Procesamiento de Lenguaje Natural Avanzado

Embeddings, Retrieval, and Search in NLP

From Prompting to Retrieval-Augmented Generation (RAG)

Dra. Helena Gómez Adorno
helena.gomez@iimas.unam.mx

Dr. Fazlourrahman Balouchzahi
fbalouc@iimas.unam.mx

Correo del curso:
pln.cienciadedatos@gmail.com

What We Have Learned So Far?

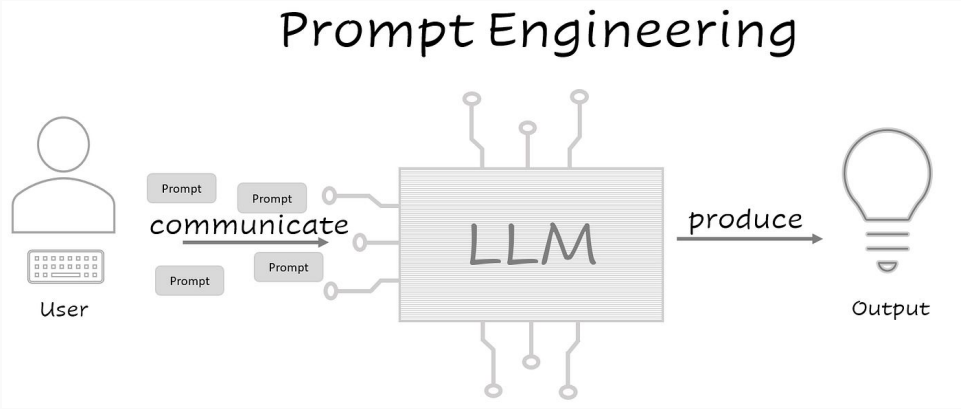
LLMs predict the next token

Strong at:

- Language generation
- Pattern recognition
- In-context learning

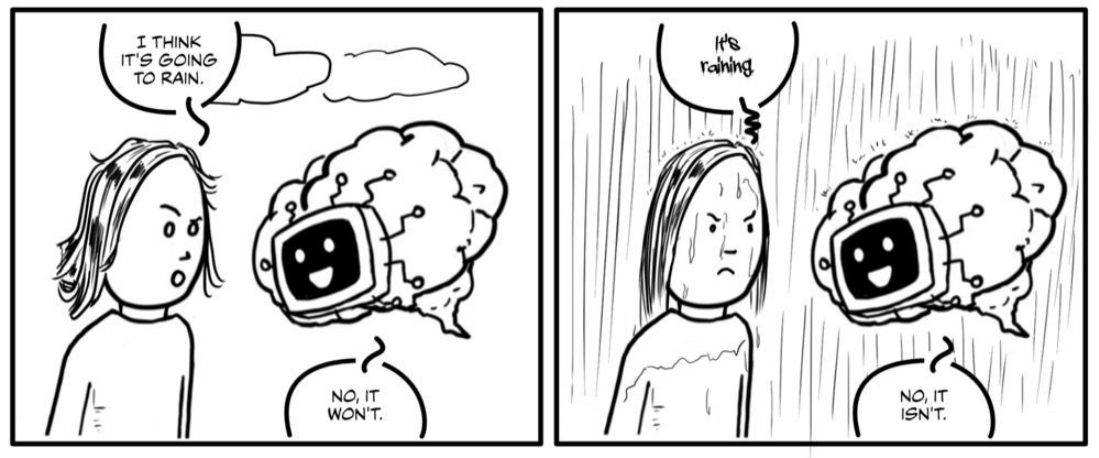
We saw:

- Prompting
- Chain-of-Thought
- Self-consistency



Where LLMs Fail?

- Hallucinations (fluent but false)
- No real-time knowledge
- Limited context window
- Weak factual reliability



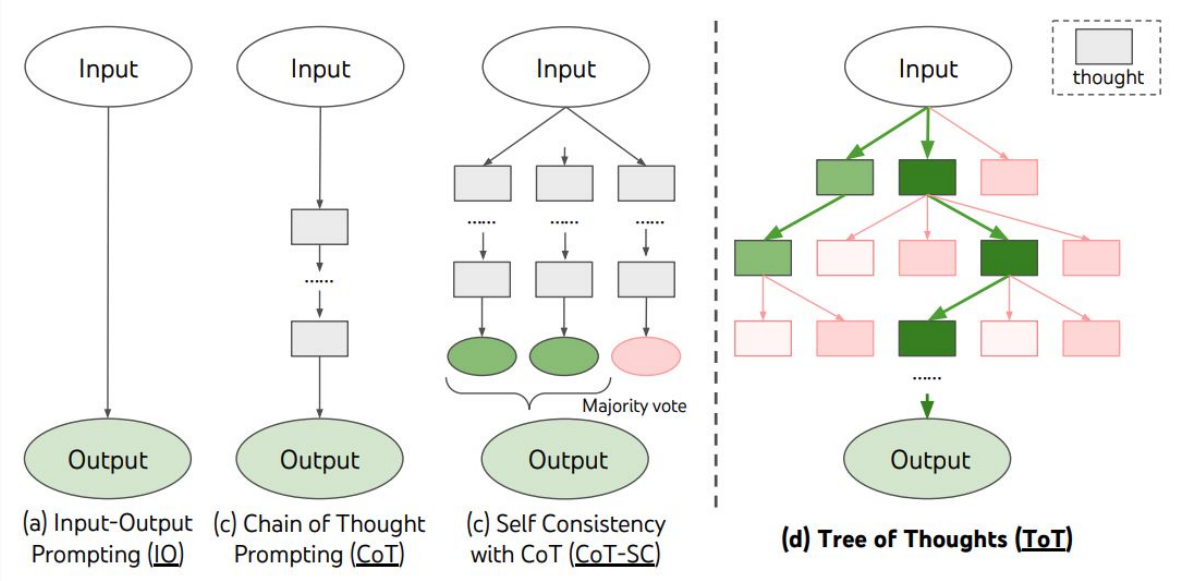
Is Prompting Enough?

Prompting improves:

- Reasoning
- Structure

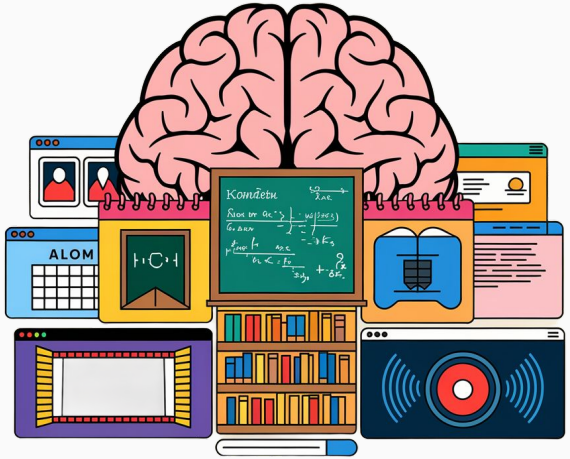
But cannot fix:

- Missing knowledge
- Outdated data
- External facts



What Is Missing?

- Access to external knowledge
- Ability to find relevant information
- Scalable search over large data
- Grounded responses



From Models to Systems

Before:

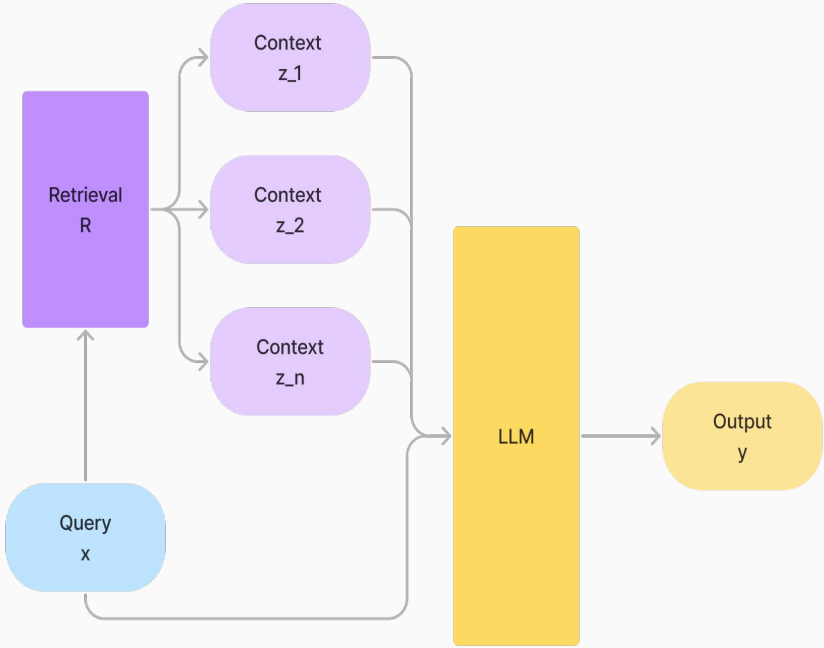
- LLM alone

Now:

- LLM + data + retrieval

Shift:

- Generation → Grounded Generation



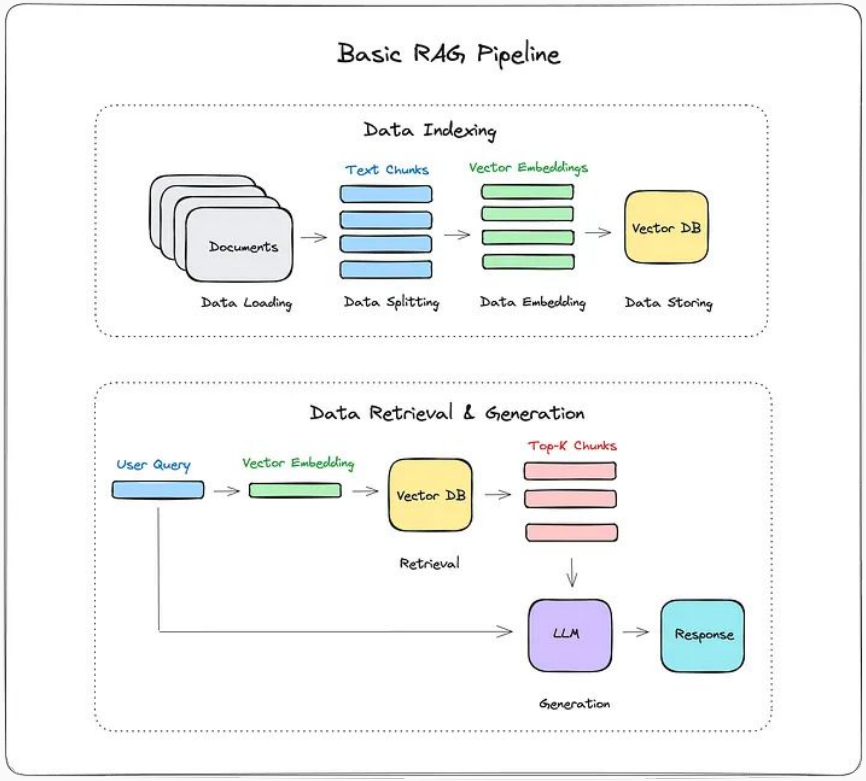
The Solution: Retrieval

- Retrieve relevant documents
- Provide them as context
- Generate grounded answers

Pipeline:

Query → Retrieve → Generate

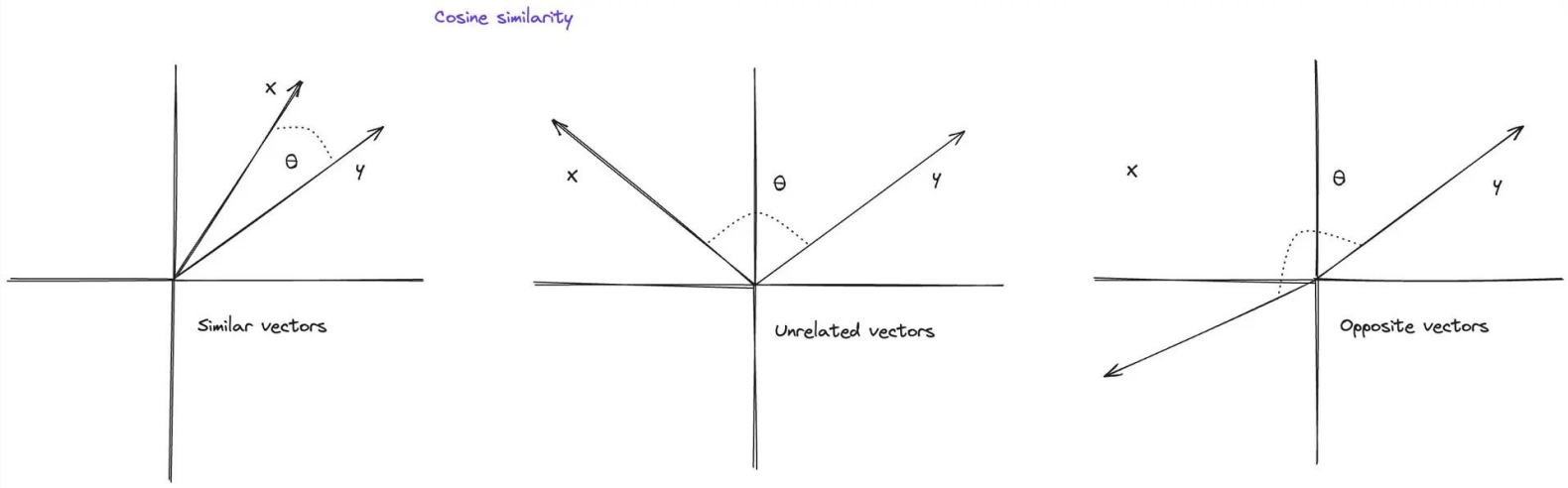
👉 **Retrieval-Augmented Generation (RAG)**



How Retrieval Works

- Convert text into embeddings (vectors)
- Search similar vectors
- Select top-k results

👉 **Similar meaning = close vectors**

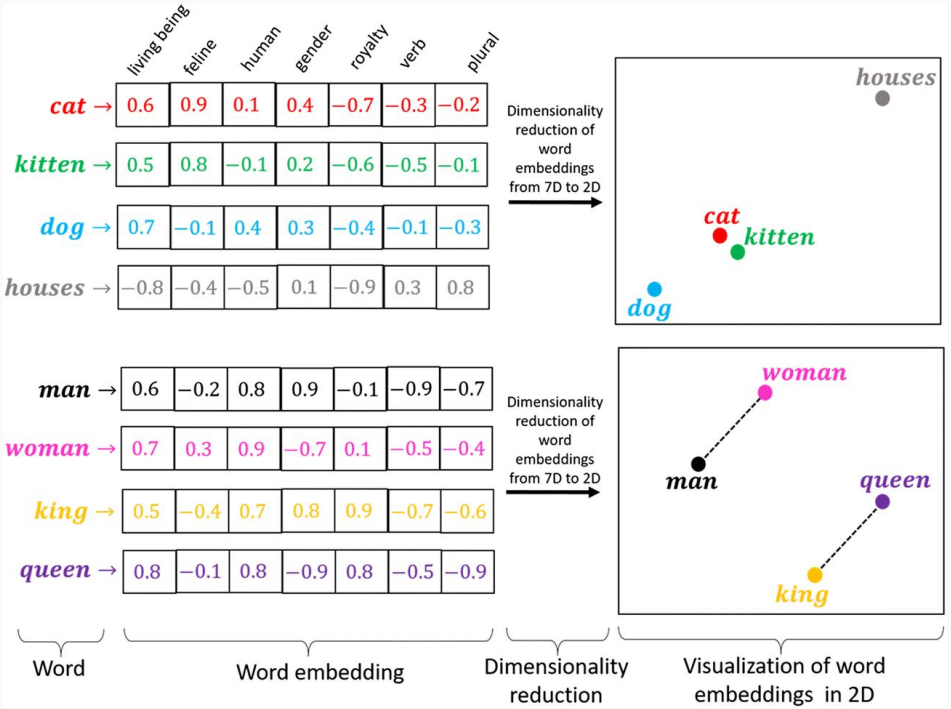


What is an Embedding?

- A numerical representation of text
- Maps words/sentences → vectors
- Captures semantic meaning

Example:

“dog” → [0.12, -0.45, 0.88, …]



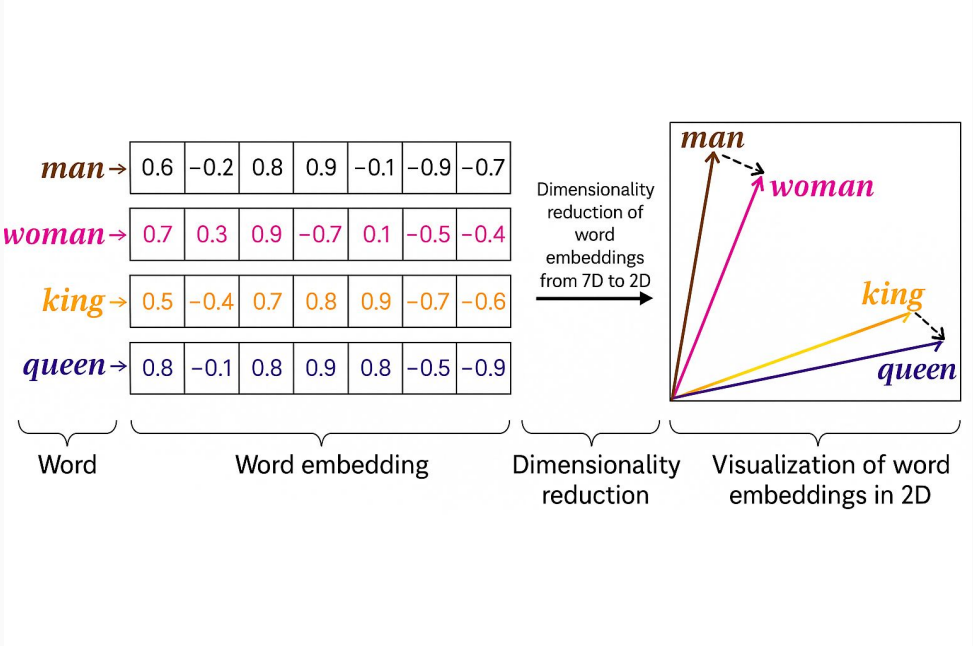
Why Do We Need Embeddings?

- Computers cannot understand raw text
- Need numerical representation
- Enable:
  - Similarity search
  - Clustering
  - Retrieval

Example:
“car” ≈ “vehicle”
“car” ≠ “banana”

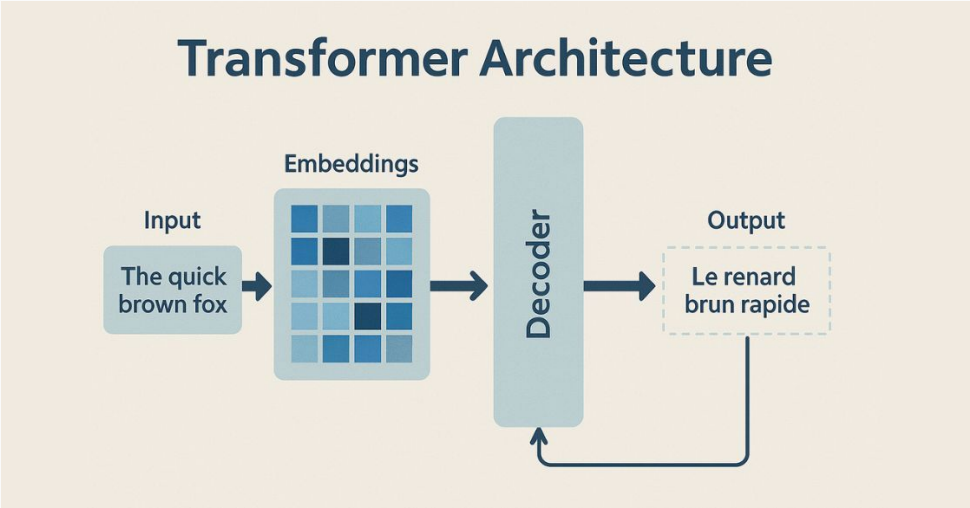
Meaning as Geometry?

- Text → points in vector space
- Similar meaning → close vectors
- Different meaning → far vectors



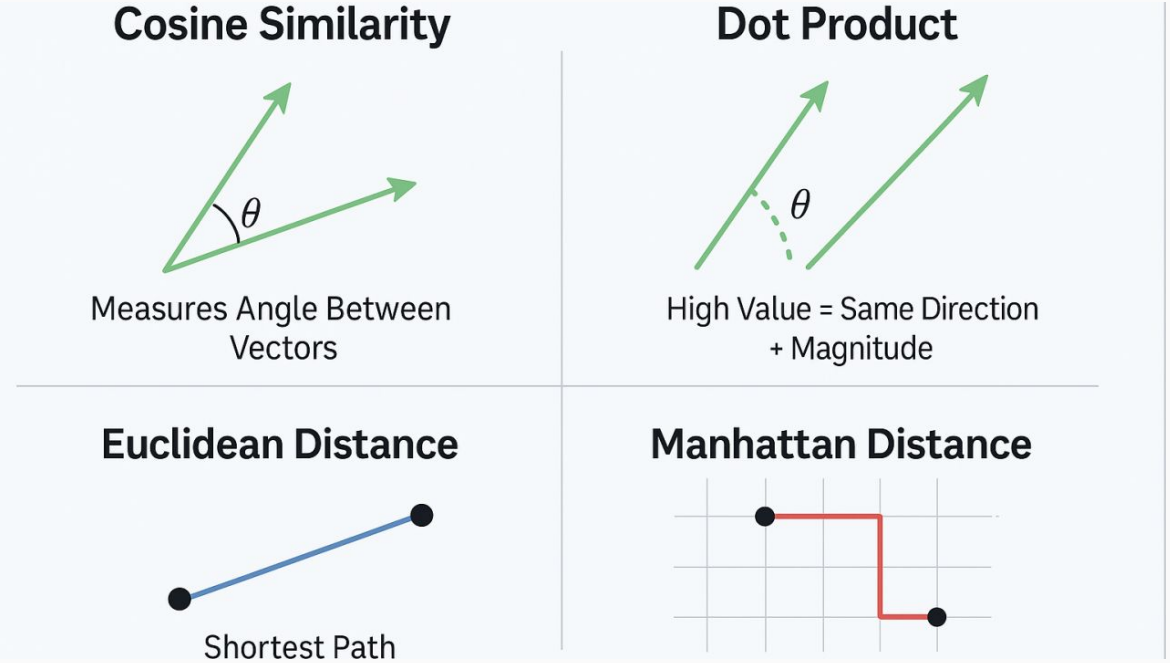
Where Do Embeddings Come From?

- Produced by neural networks
- Word2Vec, Glove, Fasttext
- Often from transformer models
- Same idea as LLM internal representations



Measuring Similarity

- Cosine similarity (most common)
  - Measures angle between vectors
  - Smaller angle = more similar
- Dot product
- Euclidean distance



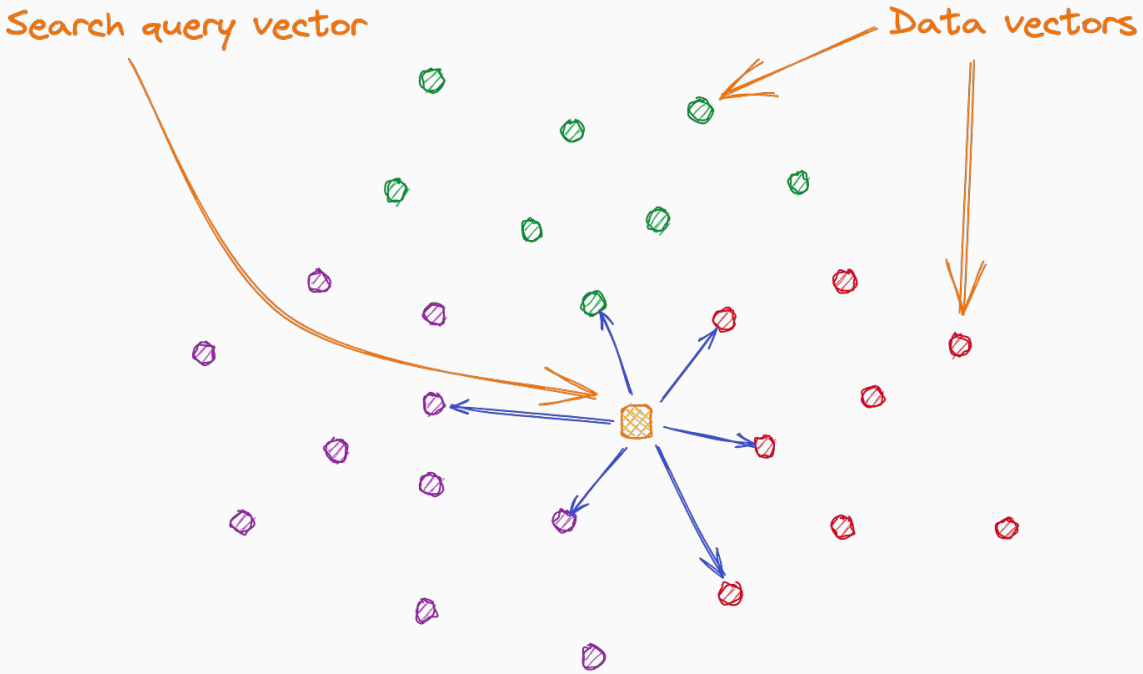
Example: Semantic Similarity

Query:
"I want to buy a car"

Documents:

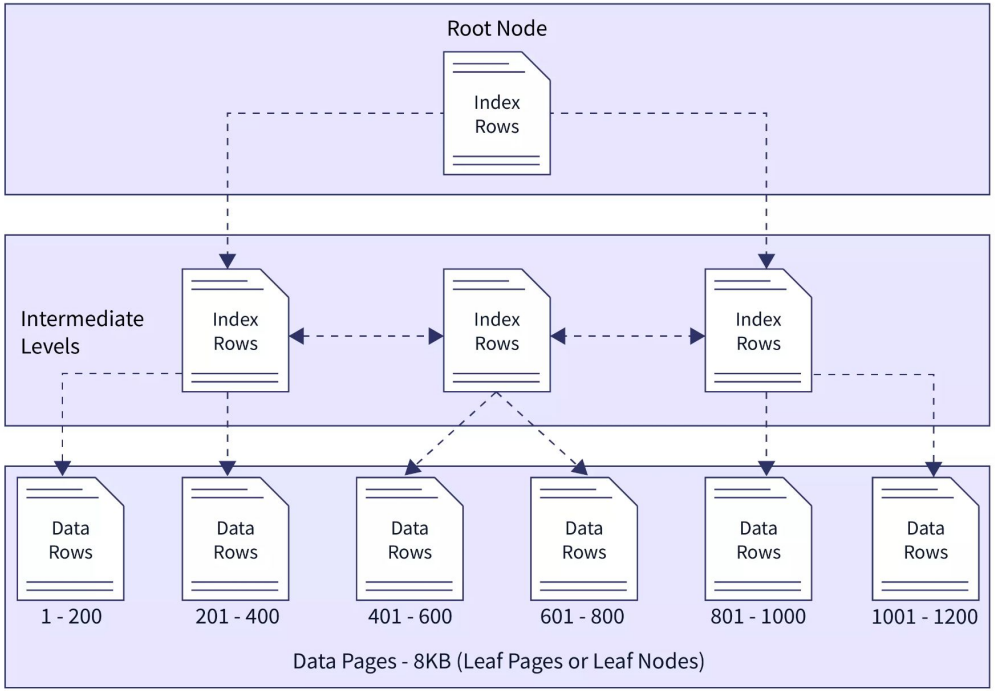
- "Best vehicles for families" ✅
- "How to cook pasta" ❌
- "Top electric cars 2025" ✅

Result:
Closest vectors = most relevant



Document Indexing

- Large collections of documents
- Millions of embeddings
- Need fast search
- Brute force = too slow
- Efficient retrieval requires indexing



What is Indexing?

- Data structure for fast search
- Organizes embeddings efficiently
- Enables quick similarity lookup

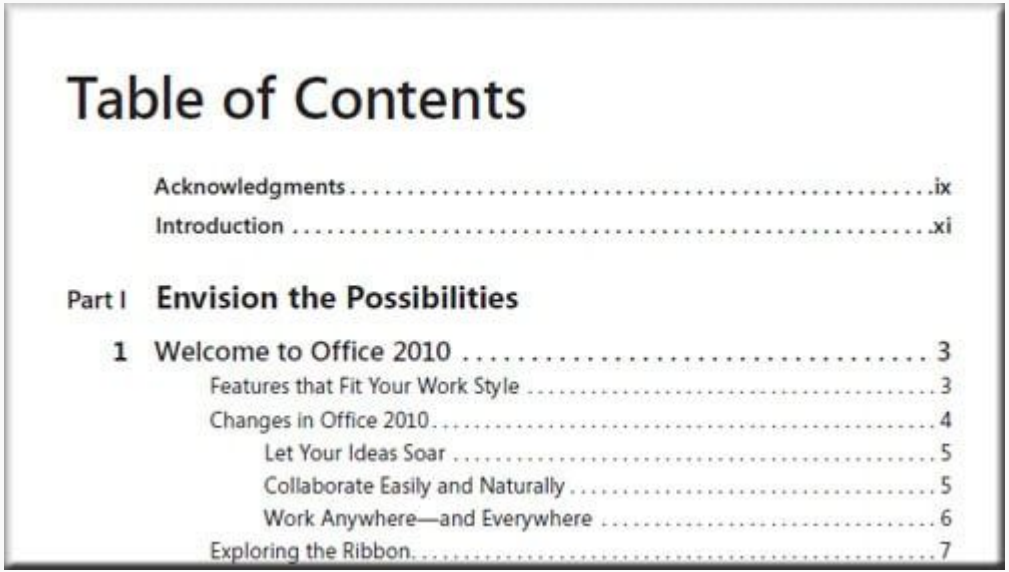


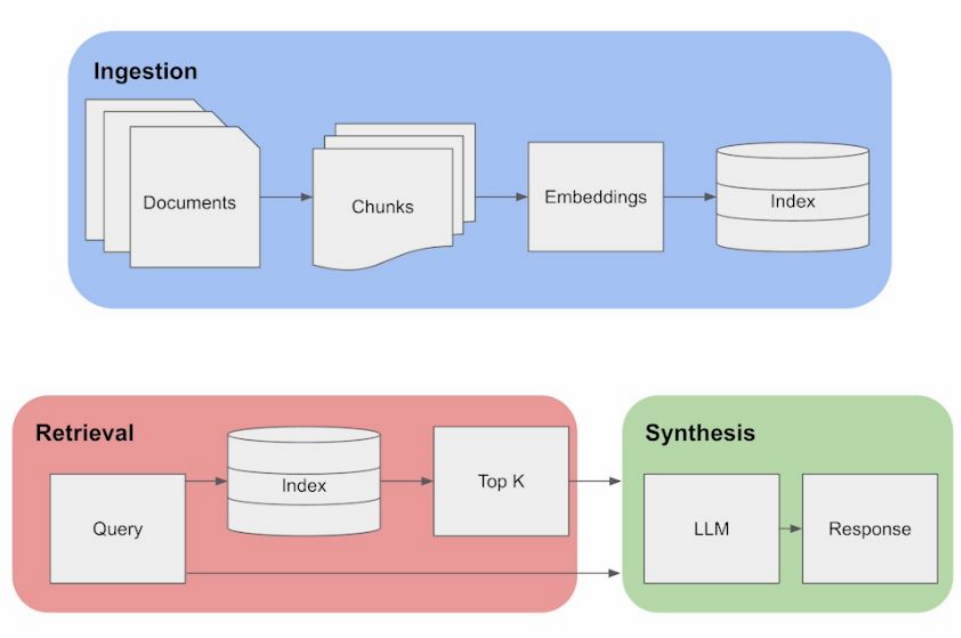
Table of Contents

Building the Index

Steps:

1. Collect documents
2. Split into chunks
3. Convert to embeddings
4. Store in index

Text → Chunks → Vectors → Index



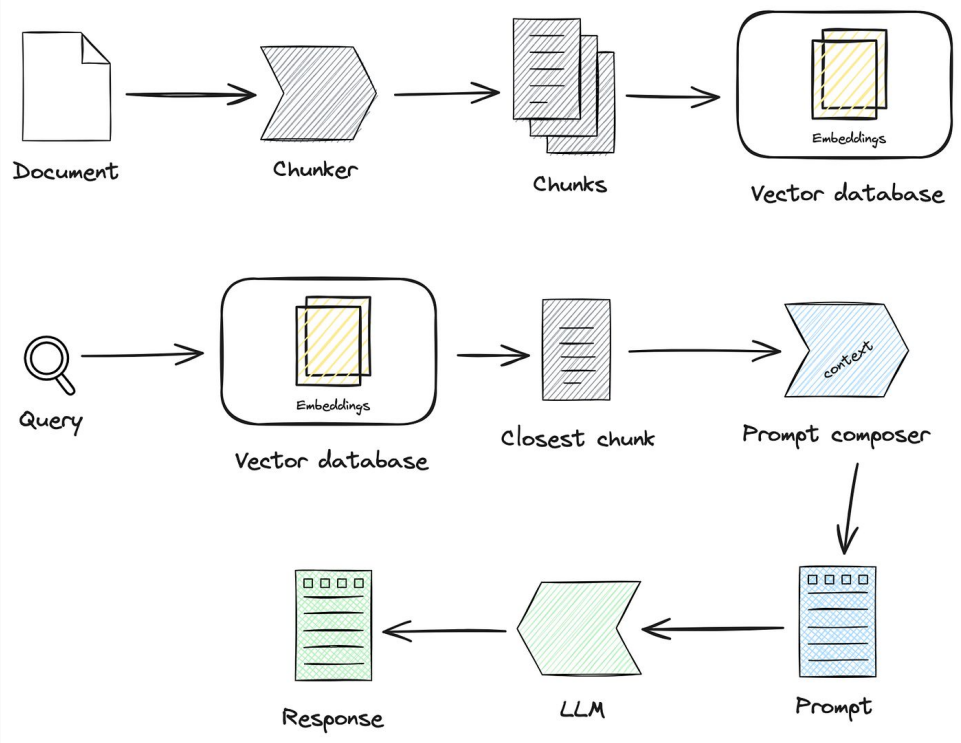
Chunking

- Documents can be very long
- LLM context is limited
- Smaller chunks = better retrieval

Example:

- Full document ❌
- Paragraphs / sections ✅

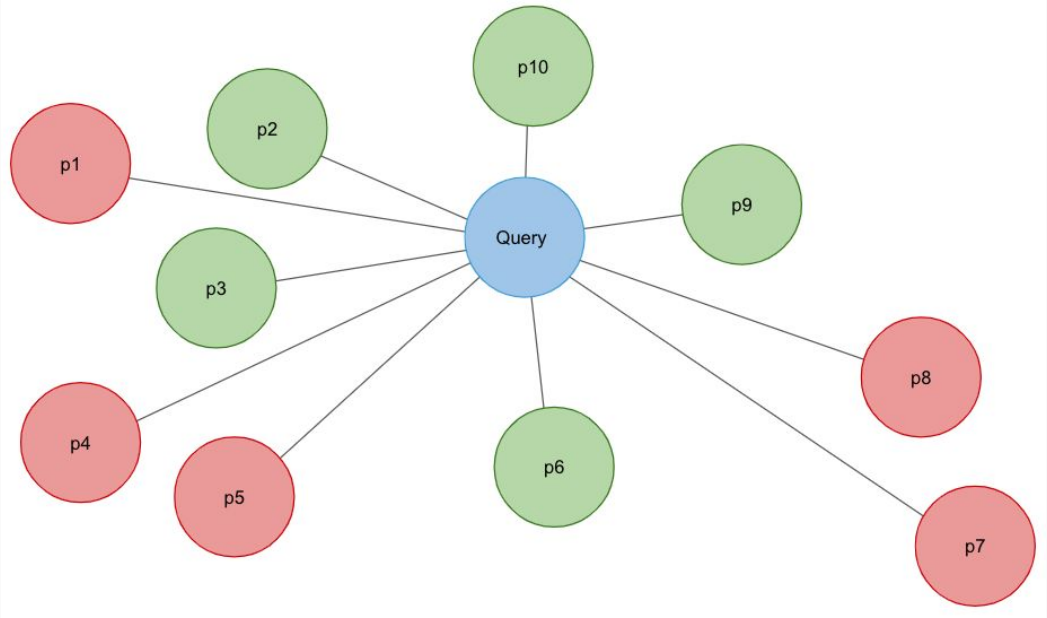
Retrieve relevant pieces, not entire documents



Indexing Methods

- Exact search:
  - Compare with all vectors
- Approximate search (ANN):
  - Faster
  - Slightly less accurate

Key idea:
Trade-off:
Speed vs Accuracy

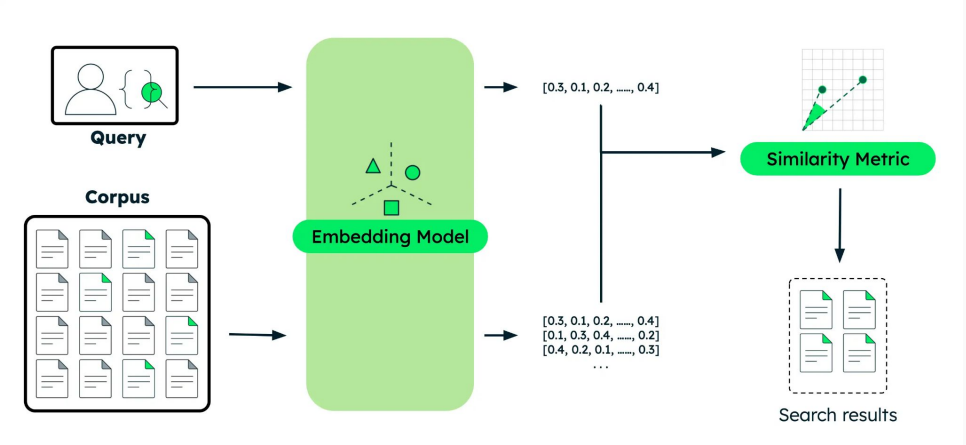


The Retrieval Step

- Input: query
- Convert query → embedding
- Search in vector index
- Return most similar results

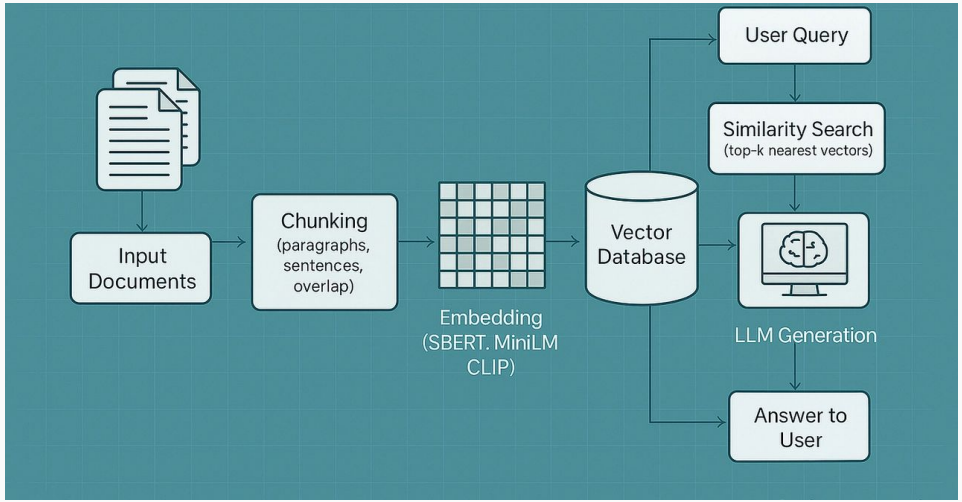
Pipeline:

Query → Embedding → Search → Results



Top-k Retrieval

- Retrieve k most similar results
- k = number of documents returned
- Common values: k = 3, 5, 10

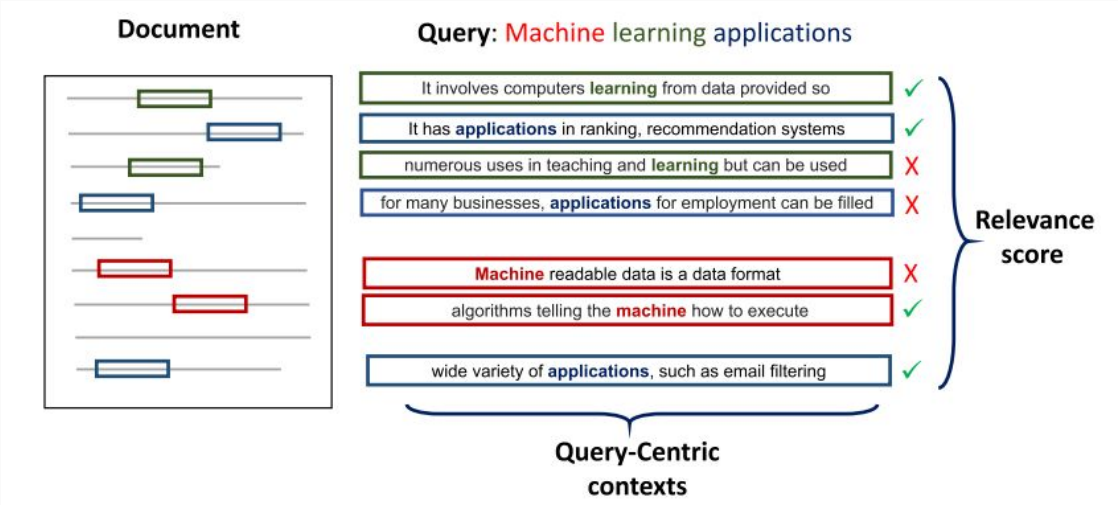


Problem: Retrieval is Not Perfect

- Similarity ≠ relevance
- Embeddings are approximate
- Important documents may be ranked lower

Example:

Correct doc ranked #5 instead of #1



Reranking

- Second stage after retrieval
- Re-evaluate top-k results
- Improve ordering

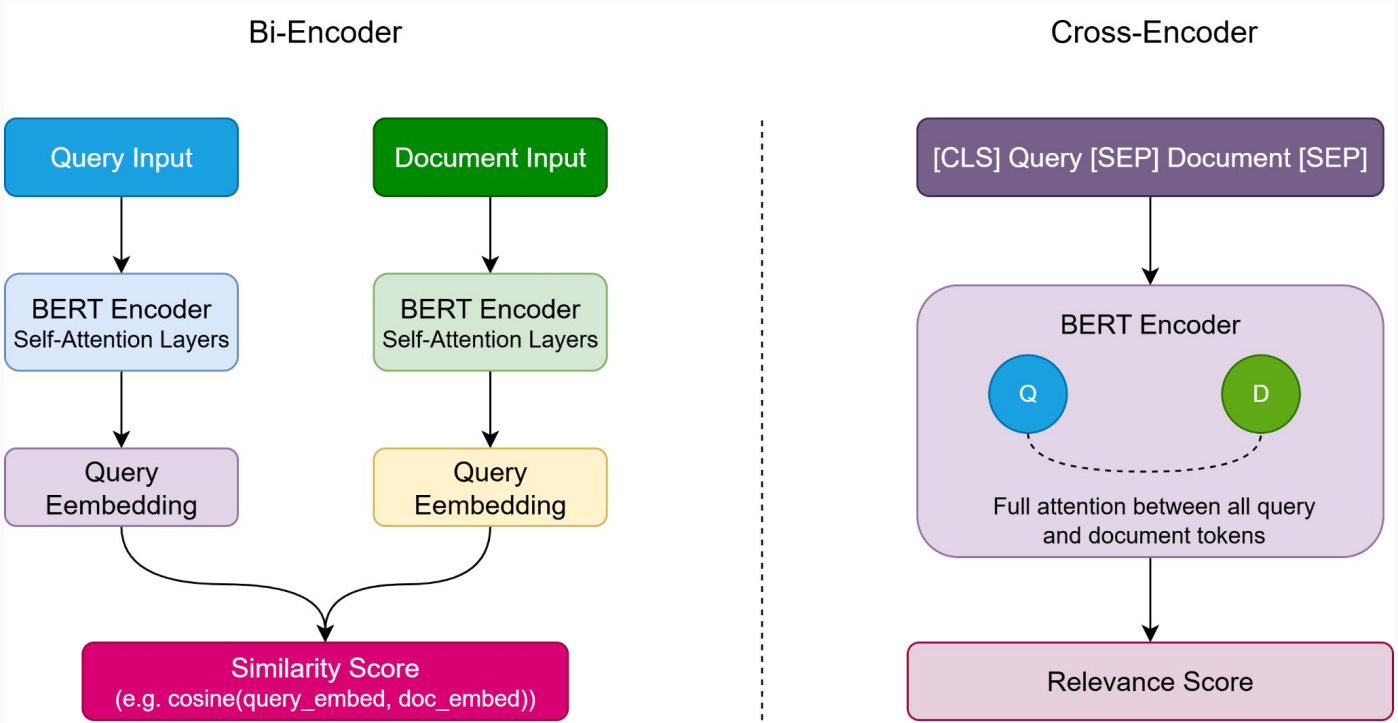
Pipeline:
Retrieve → Rerank → Final results

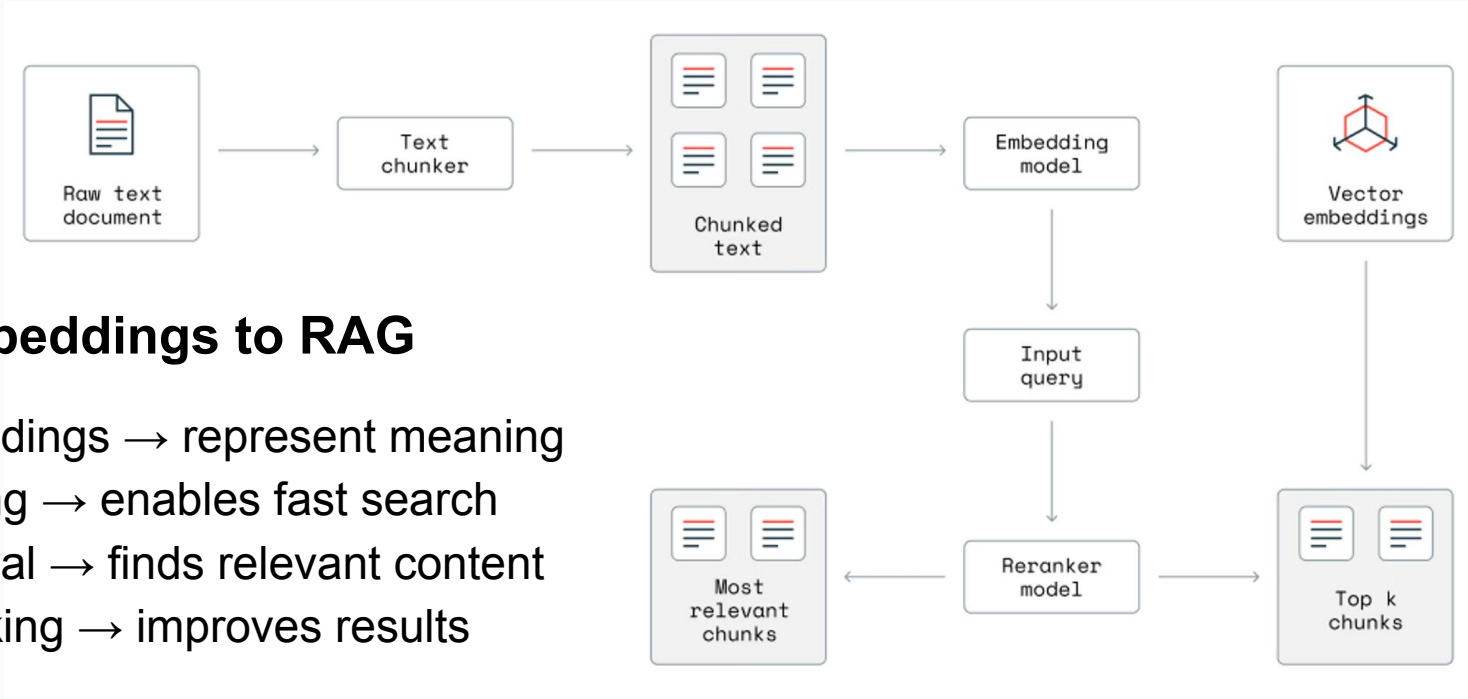
How Does Reranking Work?

- Compare query + document together
- More precise evaluation
- Often uses stronger models

Types:

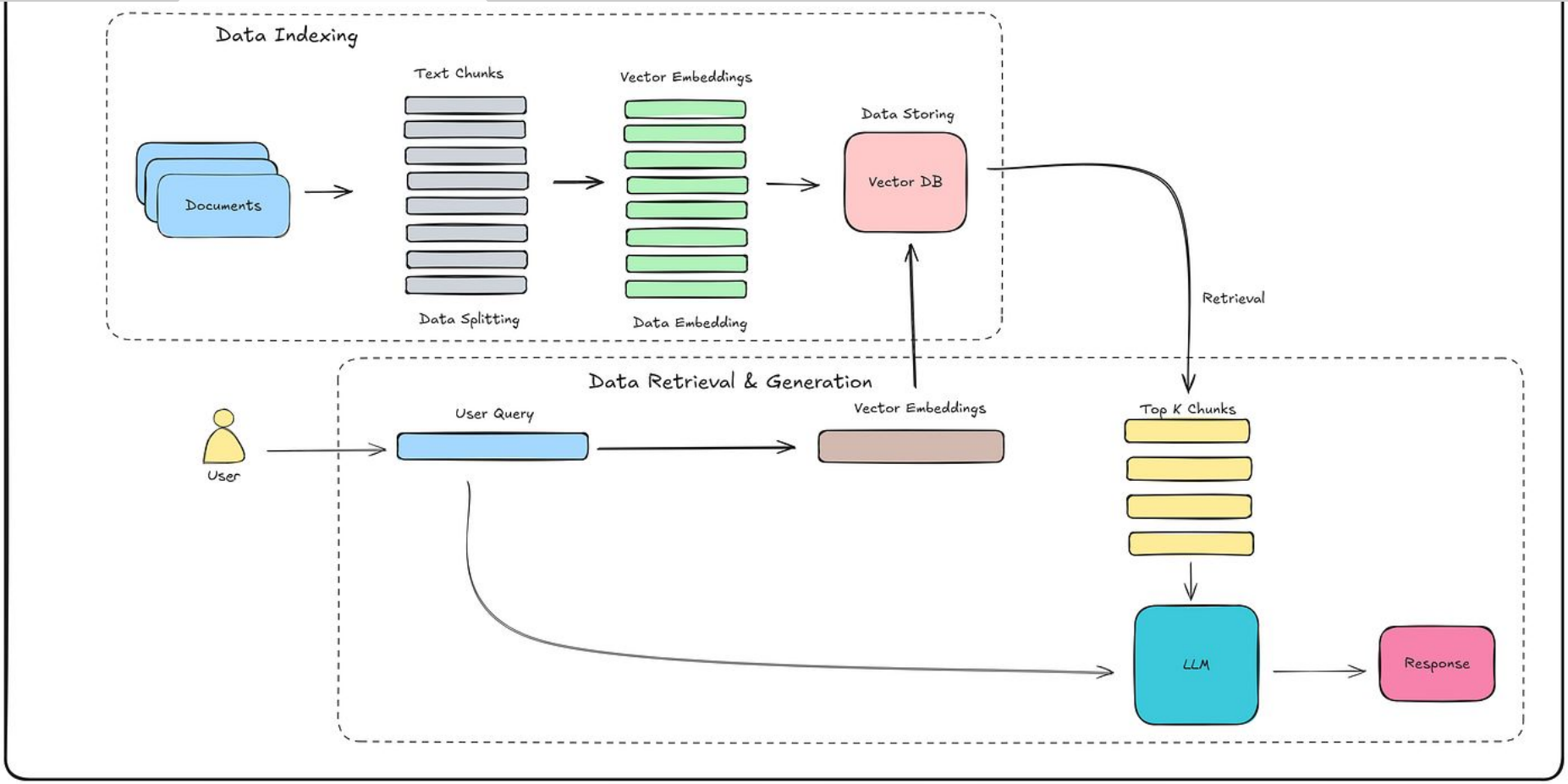
- Cross-encoder
- LLM-based scoring


Bi-Encoder
Query Input
Document Input
BERT Encoder
Self-Attention Layers
BERT Encoder
Self-Attention Layers
Query
Eembedding
Query
Eembedding
Similarity Score
(e.g. cosine(query_embed, doc_embed))
Cross-Encoder
[CLS] Query [SEP] Document [SEP]
BERT Encoder
Q
D
Full attention between all query
and document tokens
Relevance Score



From Embeddings to RAG

- Embeddings → represent meaning
- Indexing → enables fast search
- Retrieval → finds relevant content
- Reranking → improves results

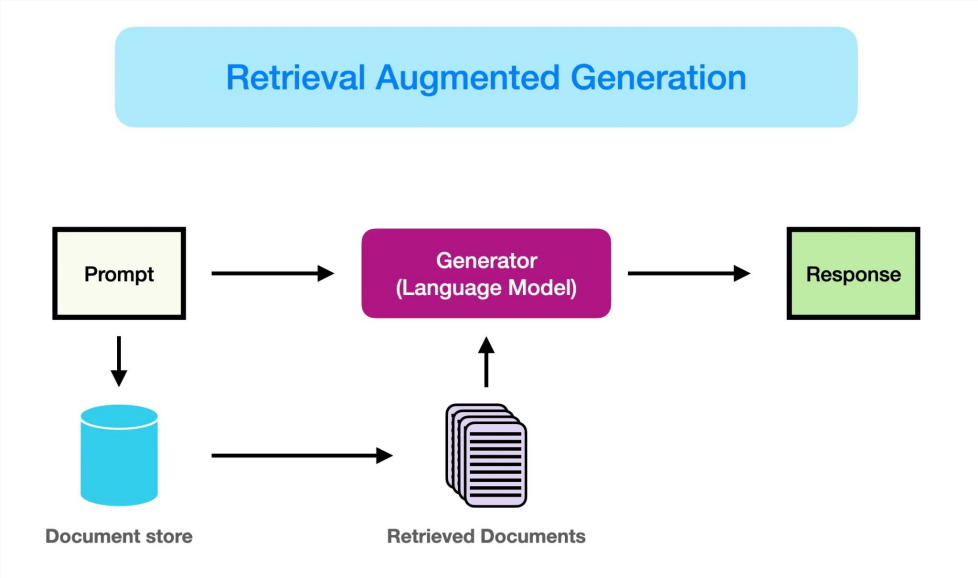

Data Indexing
Text Chunks
Vector Embeddings
Data Storing
Documents
Vector DB
Data Splitting
Data Embedding
Retrieval
Data Retrieval & Generation
User Query
Vector Embeddings
Top K Chunks
User
LLM
Response

Prompting with Retrieved Context

Example Prompt:
"Answer the question using the context below:
[Retrieved documents]
Question: …"

Key idea:

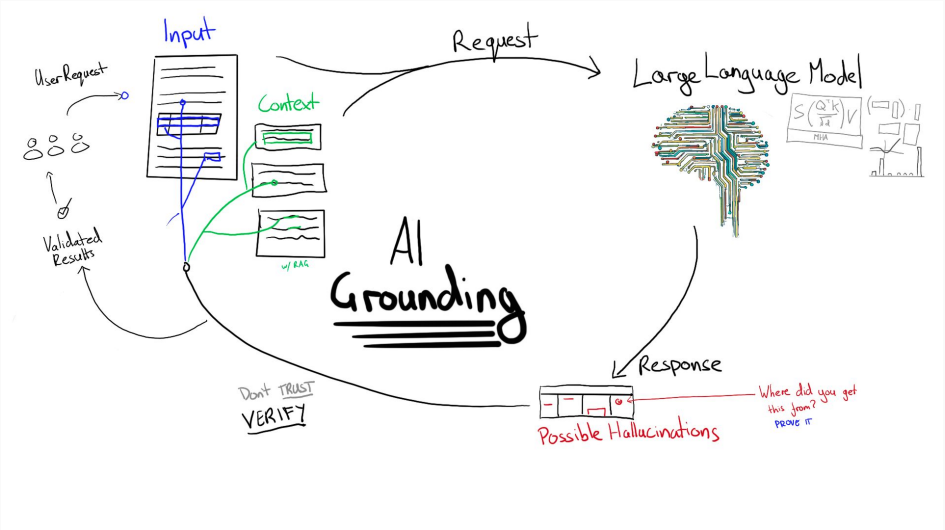
- Context = external knowledge
- Prompt = control reasoning



Advantages of using RAG

- Reduces hallucinations
- Provides up-to-date knowledge
- Handles large document collections
- Improves factual accuracy

👉 **Grounded generation**



Limitations of RAG

- Depends on retrieval quality
- Poor chunks → poor answers
- Higher latency
- More complex systems

Summary

- Prompting → improves reasoning
- Embeddings → represent meaning
- Retrieval → finds knowledge
- RAG → combines everything