# Natural Language Processing Part 1 : Tokenization, Embeddings

## Natural Language Processing Part 2 : RAG

## RAG - Retrieval Augmented Generation

- RAG is a pipeline. We first search for information, then we generate an answer using LLM and Prompts defined
- **Retrieval**
  - The system first **searches for relevant information** from external sources such as documents or databases (vector database - > java pdf doc)
  - Similar to a Google search step
  - Ranks results based on relevance (using metrics such has cosine similarity)
  - Finds relevant content from documents
- **Augmentation** - Retrieved information is augmented into the prompt using prompt engineering  - > <u>retrieved context</u>
- **Generation** - The large language model then generates an answer using both the question and retrieved context

## Why RAG Exists

- LLMs do not know private or new data
- LLMs have limited context size
- Large documents cannot fit into prompts
- RAG solves knowledge and context limitations

# Chatbot vs RAG System

- Chatbots rely only on model knowledge
- They do not fetch external information
- RAG retrieves information before answering
- RAG combines search with generation

## Documents as Knowledge Sources

- PDFs, text files, web pages
- Internal documents and manuals
- RAG works on stored documents
- Documents are the source of truth

## What and Why Chunking is Needed

- Chunks are part of documents
- LLMs cannot read large documents at once
- Documents must be split into chunks
- Chunk size impacts answer quality
- Good chunking improves retrieval

## What Are Embeddings?

- Embeddings convert text into numbers
- They represent meaning, not words
- Similar meaning gives similar vectors
- Used for semantic search

## Why Embeddings Matter in RAG

- Enable semantic search instead of keyword search
- Help find relevant document chunks
- Power similarity search in vector databases
- Foundation of the retrieval step

## Similarity Search

- User query is converted into an embedding
- Compared with document embeddings
- Closest matches are retrieved
- Enables semantic matching

# Top-K Retrieval

- Retrieve only top few relevant chunks
- Too much data causes confusion
- Common values are Top 3 or Top 5
- Balances precision and noise

## Hallucination Problem

- LLMs may generate incorrect answers
- Lack of context increases hallucinations
- RAG provides grounding information
- Guardrails improve trust

## Prompt Structure in RAG

- System instructions define behavior
- Retrieved context is injected
- User question is answered last
- Order of prompt matters

## Cost and Performance Awareness

- Embeddings and tokens have cost
- Large context increases expense
- Caching improves performance
- Efficient retrieval saves money

## What RAG Is NOT

- RAG is not fine-tuning
- RAG does not train models
- RAG does not store answers
- It only retrieves information

## When to Use RAG

- Best for large or private knowledge
- Useful for documentation Q&A
- Not needed for simple chatbots
- Choose RAG wisely

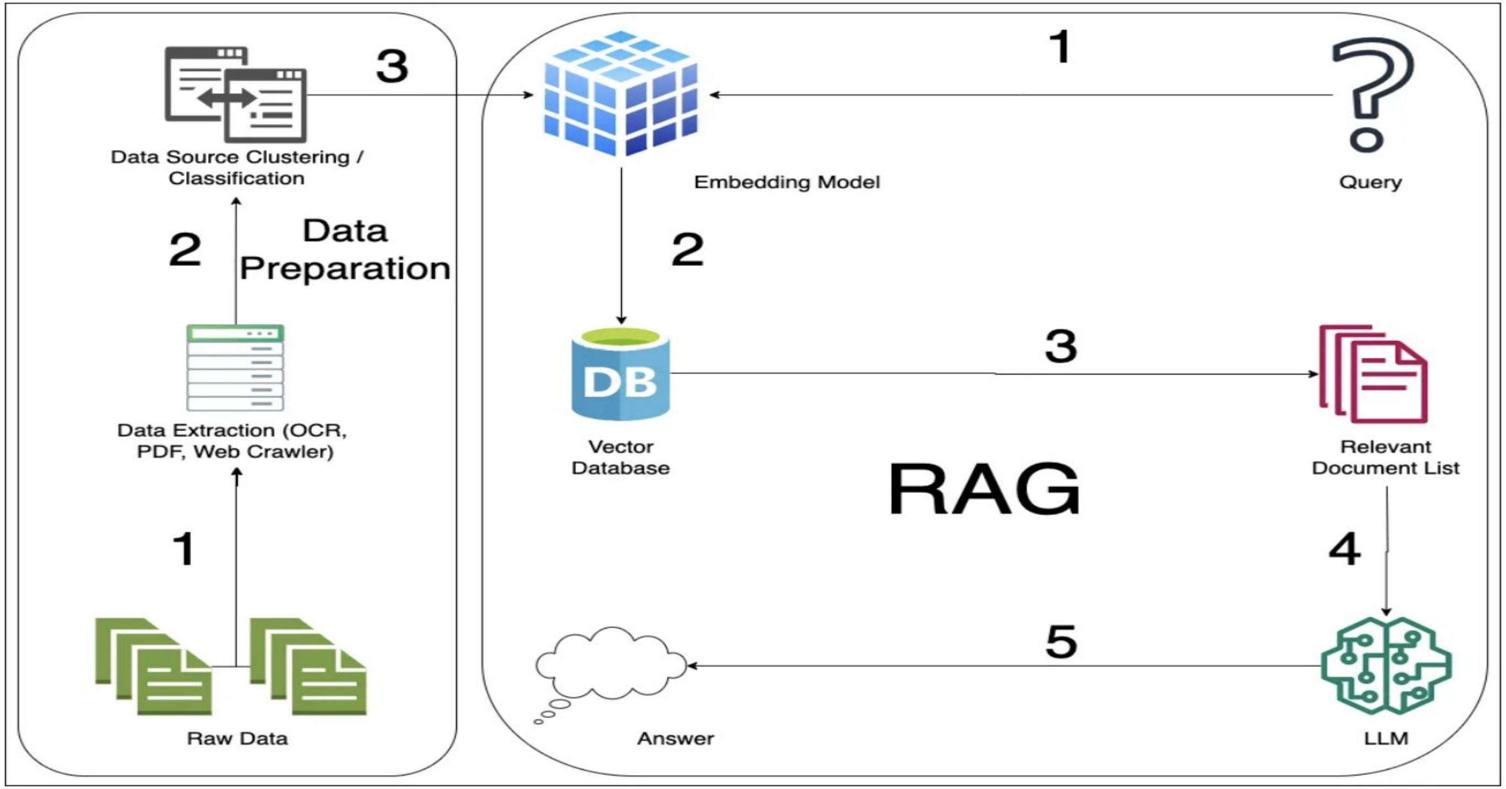
# RAG Examples

- Document QA - Question and Answer for any long documents

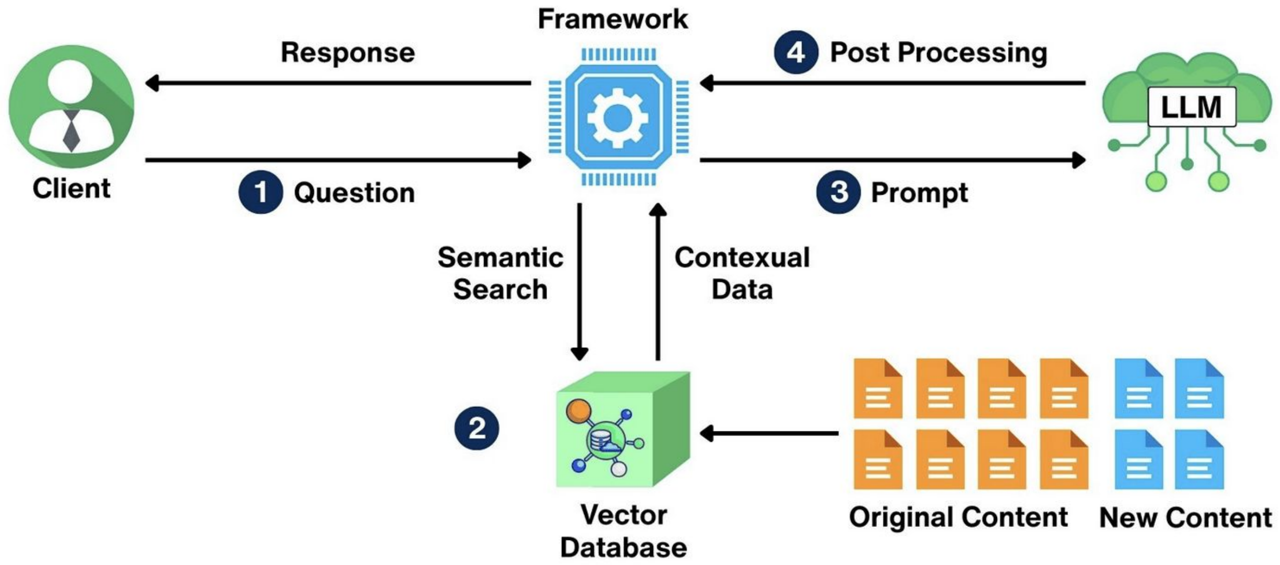
## RAG Architecture (High Level)

- User asks a question
- Query is converted into an embedding
- Relevant document chunks are retrieved
- Retrieved context is sent to the LLM
- LLM generates the final answer

# RAG – Architecture – Part 1



# RAG – Architecture – Part 2


Framework
Response
4 Post Processing
LLM
Client
1 Question
3 Prompt
Semantic Search
Contexual Data
2
Vector Database
Original Content
New Content