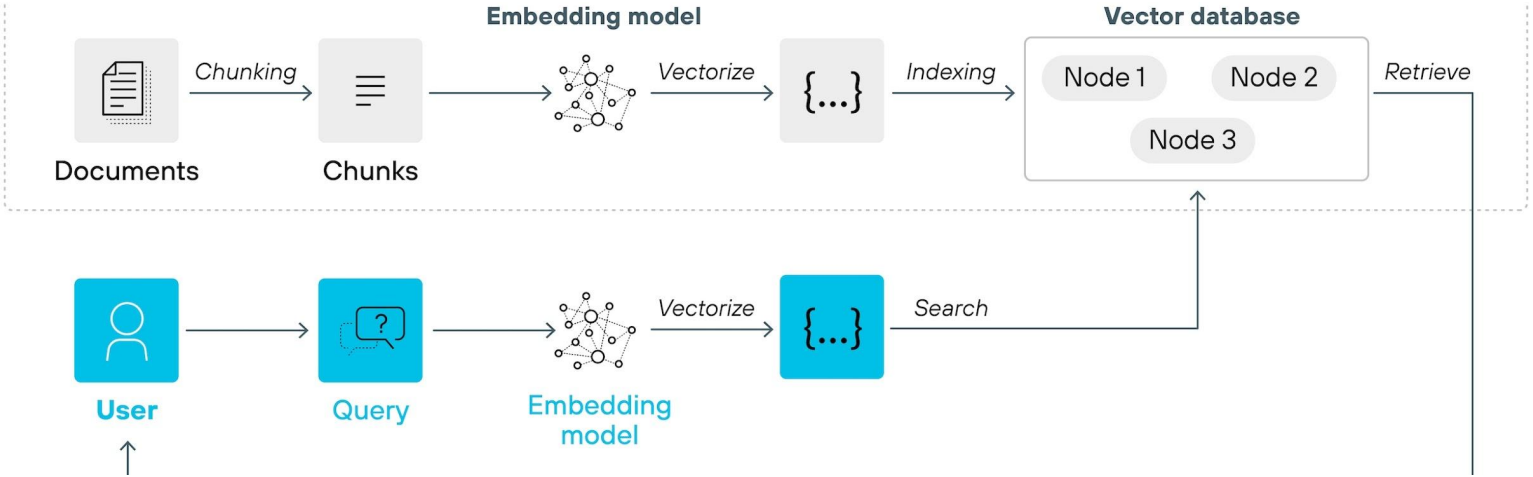
Ex: Pinecone, Weaviate, Milvus, FAISS, Chroma

Why not traditional databases

The retrieval engine must find passages semantically related to that question, even if the exact words differ i.e., ability to search semantically which traditional databases are lacking.

Retrieval

Retrieval is the process of finding documents or passages from a large collection that are most relevant to a given query.



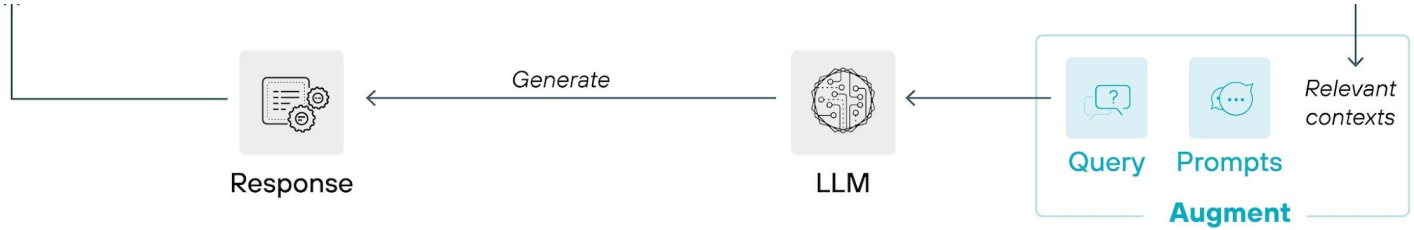
Retrieval - speed and accuracy

The speed and accuracy of retrieving relevant context from vector store depends on the indexing method we have chosen while creating the vector store.

Common indexing strategies implemented in RAG systems:

- **Approximate Nearest Neighbors (ANN)**: A fast approach that significantly reduces search time, though it sacrifices some accuracy in favor of efficiency.
- **Hierarchical Navigable Small World (HNSW)**: A popular strategy that balances speed and accuracy by organizing data in a multi-layer graph structure for optimized nearest neighbor searches.
- **IVF (Inverted File Index)**: This strategy enhances large-scale search efficiency by splitting high-dimensional vectors into clusters, thereby turning the retrieval process faster when handling massive datasets.
- **PQ (Product Quantization)**: Used in advanced RAG systems, this method compresses vector data to reduce memory usage while enabling efficient similarity searches.

Augmentation and Generation



After retrieving the relevant contexts for a given query, the user's query will be **augmented** with the relevant contexts into a single prompt.

We then invoke the LLM using the augmented user query as the prompt to **generate** the responses.

Resources

FreecodeCamp - video

Langchain’s hands-on tutorial - tutorial

Free LLM API - groq