MC05 - Large Language Models and Agents

Short Course - JCD 2026

Bruno Menezes (LNCC), Eduardo Bezerra (CEFET/RJ)

Outline

From LLMs to LLM-based Agents

From Prompting Techniques to Interaction Patterns

Tool Calling

Retrieval-Augmented Generation

From LLMs to LLM-based Agents

From LLMs to LLM-based Agents

An LLM is, in essence, a **probabilistic text generator**: it predicts the next token, given an input token sequence.

From LLMs to LLM-based Agents

An LLM is, in essence, a **probabilistic text generator**: it predicts the next token, given an input token sequence.
Yet at scale, LLMs exhibit reasoning-like abilities (planning, inference, language understanding).

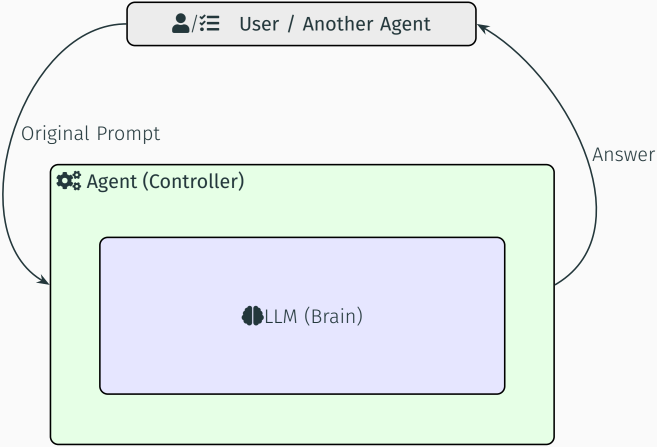
From LLMs to LLM-based Agents

An LLM is, in essence, a **probabilistic text generator**: it predicts the next token, given an input token sequence.
Yet at scale, LLMs exhibit reasoning-like abilities (planning, inference, language understanding).

An LLM-based agent combines:

- These **reasoning-like abilities** of the LLM,
- **Control logic** to orchestrate steps and manage context,
- Access to **external tools** (search, databases, APIs),
- Mechanisms for **perception and action**, so reasoning can affect the real world.

LLM-based Agent



From Prompting Techniques to Interaction Patterns

Prompting Techniques: Examples

Zero-shot

Prompt:

```
What is 47 + 35?
```

Output:

```
82
```

Few-shot

Prompt:

```
Examples:
12 + 7 = 19
5 + 9 = 14
23 + 18 = 41

Now, what
is 47 + 35?
```

Output:

```
82
```

Chain-of-Thought

Prompt:

```
What is 47 + 35?
Let's think
step by step.
```

Output:

```
First, add tens:
40 + 30 = 70
Then, ones:
7 + 5 = 12
Sum:
70 + 12 = 82
Answer: 82
```

From Prompting to Patterns

- Prompting techniques (Zero-shot, Few-shot, Chain-of-Thought) steer a **single LLM call**:
- But complex tasks require multi-step reasoning and tool use.

From Prompting to Agents

Examples of complex tasks that require multi-step reasoning and tool use:

- Answering questions over a large knowledge base (search + reasoning)
- Text-to-SQL (NL → SQL query → execution → formatted result)
- Planning a trip (dates, flights, hotels, budget)
- Solving math word problems (step-by-step reasoning + calculation)
- Data analysis (retrieve data → transform → summarize)

Interaction Patterns

Multi-step reasoning and tool use demand the use of **interaction patterns**.

Interaction Patterns

Multi-step reasoning and tool use demand the use of **interaction patterns**.

In the context of LLMs, an interaction pattern refers to the structured way a user or a system communicates with a model to achieve a specific outcome. Rather than just asking a single question, these patterns define the ”flow” of reasoning, the role the AI plays, and how it processes information to reach a solution.

Interaction Patterns

- **Interaction patterns** structure reasoning + action.
- Ensure consistency and integration with external tools.

Interaction Patterns

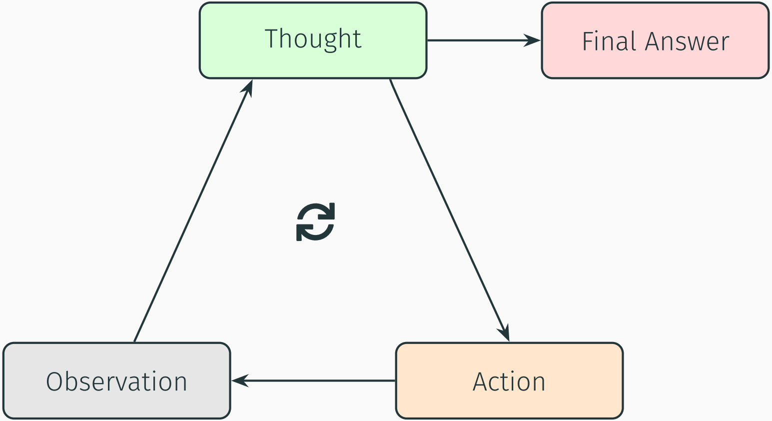
- **Interaction patterns** structure reasoning + action.
- Ensure consistency and integration with external tools.
- Examples:
  - ReAct (Reason + Act)
  - Plan-and-Act
  - Reflexion

Interaction Patterns

- **Interaction patterns** structure reasoning + action.
- Ensure consistency and integration with external tools.
- Examples:
  - ReAct (Reason + Act)
  - Plan-and-Act
  - Reflexion
- We focus on ReAct.

ReAct: Synergizing Reasoning and Acting in Language Models [**?**]

ReAct Pattern



The agent **thinks**, **acts**, and **observes** in a loop,
until enough information is gathered to deliver the final answer.

Example: Finding Payroll for Sales

- Scenario: A user asks for the total payroll expense of the Sales department for “last month”.

Example: Finding Payroll for Sales

- Scenario: A user asks for the total payroll expense of the Sales department for "last month".
- Why this is non-trivial: "Last month" is *relative* and depends on the current date.

Example: Finding Payroll for Sales

- Scenario: A user asks for the total payroll expense of the Sales department for “last month”.
- Why this is non-trivial: “Last month” is *relative* and depends on the current date.
- Data requirement: We must query a payroll/financial database filtered by
    - month + year (resolved from “last month”)
    - department = `Sales`

Example: Finding Payroll for Sales

- Scenario: A user asks for the total payroll expense of the Sales department for "last month".
- Why this is non-trivial: "Last month" is *relative* and depends on the current date.
- Data requirement: We must query a payroll/financial database filtered by
  - month + year (resolved from "last month")
  - department = `Sales`
- Goal: Return a single, clear number (the total payroll expense) with the correct time reference.

The next slide shows a step-by-step trace (Thought $\rightarrow$ Action $\rightarrow$ Observation) ending in the final answer.

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was "last month" relative to today (February 2026).

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()
- Observation: Today is February 10, 2026. ”Last month” is January 2026.

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()
- Observation: Today is February 10, 2026. ”Last month” is January 2026.
- Thought: Now I need to query the financial database for January 2026 payroll expenses specifically filtered for the ”Sales” department.

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()
- Observation: Today is February 10, 2026. ”Last month” is January 2026.
- Thought: Now I need to query the financial database for January 2026 payroll expenses specifically filtered for the ”Sales” department.
- Action: query_payroll_db(month=”January”, year=2026, department=”Sales”)

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()
- Observation: Today is February 10, 2026. ”Last month” is January 2026.
- Thought: Now I need to query the financial database for January 2026 payroll expenses specifically filtered for the ”Sales” department.
- Action: query_payroll_db(month=”January”, year=2026, department=”Sales”)
- Observation: Result: $142,500.00.

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()
- Observation: Today is February 10, 2026. ”Last month” is January 2026.
- Thought: Now I need to query the financial database for January 2026 payroll expenses specifically filtered for the ”Sales” department.
- Action: query_payroll_db(month=”January”, year=2026, department=”Sales”)
- Observation: Result: $142,500.00.
- Thought: I have the information. I can now answer the user.

Example: Finding Payroll for Sales

- Thought: I need to find the payroll data for the Sales department. First, I need to know which month was ”last month” relative to today (February 2026).
- Action: get_current_date()
- Observation: Today is February 10, 2026. ”Last month” is January 2026.
- Thought: Now I need to query the financial database for January 2026 payroll expenses specifically filtered for the ”Sales” department.
- Action: query_payroll_db(month=”January”, year=2026, department=”Sales”)
- Observation: Result: $142,500.00.
- Thought: I have the information. I can now answer the user.
- Final Response: The total payroll expense for the Sales department in January 2026 was $142,500.00.

Example: Finding Payroll for Sales

But how the agent knows <u>what</u> tools it can invoke?!

Example: Finding Payroll for Sales

But how the agent knows <u>what</u> tools it can invoke?!

And how the agent knows <u>how</u> and <u>when</u> to invoke them?!

Example: Finding Payroll for Sales

But how the agent knows what tools it can invoke?!

And how the agent knows how and when to invoke them?!

To answer these questions, we need to talk about the mechanism of tool calling.

Tool Calling

Tool Calling

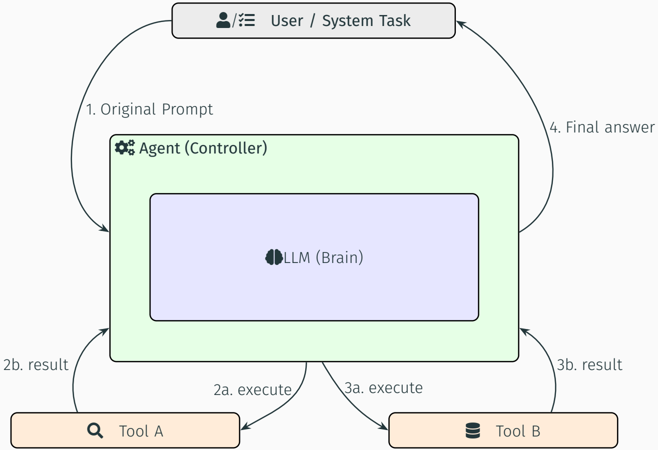
Multi-step tasks often require **external actions**, not just text generation.

Tool Calling

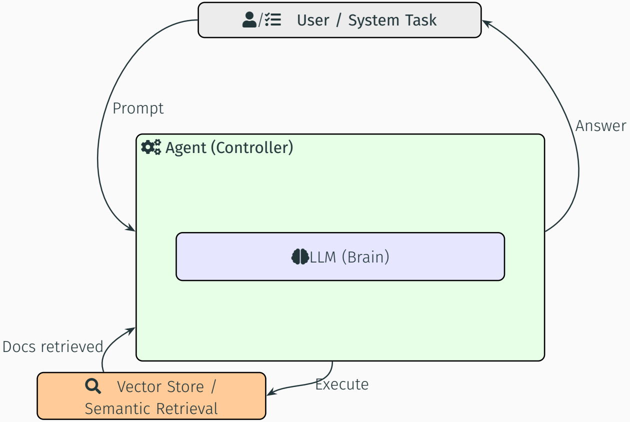
Multi-step tasks often require **external actions**, not just text generation.

In the context of LLMs, tool calling refers to the capability of a model to invoke external functions (tools) — such as APIs, databases, calculators, or search engines — as part of its reasoning process. Instead of relying only on internal knowledge, the model can retrieve fresh data, perform computations, and interact with systems to produce grounded and verifiable answers.

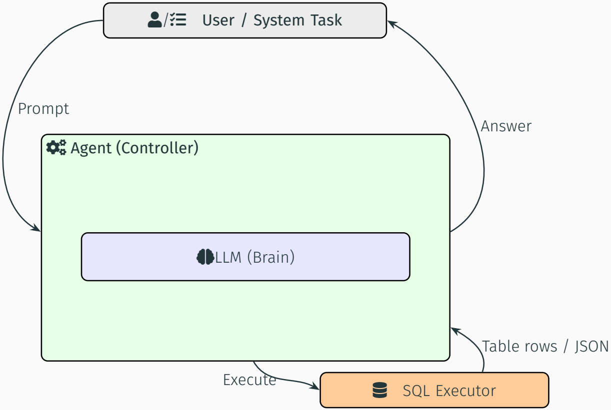
Tool Calling



Tool Calling Instance: RAG



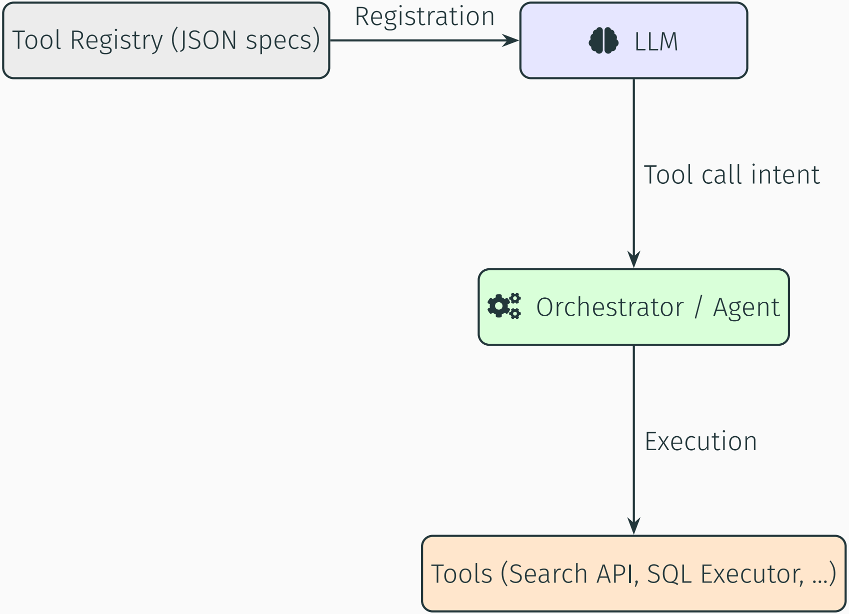
Tool Calling Instance: Text-to-SQL



Tool Registration?

- LLMs do not “magically” know what tools exist.
- Each tool must be **registered** via a structured description that includes:
  - **Name** and **purpose** of the tool,
  - **Input parameters** (types, constraints, defaults),
  - **Expected outputs**.
- This information is added to the LLM context, so the model can decide when and how to use the tool.
- Tool registration is the gateway for RAG, Text-to-SQL, and many other applications.

Tool Registration?



Demo Time

Retrieval-Augmented Generation

Definition

Retrieval-Augmented Generation (RAG) is a framework that improves language model outputs by retrieving relevant external documents and injecting them into the prompt, so the model can generate grounded and evidence-based responses.

Motivation

- LLMs may **hallucinate** when training data is insufficient, outdated, or domain-specific.
- RAG grounds model outputs in **external, authoritative sources**.
- Aims at (1) reducing unsupported generated content and (2) enabling handling of queries beyond the pretraining corpus.

A typical RAG pipeline

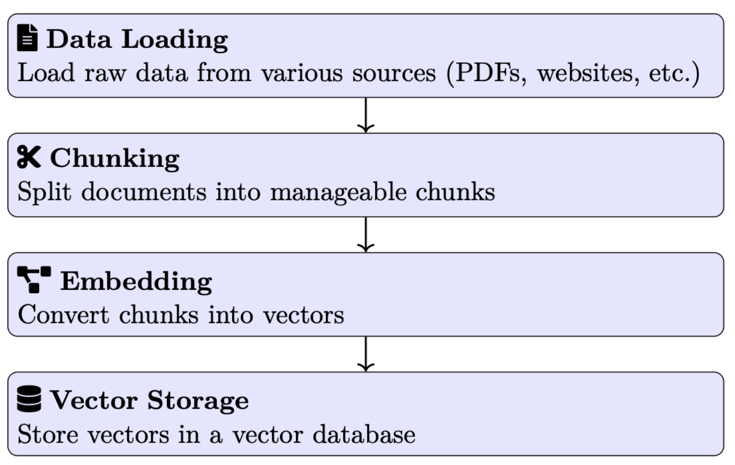
- Break available documents into pieces and index them.
- Retrieve relevant pieces or passages at query time.
- Inject retrieved chunks into the LLM’s context window before generation.

A typical RAG pipeline

- Break available documents into pieces and index them.
- Retrieve relevant pieces or passages at query time.
- Inject retrieved chunks into the LLM's context window before generation.

Goal: generate output that combines fluency of the LLM with evidence-based retrieval.

Phase 1: Indexing (Offline)



Data Loading

Goal: ingest raw content from multiple sources and convert it into a clean, consistent **document collection** for downstream chunking and embedding.

Data Loading

Goal: ingest raw content from multiple sources and convert it into a clean, consistent **document collection** for downstream chunking and embedding.

- **Typical inputs:**
    - PDFs (reports, regulations, manuals)
    - Web pages (HTML, intranet portals)
    - Office files (DOCX, PPTX)
    - Plain text (TXT, Markdown)

Data Loading

Goal: ingest raw content from multiple sources and convert it into a clean, consistent **document collection** for downstream chunking and embedding.

- **Typical inputs:**
    - PDFs (reports, regulations, manuals)
    - Web pages (HTML, intranet portals)
    - Office files (DOCX, PPTX)
    - Plain text (TXT, Markdown)
- **Key outputs:**
    - extracted text (and optionally tables/images)
    - metadata (source, title, date, author, URL, page number)
    - a standardized internal representation (e.g., `Document` objects)

Data Loading

Goal: ingest raw content from multiple sources and convert it into a clean, consistent **document collection** for downstream chunking and embedding.

- **Typical inputs:**
  - PDFs (reports, regulations, manuals)
  - Web pages (HTML, intranet portals)
  - Office files (DOCX, PPTX)
  - Plain text (TXT, Markdown)
- **Key outputs:**
  - extracted text (and optionally tables/images)
  - metadata (source, title, date, author, URL, page number)
  - a standardized internal representation (e.g., `Document` objects)
- **Main challenges:**
  - noisy extraction (headers/footers, duplicated text, broken paragraphs)
  - encoding issues and multilingual text
  - scanned PDFs requiring OCR

Data Loading

Goal: ingest raw content from multiple sources and convert it into a clean, consistent **document collection** for downstream chunking and embedding.

- **Typical inputs:**
  - PDFs (reports, regulations, manuals)
  - Web pages (HTML, intranet portals)
  - Office files (DOCX, PPTX)
  - Plain text (TXT, Markdown)
- **Key outputs:**
  - extracted text (and optionally tables/images)
  - metadata (source, title, date, author, URL, page number)
  - a standardized internal representation (e.g., `Document` objects)
- **Main challenges:**
  - noisy extraction (headers/footers, duplicated text, broken paragraphs)
  - encoding issues and multilingual text
  - scanned PDFs requiring OCR

Why it matters: poor data loading produces poor chunks, weak embeddings, and unreliable retrieval

Chunking Step

- In real-world RAG, we rarely retrieve from short texts.

Chunking Step

- In real-world RAG, we rarely retrieve from short texts.
- Typical sources include:
  - long PDFs (regulations, manuals, contracts)
  - technical documentation
  - internal knowledge bases (KBs)

Chunking Step

- In real-world RAG, we rarely retrieve from short texts.
- Typical sources include:
  - long PDFs (regulations, manuals, contracts)
  - technical documentation
  - internal knowledge bases (KBs)
- These documents may contain **hundreds of pages** and exceed the LLM context window.

Chunking Step

- In real-world RAG, we rarely retrieve from short texts.
- Typical sources include:
  - long PDFs (regulations, manuals, contracts)
  - technical documentation
  - internal knowledge bases (KBs)
- These documents may contain **hundreds of pages** and exceed the LLM context window.
- **Chunking** transforms a long document into smaller retrievable units.

Chunking Step

- In real-world RAG, we rarely retrieve from short texts.
- Typical sources include:
  - long PDFs (regulations, manuals, contracts)
  - technical documentation
  - internal knowledge bases (KBs)
- These documents may contain **hundreds of pages** and exceed the LLM context window.
- **Chunking** transforms a long document into smaller retrievable units.
- The chunking strategy directly affects:
  - retrieval precision (noise vs. relevance)
  - completeness (missing key context)
  - faithfulness (avoiding hallucinations)

Chunking Step: Strategies

- **Fixed-size chunks**
  Split text into uniform blocks (e.g., 500 tokens), usually with overlap (e.g., 50–100 tokens), regardless of structure.
- **Recursive chunking**
  Split hierarchically using fallbacks: sections $\rightarrow$ paragraphs $\rightarrow$ sentences $\rightarrow$ smaller units if needed.
- **Semantic chunking**
  Use embeddings/similarity to detect topic shifts and define chunk boundaries, preserving meaning across segments.
- **Structure-based chunking**
  Exploit document structure (e.g., sections, headings, tables, code blocks).

Embedding Step

```python
from langchain.embeddings import OpenAIEmbeddings

# Example texts (chunks from a document)
texts = [
    "The cat sits outside.",
    "It is sunny today.",
    "The dog barks loudly."
]

# Create embedding model
embedding_model = OpenAIEmbeddings()

# Generate vector representations
vectors = embedding_model.embed_documents(texts)

print(len(vectors), "embeddings generated.")
print("Dimension of each embedding:", len(vectors[0]))
```

Each text chunk is mapped to a high-dimensional vector capturing semantic meaning.

Chunking Step: Practical Guidelines

- Chunk size is a trade-off:
  - too small $\rightarrow$ loss of context, fragmented retrieval
  - too large $\rightarrow$ noisy retrieval, wasted context window

Chunking Step: Practical Guidelines

- Chunk size is a trade-off:
    - too small $\rightarrow$ loss of context, fragmented retrieval
    - too large $\rightarrow$ noisy retrieval, wasted context window
- Use overlap (almost always):
  Typical values: **10–20%** of the chunk size (e.g., 500 + 75 tokens).

Chunking Step: Practical Guidelines

- **Chunk size is a trade-off:**
  - too small $\rightarrow$ loss of context, fragmented retrieval
  - too large $\rightarrow$ noisy retrieval, wasted context window
- **Use overlap (almost always):**
  Typical values: **10–20%** of the chunk size (e.g., 500 + 75 tokens).
- **Prefer structure when available:**
  Splitting by sections/headings usually improves retrieval quality.

Chunking Step: Practical Guidelines

- **Chunk size is a trade-off:**
  - too small $\rightarrow$ loss of context, fragmented retrieval
  - too large $\rightarrow$ noisy retrieval, wasted context window
- **Use overlap (almost always):**
  Typical values: **10–20%** of the chunk size (e.g., 500 + 75 tokens).
- **Prefer structure when available:**
  Splitting by sections/headings usually improves retrieval quality.
- **Keep tables and code blocks intact:**
  Avoid breaking them across chunks whenever possible.

Chunking Step: Practical Guidelines

- **Chunk size is a trade-off:**
  - too small $\rightarrow$ loss of context, fragmented retrieval
  - too large $\rightarrow$ noisy retrieval, wasted context window
- **Use overlap (almost always):**
  Typical values: **10–20%** of the chunk size (e.g., 500 + 75 tokens).
- **Prefer structure when available:**
  Splitting by sections/headings usually improves retrieval quality.
- **Keep tables and code blocks intact:**
  Avoid breaking them across chunks whenever possible.
- **Tune empirically:**
  The "best" strategy depends on the document type and user queries.

Vector Storage Step

- After generating embeddings, store them in a **vector database**.
- Each entry typically contains:
  - The **embedding vector** (high-dimensional representation).
  - The **original text chunk**.
  - Optional **metadata** (source, page number, section, etc.).
- Vector DBs (e.g., `Chroma`, `FAISS`, `Weaviate`, `Pinecone`) enable:
  - Fast similarity search (cosine, dot product).
  - Efficient retrieval of relevant chunks for grounding.

Vector Storage Step

- After generating embeddings, store them in a **vector database**.
- Each entry typically contains:
  - The **embedding vector** (high-dimensional representation).
  - The **original text chunk**.
  - Optional **metadata** (source, page number, section, etc.).
- Vector DBs (e.g., `Chroma`, `FAISS`, `Weaviate`, `Pinecone`) enable:
  - Fast similarity search (cosine, dot product).
  - Efficient retrieval of relevant chunks for grounding.

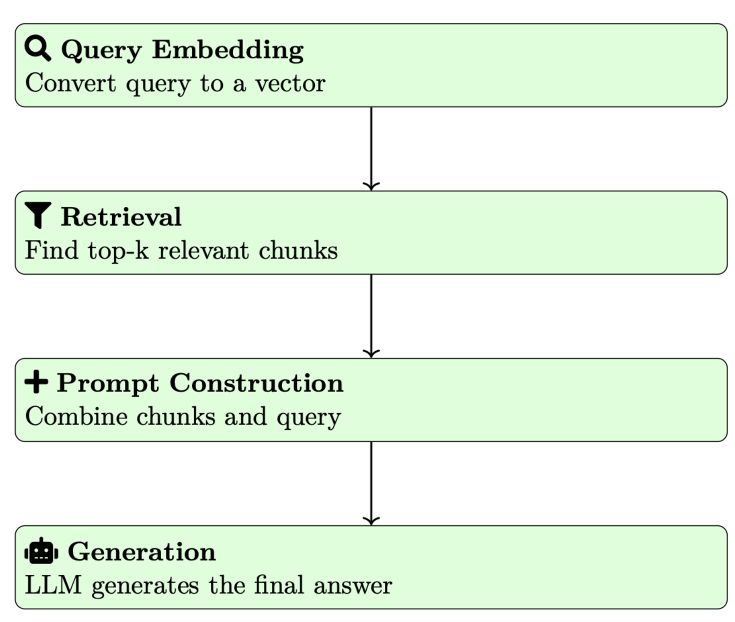
Key idea: Store once (offline), query many times (online).

Vector Storage Step

```python
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter

# Example document
text = "The cat sits outside. It is sunny today. The dog barks loudly."
chunks = CharacterTextSplitter(chunk_size=40,
↪  chunk_overlap=0).split_text(text)

# Embedding model
embedding_model = OpenAIEmbeddings()

# Store chunks + embeddings in Chroma vector DB
vectorstore = Chroma.from_texts(chunks, embedding_model)

# Example query
query = "What is the weather like?"
docs = vectorstore.similarity_search(query, k=2)

for d in docs:
    print(d.page_content)
```

Chunks are stored once with their embeddings, enabling efficient retrieval at query time.

Phase 2: Retrieval & Generation (Online)



Query Embedding Step

Query $\rightarrow$ Embedding $\rightarrow$ Vector DB similarity search.

- When a user submits a **query**, it is also converted into an embedding vector.
- This embedding captures the **semantic meaning** of the query.
- The query vector is then compared (via similarity search) to the stored document vectors.
- The most similar chunks are retrieved and provided to the LLM as context.
- **Key idea:** Questions and documents live in the **same vector space**.

Retrieval Step

Query vector $\rightarrow$ similarity search $\rightarrow$ top-k chunks.

- The query embedding is compared with all stored document embeddings.
- A similarity function (e.g., cosine similarity) measures closeness in the vector space.
- The system retrieves the **top-k most relevant chunks**.
- Retrieved chunks are injected into the LLM prompt as additional context.
- **Key idea:** Retrieval bridges the user query with the most useful knowledge.

Prompt Construction Step

Instruction + Context + Question $\rightarrow$ LLM input

- Retrieved chunks are concatenated with the user query.
- The combined text forms the **augmented prompt** sent to the LLM.
- Ensures that generation is grounded in **relevant external knowledge**.
- Prompt typically includes:
  - **Instruction:** what the model should do.
  - **Context:** retrieved chunks from the vector DB.
  - **Question:** the user's original query.
- **Key idea:** Retrieval + Query $\rightarrow$ Prompt for grounded generation.

Generation Step

LLM + augmented prompt ⇒ grounded response.

- The augmented prompt (instruction + retrieved chunks + user query) is sent to the LLM.
- The model generates a **grounded answer**, combining fluency with retrieved evidence.
- Output may include:
  - **Direct answer** to the user's query.
  - **Citations or references** from the retrieved chunks.
  - **Structured formats** (tables, JSON, summaries), depending on the task.
- **Key idea:** The LLM no longer relies only on pretraining — it reasons over the retrieved knowledge.

Example Code (LangChain)

```python
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
from langchain.chains import RetrievalQA
from utils import get_llm  # helper for model selection

# Build index
vectorstore = Chroma.from_documents(docs, embedding=OpenAIEmbeddings())

# Create retriever
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})

# RAG pipeline
qa = RetrievalQA.from_chain_type(llm=get_llm(), retriever=retriever)
qa.invoke({"query": "Summarize the main differences between RAG and
↪  fine-tuning"})
```

Demo Time

(Extra) Chunking Strategies: Examples

Base Document (Single Source Text)

In the next slides, we will apply different chunking strategies to the **same document**.

A Norma 175 da CVM é um marco regulatório para os fundos de investimento.
Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço.
Além disso, a norma foca na transparência para o investidor final.

Base Document (Single Source Text)

In the next slides, we will apply different chunking strategies to the **same document**.

A Norma 175 da CVM é um marco regulatório para os fundos de investimento.
Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço.
Além disso, a norma foca na transparência para o investidor final.

Goal: compare how each chunking strategy splits this text, and why that matters for retrieval.

Chunking Strategy: Fixed-size chunks

Strategy: split the text into uniform blocks (e.g., `max 12 tokens`), ignoring structure.

Chunking Strategy: Fixed-size chunks

Strategy: split the text into uniform blocks (e.g., `max 12 tokens`), ignoring structure.

Example chunks (illustrative):

- Chunk 1: *A Norma 175 da CVM é um marco regulatório*
- Chunk 2: *para os fundos de investimento. Ela estabelece regras claras*
- Chunk 3: *sobre a estrutura e deveres dos prestadores de serviço. Além disso,*
- Chunk 4: *a norma foca na transparência para o investidor final.*

Chunking Strategy: Fixed-size chunks

Strategy: split the text into uniform blocks (e.g., `max 12 tokens`), ignoring structure.

Example chunks (illustrative):

- Chunk 1: *A Norma 175 da CVM é um marco regulatório*
- Chunk 2: *para os fundos de investimento. Ela estabelece regras claras*
- Chunk 3: *sobre a estrutura e deveres dos prestadores de serviço. Além disso,*
- Chunk 4: *a norma foca na transparência para o investidor final.*

Typical effect in RAG:

- It may break sentences and separate key context across chunks.
- Retrieval may return an "incomplete" chunk (e.g., Chunk 2 alone).

Chunking Strategy: Recursive chunking

Strategy: split using fallbacks, from coarse to fine.

Example: paragraphs $\rightarrow$ sentences $\rightarrow$ tokens.

Chunking Strategy: Recursive chunking

Strategy: split using fallbacks, from coarse to fine.
Example: paragraphs $\rightarrow$ sentences $\rightarrow$ tokens.

Example chunks (with a small chunk limit):

- Chunk 1 (sentence): *A Norma 175 da CVM é um marco regulatório para os fundos de investimento.*
- Chunk 2 (sentence): *Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço.*
- Chunk 3 (sentence): *Além disso, a norma foca na transparência para o investidor final.*

Chunking Strategy: Recursive chunking

Strategy: split using fallbacks, from coarse to fine.
Example: paragraphs $\rightarrow$ sentences $\rightarrow$ tokens.

Example chunks (with a small chunk limit):

- Chunk 1 (sentence): *A Norma 175 da CVM é um marco regulatório para os fundos de investimento.*
- Chunk 2 (sentence): *Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço.*
- Chunk 3 (sentence): *Além disso, a norma foca na transparência para o investidor final.*

Typical effect in RAG:

- Preserves natural linguistic units (complete sentences).
- Avoids the worst boundary errors of fixed-size chunking.

Chunking Strategy: Semantic chunking

Strategy: build chunks that preserve meaning by grouping sentences with the same topic.

Chunking Strategy: Semantic chunking

Strategy: build chunks that preserve meaning by grouping sentences with the same topic.

Example chunks (illustrative):

- Chunk 1 (context): *A Norma 175 da CVM é um marco regulatório para os fundos de investimento.*
- Chunk 2 (rules + purpose): *Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço. Além disso, a norma foca na transparência para o investidor final.*

Chunking Strategy: Semantic chunking

Strategy: build chunks that preserve meaning by grouping sentences with the same topic.

Example chunks (illustrative):

- Chunk 1 (context): *A Norma 175 da CVM é um marco regulatório para os fundos de investimento.*
- Chunk 2 (rules + purpose): *Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço. Além disso, a norma foca na transparência para o investidor final.*

Typical effect in RAG:

- Tries to keep chunks semantically self-contained.
- Often groups sentences that complement each other.

Chunking Strategy: Structure-based chunking

Strategy: exploit document structure (titles, sections, lists, paragraphs).

Chunking Strategy: Structure-based chunking

Strategy: exploit document structure (titles, sections, lists, paragraphs).

Example chunks (paragraph-based):

- Chunk 1 (paragraph 1): *A Norma 175 da CVM é um marco regulatório para os fundos de investimento.*
- Chunk 2 (paragraph 2): *Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço.*
- Chunk 3 (paragraph 3): *Além disso, a norma foca na transparência para o investidor final.*

Chunking Strategy: Structure-based chunking

Strategy: exploit document structure (titles, sections, lists, paragraphs).

Example chunks (paragraph-based):

- Chunk 1 (paragraph 1): *A Norma 175 da CVM é um marco regulatório para os fundos de investimento.*
- Chunk 2 (paragraph 2): *Ela estabelece regras claras sobre a estrutura e deveres dos prestadores de serviço.*
- Chunk 3 (paragraph 3): *Além disso, a norma foca na transparência para o investidor final.*

Typical effect in RAG:

- Very effective for PDFs with headings, articles, tables, and numbered items.
- In short texts, it looks trivial — but it scales extremely well.