Large Language Models (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Spring Semester 2026

October 17, 2026

Overview

1. Large Language Models (LLMs)
2. Computational Foundations
   - Tokenization, Text Embedding, Text-Position Embedding
   - Vector Store, Vector Database, Similarity Search
   - Probability Estimation and Sampling
3. Working with Large Language Models (LLMs)
   - Prompt Engineering, Context Engineering
   - Retrieval Augmented Generation (RAG)
   - LLM-as-a-Judge
4. Working with LangChain and LangGraph
5. Applications
6. Appendix A: Cheatsheets
7. Appendix B: Anthropic Claude

Large Language Models (LLMs)

Definition of Large Language Model (LLM)

Large Language Model (LLM)

A LLM is an advanced computational model designed to understand, generate human language, and perform a wide range of tasks.

LLM Process

LLMs use deep learning techniques to learn statistical relationships between words and phrases, enabling them to perform tasks like language translation, text summarization, and question answering.

Popular LLMs

Popular LLMs:

- GPT-4 and GPT-4o (OpenAI): Advanced multimodal reasoning and dialogue capabilities.
- Gemini 1.5 (Google DeepMind): Long-context reasoning, capable of handling 1M+ tokens.
- Claude 3 (Anthropic): Safety-focused, strong at reasoning and summarization.
- LLaMA 3 (Meta): Open-weight model, popular in research.
- Mistral 7B / Mixtral (Mistral AI): Efficient open-source alternatives for developers.
- BERT (Google/Facebook): Strong embedding for NLP.

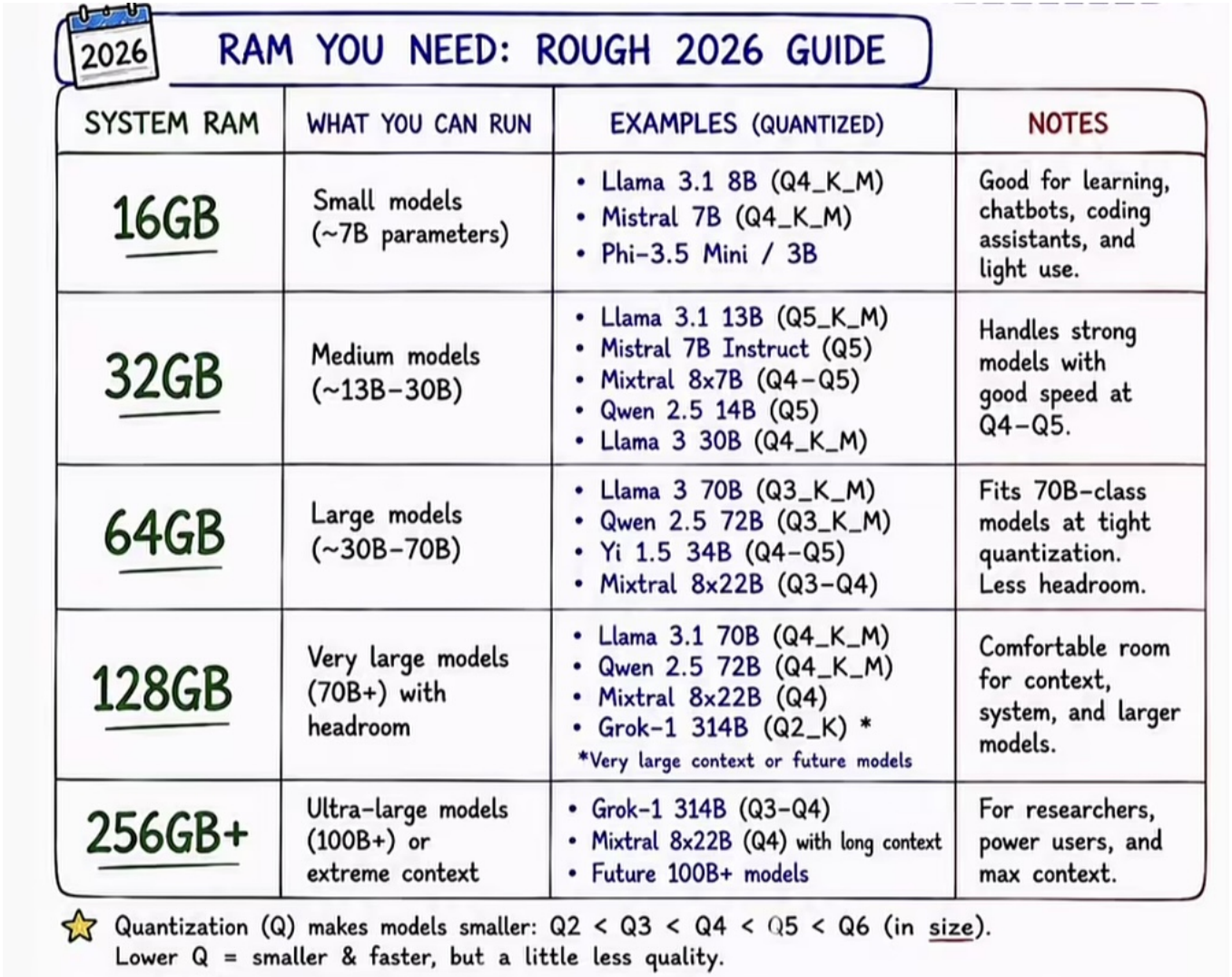
Source: Geeksforgeeks.org

LLM Type and Purpose

Type	Example	Purpose
Autoregressive	GPT, LLaMA	Text generation (next-token prediction)
Masked Language	BERT	Understanding context (classification, NER)
Multimodal	Gemini, Flamingo	Text + Image/Audio/Video processing
Instruction-Tuned	Open Chat, Alpaca	Better alignment with user commands
MoE (Sparse)	Mixtral	Activates select expert layers per input
Agentic Models	AutoGPT, CrewAI	Combine tools, reasoning, and planning
Small Language Models (SLMs)	Phi, TinyLLaMA	Efficient inference on limited compute

Source: LLM cheatsheet.

LLM Hardware Requirements



2026

RAM YOU NEED: ROUGH 2026 GUIDE

SYSTEM RAM	WHAT YOU CAN RUN	EXAMPLES (QUANTIZED)	NOTES
16GB	Small models (~7B parameters)	• Llama 3.1 8B (Q4_K_M) • Mistral 7B (Q4_K_M) • Phi-3.5 Mini / 3B	Good for learning, chatbots, coding assistants, and light use.
32GB	Medium models (~13B–30B)	• Llama 3.1 13B (Q5_K_M) • Mistral 7B Instruct (Q5) • Mixtral 8x7B (Q4–Q5) • Qwen 2.5 14B (Q5) • Llama 3 30B (Q4_K_M)	Handles strong models with good speed at Q4–Q5.
64GB	Large models (~30B–70B)	• Llama 3 70B (Q3_K_M) • Qwen 2.5 72B (Q3_K_M) • Yi 1.5 34B (Q4–Q5) • Mixtral 8x22B (Q3–Q4)	Fits 70B-class models at tight quantization. Less headroom.
128GB	Very large models (70B+) with headroom	• Llama 3.1 70B (Q4_K_M) • Qwen 2.5 72B (Q4_K_M) • Mixtral 8x22B (Q4) • Grok-1 314B (Q2_K) * *Very large context or future models	Comfortable room for context, system, and larger models.
256GB+	Ultra-large models (100B+) or extreme context	• Grok-1 314B (Q3–Q4) • Mixtral 8x22B (Q4) with long context • Future 100B+ models	For researchers, power users, and max context.

Quantization (Q) makes models smaller: Q2 < Q3 < Q4 < Q5 < Q6 (in size).
Lower Q = smaller & faster, but a little less quality.

LLM Hardware Requirements

WHAT AFFECTS RAM NEEDS?

- ✓ Model size (in parameters)
- ✓ Quantization level (Q2, Q3, Q4, Q5...)
- ✓ Context length
- ✓ KV cache (grows with context & layers)
- ✓ System + app overhead (2–6GB typical)

RAM USAGE FORMULA (SIMPLIFIED)

RAM needed ≈ Model size (GB) + KV cache (GB) + Context (GB) + Overhead (GB)

Use a hardware calculator for exact numbers.



CHECK YOUR MODEL BEFORE YOU BUY

Use one of these calculators to size the exact model, quantization, and context:

- ✓ Hugging Face LLM Inference Calculator
  https://huggingface.co/spaces/hf-accelerate/llm-inference-calculator
- ✓ LLM Calculator (by Nils Rogge)
  https://llm-calculator.streamlit.app/
- ✓ Misty's LLM RAM Calculator
  https://mistys-infrastructure-tools.streamlit.app/llm_ram_calculator

YOU'LL INPUT:
- Model
- Quantization
- Context length
- System specs

→ You'll get exact RAM estimate!



BOTTOM LINE: 16GB runs small, 32GB handles medium, 64GB fits 70B (tight), and 128GB runs 70B comfortably with room for larger models.



LLM Applications

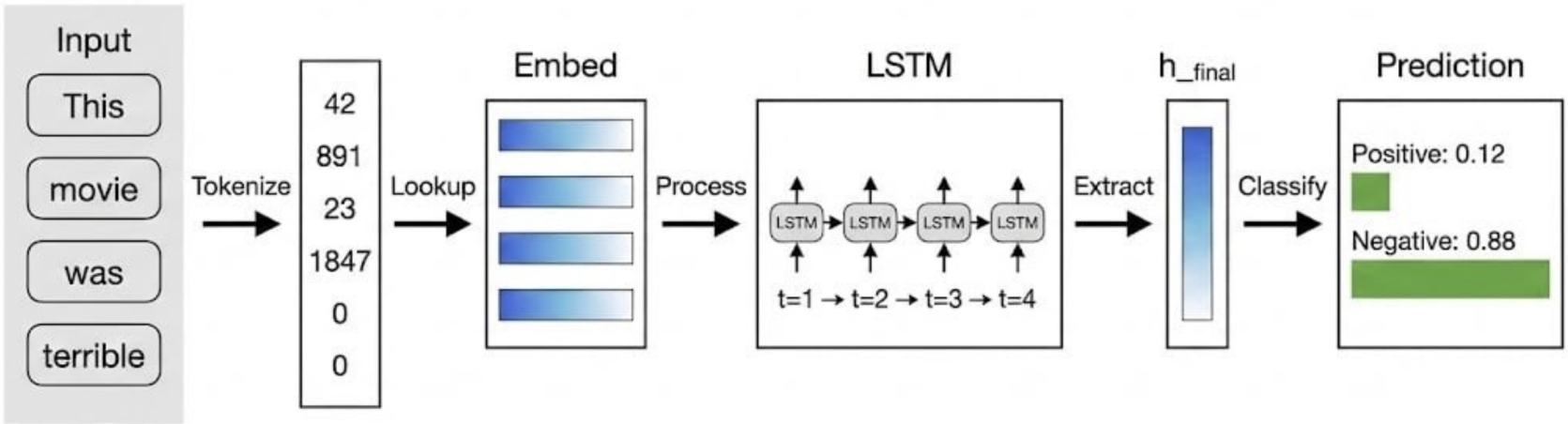
Applications:

- **Code Generation:** Generate accurate code based on user instructions for specific tasks.
- **Debugging and Documentation:** Assist in identifying code errors, suggesting fixes and even automating project documentation.
- **Question Answering:** Ask both casual and complex questions, receiving detailed, context-aware responses.
- **Language Translation and Correction:** Translate across many languages (often dozens to 100+).
- **Prompt-Based Versatility:** Craft creative prompts. LLMs excel in one-shot and zero-shot learning scenarios.

Source: Geeksforgeeks.org

LLM Applications

Text Processing Pipeline:



LLM Applications

Image-to-Text Transformation:



LLM Strengths

Strengths:

- Can perform new tasks using zero-shot and few-shot learning without retraining
- Efficiently process and understand large amounts of text data
- Adapt easily to specific domains through fine-tuning
- Automate repetitive language-based tasks, reducing human effort
- Work effectively across multiple domains like healthcare, education and business

Source: Geeksforgeeks.org

LLM Limitations

Limitations:

- No Persistent memory; no Reasoning ability; limited planning.
- Require very high computational resources, making them expensive to train.
- Training can take a long time, often weeks or months
- Depend on large amounts of high-quality and unbiased data
- Consume significant energy, contributing to environmental impact
- Can introduce bias and misinformation, raising ethical concerns

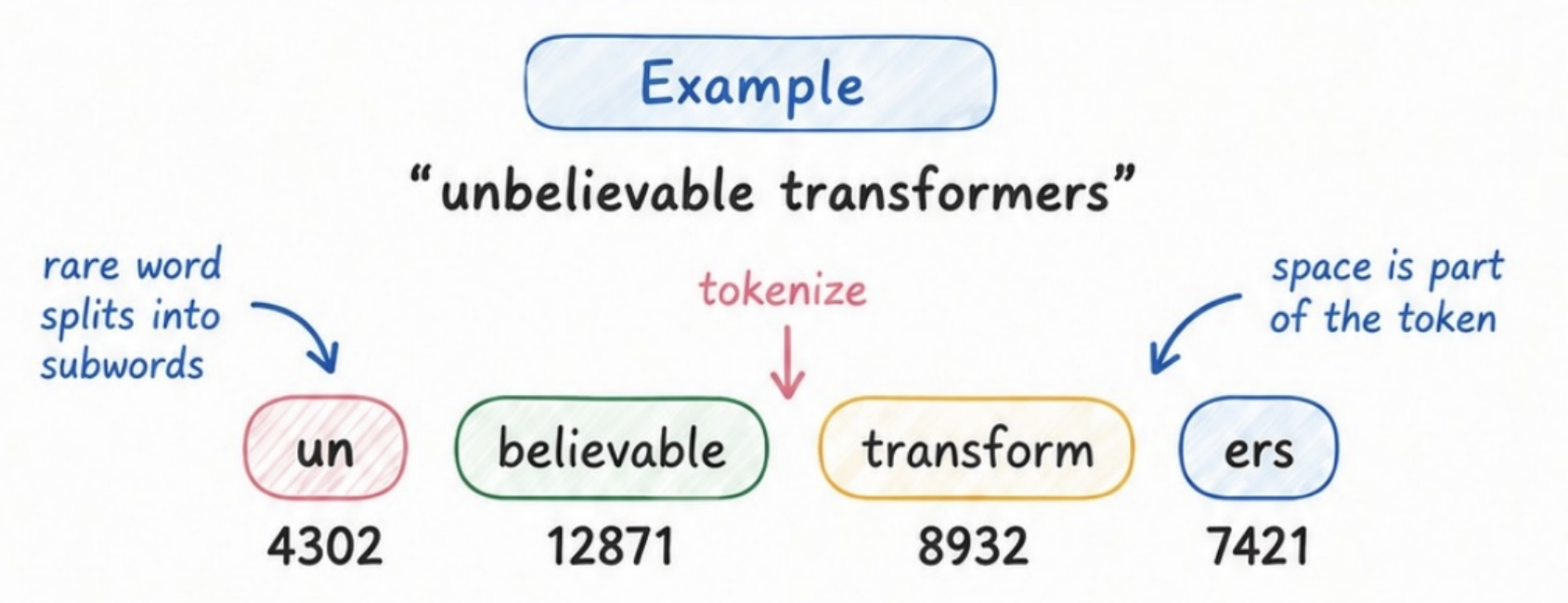
Source: Geeksforgeeks.org

Computational Foundations

Tokenization

Tokenization

Tokenization is the process of breaking raw text into smaller pieces called tokens. Tokens are then mapped onto integer IDs using a fixed vocabulary so that AI can understand them.



Tokenization

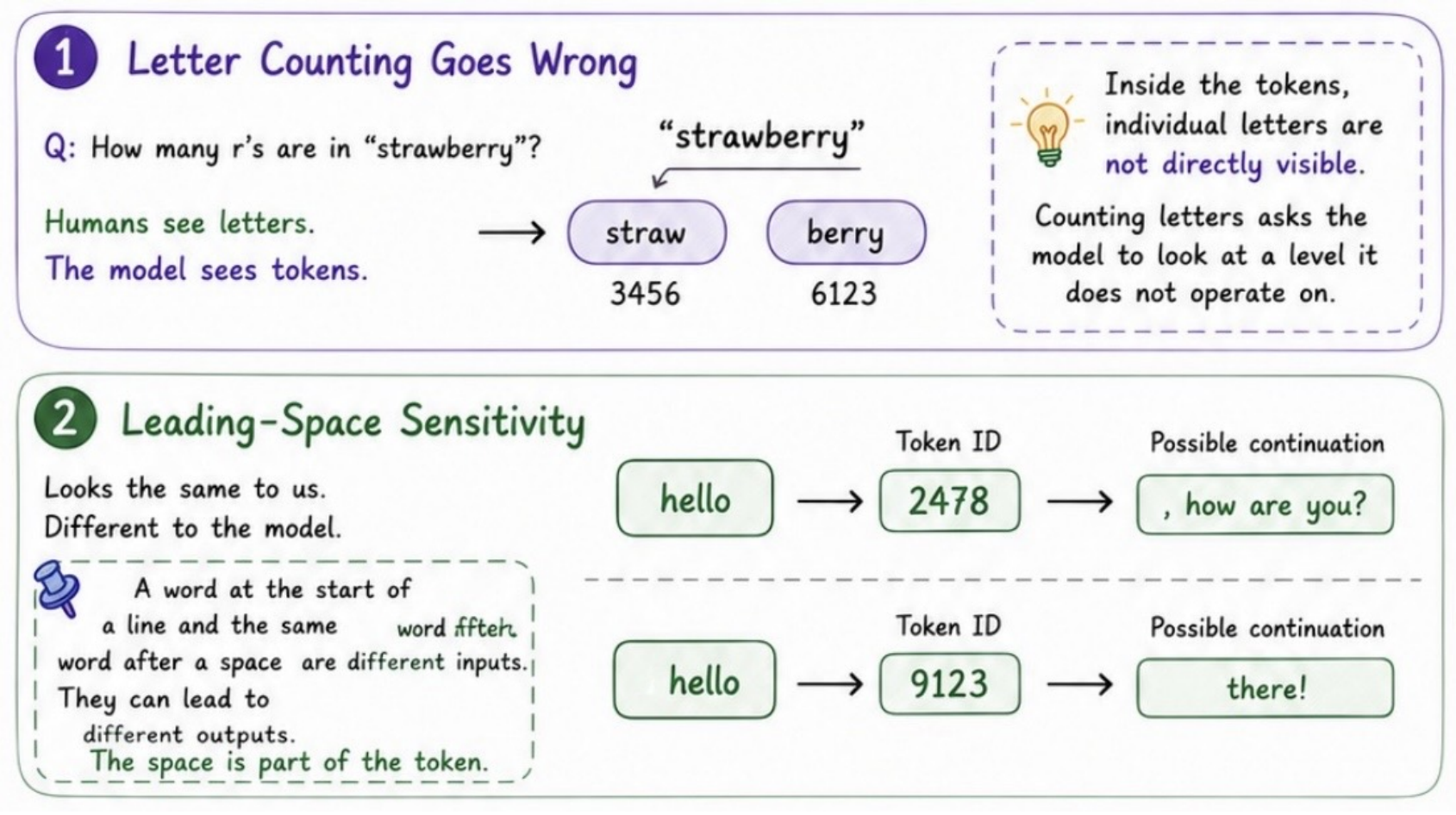
Type	Pros	Cons	Illustration
Word	• Easy to interpret • Short sequence	• Large vocabulary size • Word variations not handled	teddy bear
Subword	• Word roots leveraged • Intuitive embeddings	• Increased sequence length • Tokenization more complex	ted ##dy bear
Character Byte	• No out-of-vocabulary concerns • Small vocabulary size	• Much longer sequence length • Patterns hard to interpret because too low-level	t e d d y ␣ b e a r

Tokens vs Characters vs Words:

- **chars:** too short, no semantics.
- **words:** too many, can't handle new ones.
- **tokens:** subwords are a practical compromise.

Every model has a tokenizer; vocab size $\approx$ 30k – 200k.

Tokenization Artifacts (What can go wrong?)



Tokenization Artifacts (What can go wrong?)

3 Arithmetic Gets Tricky

Example: 380 + 17

Numbers don't have stable token boundaries.
Splits depend on context.

Can tokenize as:

380	+	17
↓	↓	↓
5021	13	286

Or can split as:

3	80	+	1	7
↓	↓	↓	↓	↓
14	908	13	16	21

Digits the model needs to add are not laid out consistently.

Inconsistent splits make reliable place-value arithmetic hard.

That's why models often reach for tools for exact math.

KEY TAKEAWAYS

- These are not bugs in a specific model; they are consequences of the representation.
- They appear across models and improve slowly.
- When a model fumbles spelling, counting, or digits, the tokenizer is usually the reason.
- The fix is to change the task, not scold the model.

Models read tokens, not characters.
Understand the tokenizer, and the model makes more sense.

Tokenization

Example:

Text Embedding

Text-Position Embedding

Vector Store and Vector Database

Vector Stores and Vector Database

Vector databases store, index, and retrieve vector embeddings for fast similarity search at scale.

Key Components:

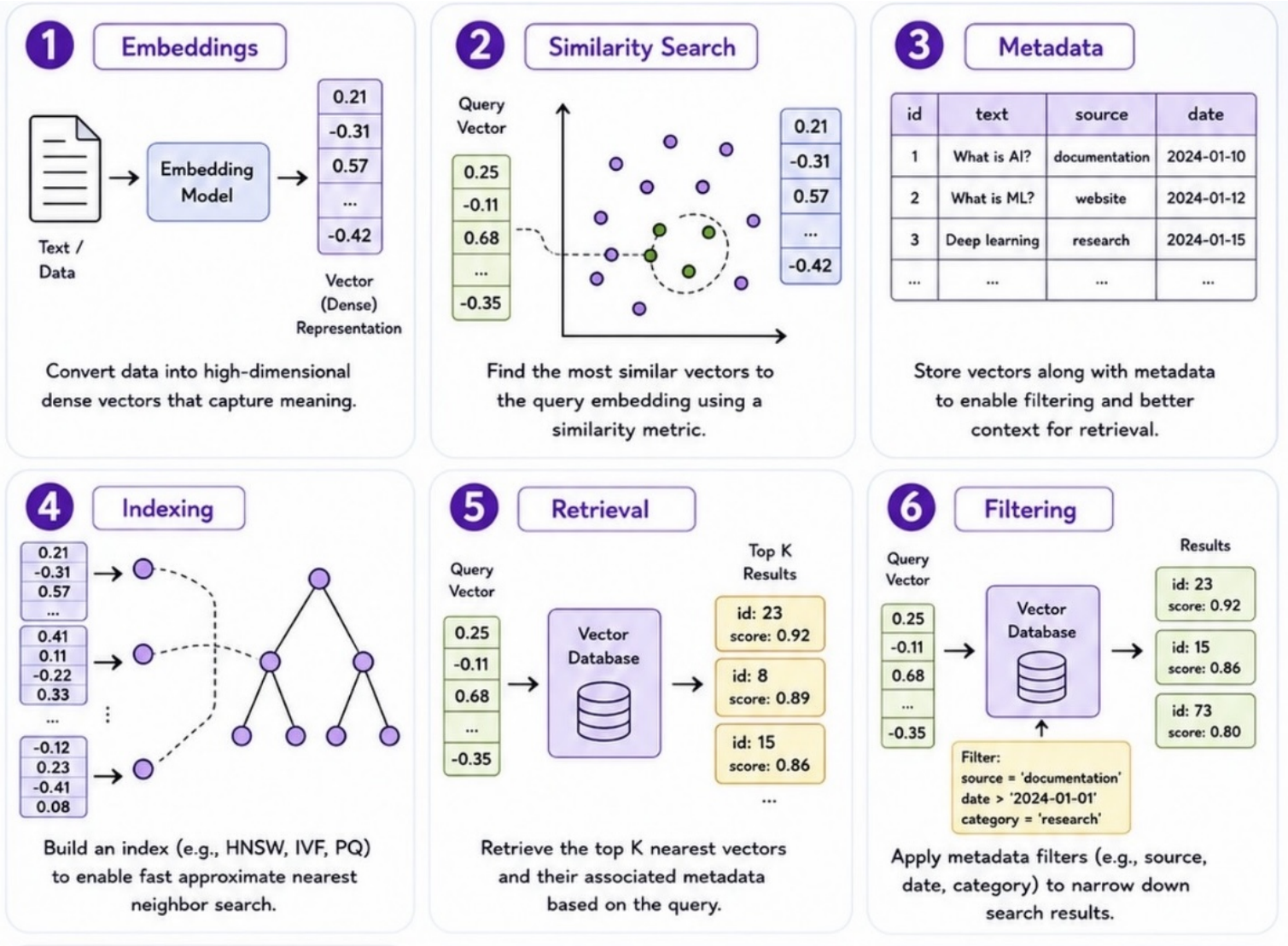
- **Embeddings:** Convert sources (text, images, audio) into high-dimensional dense vectors of numbers, designed to capture core meaning.
- **Similarity Search:** Use a similarity metric to find the most similar vectors to the query embedding.
- **Metadata:** Store vectors along with metadata to enable filtering and better context for retrieval.

Vector Store and Vector Database

Key Components:

- **Indexing:** Build an index (i.e., math shortcuts) to enable fast approximate nearest neighbor search..
- **Retrieval:** Retrieve the top K nearest vectors and their associated metadata based on specifics of a query.
- **Filtering:** Apply metadata filters (e.g., source, date, category) to narrow down search results.

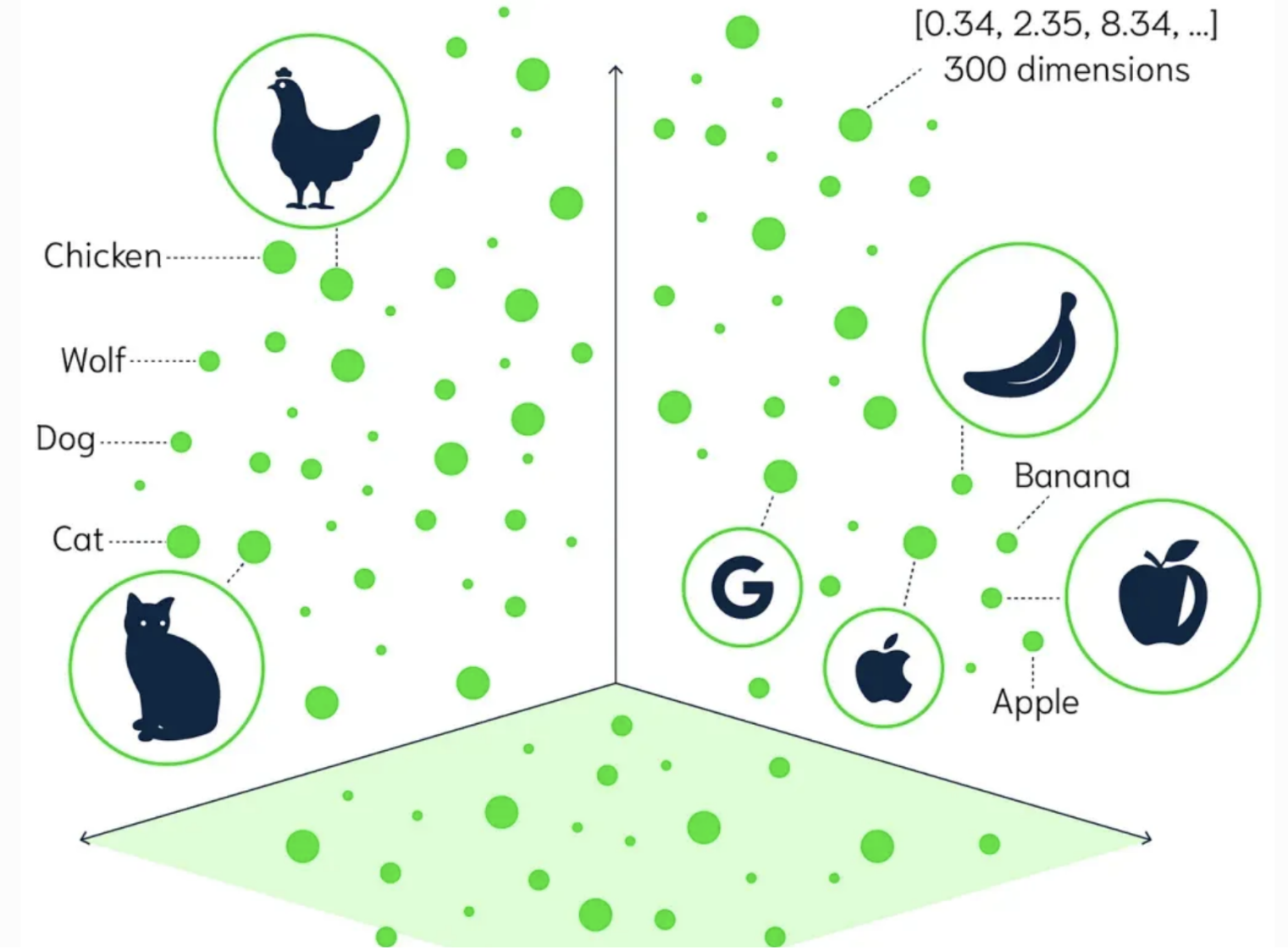
Vector Store and Vector Database



Vector Store and Vector Database

Example:

Similarity Search



Similarity Search

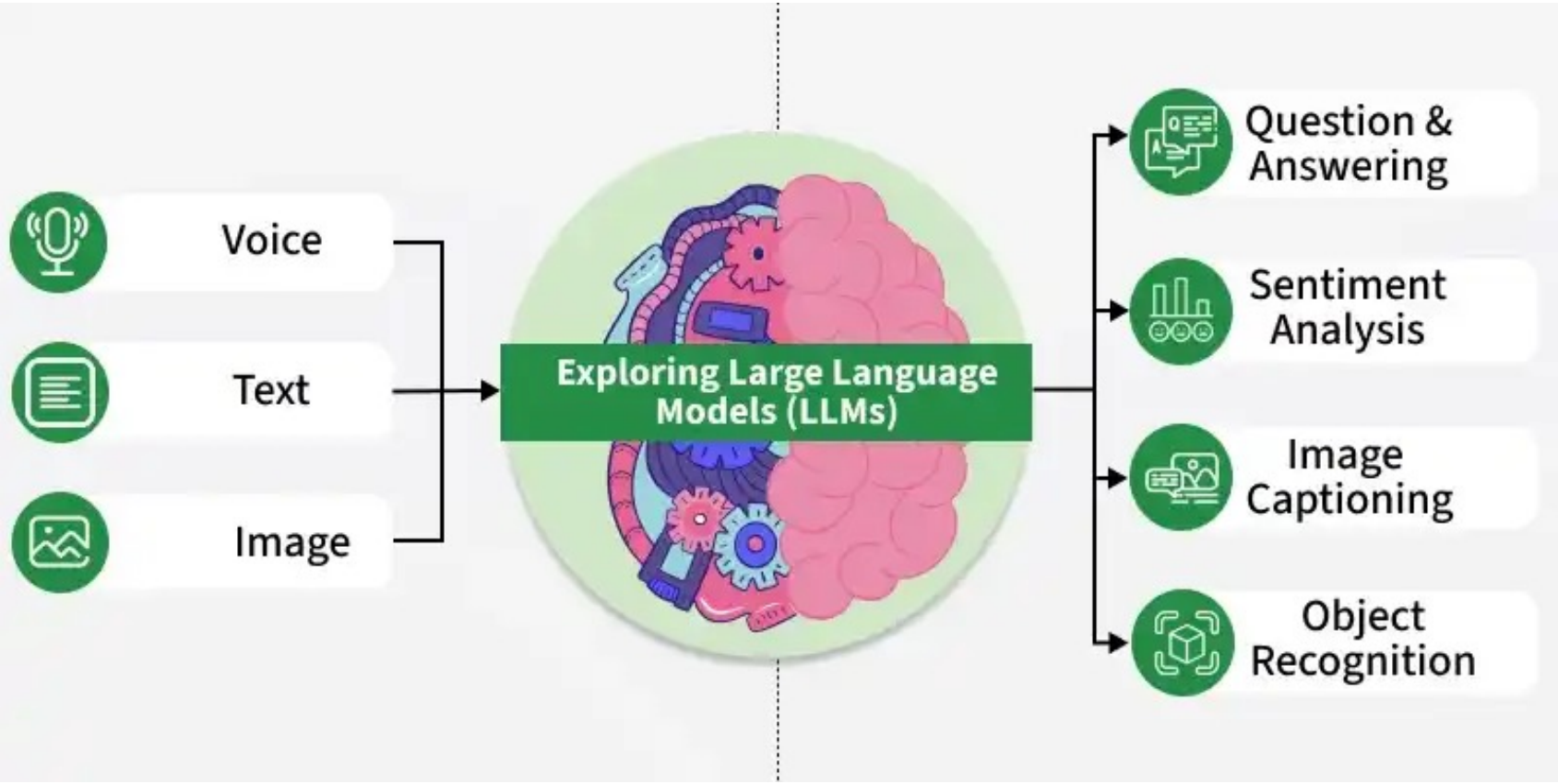
Example:

Probability Estimation and Sampling

Working with Large Language Models (LLMs)

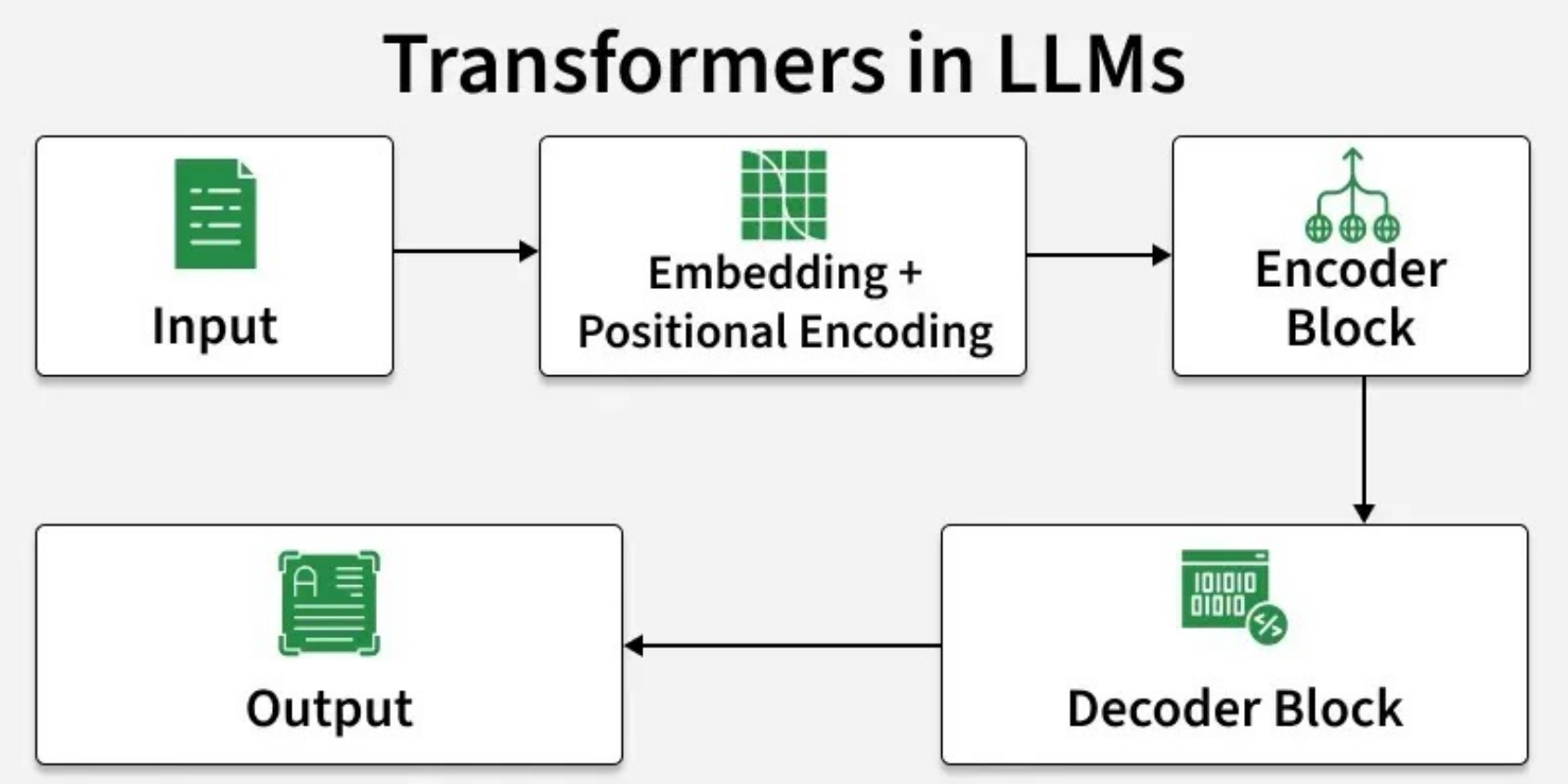
LLM Overview

LLM Overview



LLM Overview

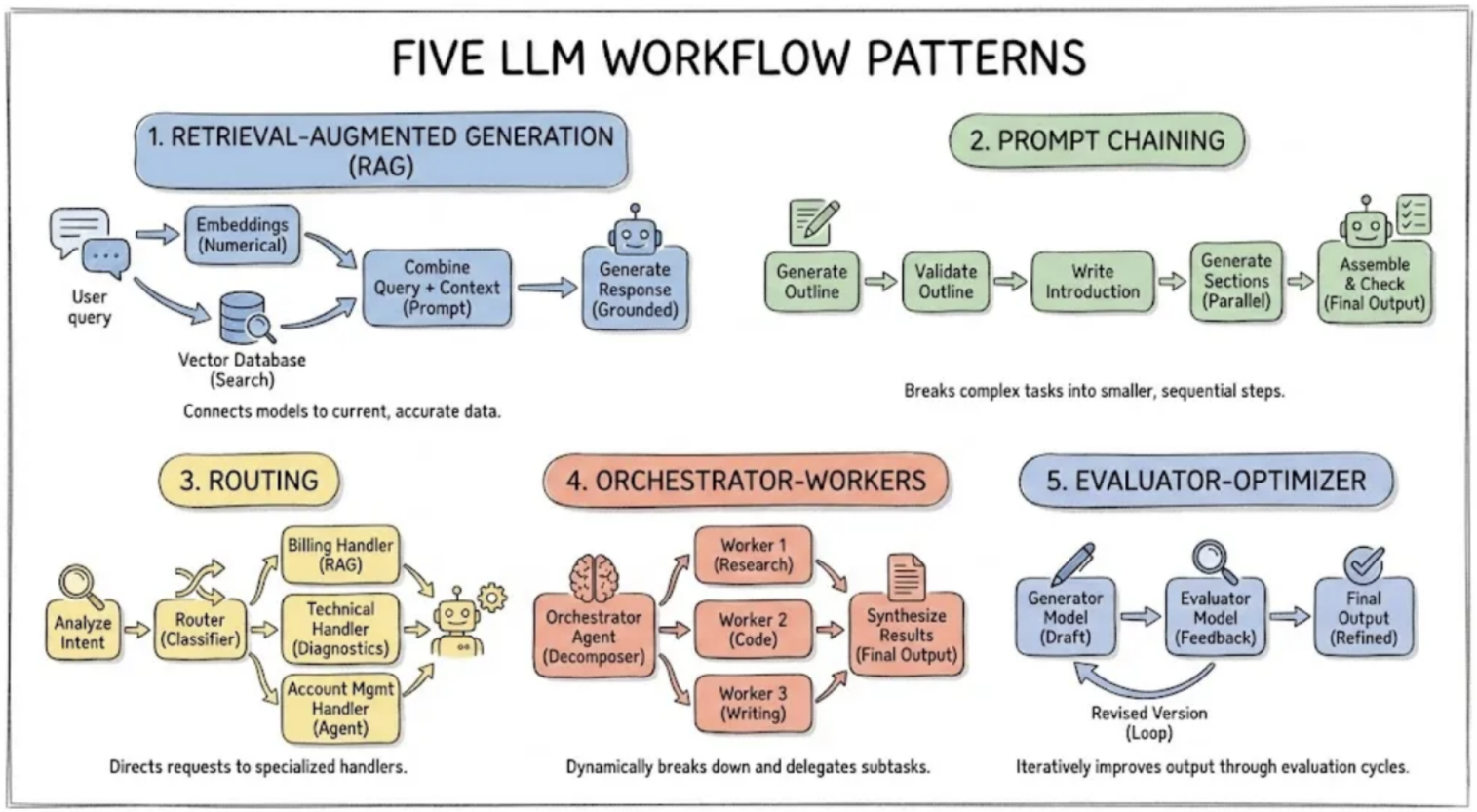
Transformers in LLMs



LLM Core Workflow Patterns

LLM Workflows

Use predetermined logic, routing and conditional checks to ensure system consistency and reliability.



LLM Core Workflow Patterns

Prompt Chaining:

- Passes the output of one LLM call directly into the next in a linear sequence (e.g., generate, critique, refine).

Routing:

- Examines unstructured user input and directs it to the most appropriate predefined handler or specialized model.

Parallelization:

- Splits a task to process multiple sub-components.

Orchestrator-Worker:

- Employs a central coordinator LLM to break a complex goal into subtasks, delegates them to specialized workers, and synthesizes the final result.

Prompt Engineering

Prompt Engineering

Prompt Engineering

Prompt Engineering is the art and science of designing, refining and optimizing inputs (prompts) to guide generative AI models toward producing accurate, relevant and high-quality outputs.

Importance:

- Prompt engineering influences the quality, relevance and accuracy of Generative AI outputs.
- A well-crafted prompt ensures that AI comprehends the user's intents, a prerequisite for producing meaningful responses.

Prompt Engineering

How it Works:

- Prompt engineering acts as a bridge between human intent and and AI understanding, transforming vague requests into structured commands.

Common Use Cases:

- Automated code generation; generate text summary; data processing and extraction; logical reasoning.

Examples:

- Write a python program to compute the factorial of a number.
- Translate the following python code to Java.
- Debug the Java code and explain why it throws an exception.

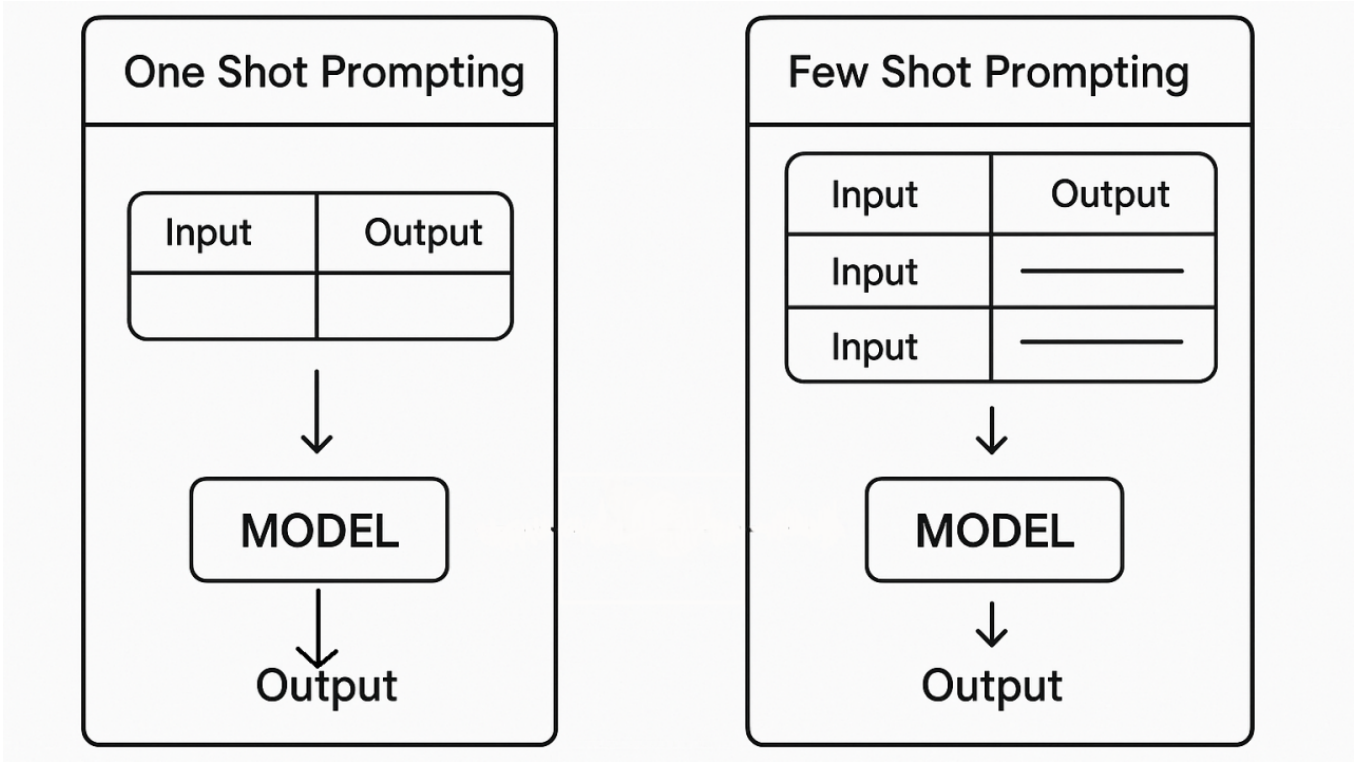
Prompt Engineering

Prompting Techniques:

- **Zero-shot prompting:** Provides the model with a direct instruction or question without any additional context or examples.
- **Few-shot prompting:** Provide the model with one or more examples of the desired input-output pairs before presenting the actual prompt.
- **Chain-of-thought prompting:** Encourages the model to break down complex reasoning into a series of intermediate steps, leading to a more comprehensive and detailed output.

Prompt Engineering

Prompt Overview:



Prompt Engineering

Standard and Chain-of-Thought Prompting:

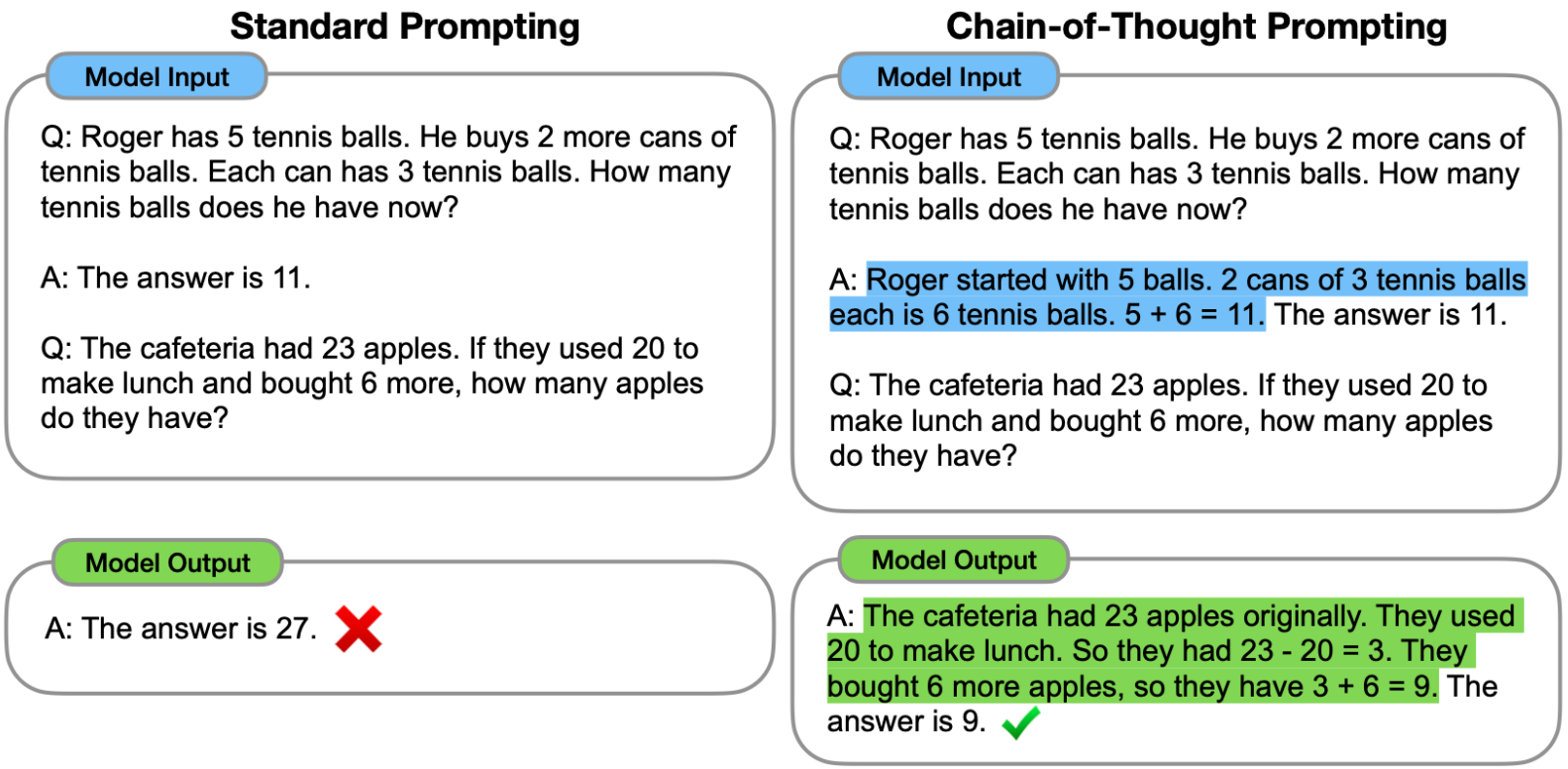


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Prompting

Math Word Problems (free response)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Math Word Problems (multiple choice)

Q: How many keystrokes are needed to type the numbers from 1 to 500?
Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788

A: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. 9 + 90(2) + 401(3) = 1392. The answer is (b).

CSQA (commonsense)

Q: Sammy wanted to go to where the people were. Where might he go?
Options: (a) race track (b) populated areas (c) desert (d) apartment (e) roadblock

A: The answer must be a place with a lot of people. Race tracks, desert, apartments, and roadblocks don't have a lot of people, but populated areas do. So the answer is (b).

StrategyQA

Q: Yes or no: Would a pear sink in water?

A: The density of a pear is about 0.6 g/cm^3, which is less than water. Thus, a pear would float. So the answer is no.

Date Understanding

Q: The concert was scheduled to be on 06/01/1943, but was delayed by one day to today. What is the date 10 days ago in MM/DD/YYYY?

A: One day after 06/01/1943 is 06/02/1943, so today is 06/02/1943. 10 days before today is 05/23/1943. So the answer is 05/23/1943.

Sports Understanding

Q: Is the following sentence plausible? "Joao Moutinho caught the screen pass in the NFC championship."

A: Joao Moutinho is a soccer player. The NFC championship is part of American football, not soccer. So the answer is no.

SayCan (Instructing a robot)

Human: How would you bring me something that isn't a fruit?

Explanation: the user wants something to eat that isn't a fruit. An energy bar is not a fruit, so I will bring the user an energy bar.
Plan: 1. find(energy bar) 2. pick(energy bar) 3. find(user) 4. put(energy bar) 5. done().

Last Letter Concatenation

Q: Take the last letters of the words in "Lady Gaga" and concatenate them.

A: The last letter of "Lady" is "y". The last letter of "Gaga" is "a". Concatenating them is "ya". So the answer is ya.

Coin Flip (state tracking)

Q: A coin is heads up. Maybelle flips the coin. Shalonda does not flip the coin. Is the coin still heads up?

A: The coin was flipped by Maybelle. So the coin was flipped 1 time, which is an odd number. The coin started heads up, so after an odd number of flips, it will be tails up. So the answer is no.

Figure 3: Examples of ⟨input, chain of thought, output⟩ triples for arithmetic, commonsense, and symbolic reasoning benchmarks. Chains of thought are highlighted. Full prompts in Appendix G.

Prompting

Question

Tracy used a piece of wire 4 feet long to support tomato plants in the garden. The wire was cut into pieces 6 inches long. How many pieces did she obtain?

62B Model Output

She cut the wire into 6 inch pieces. This means she got 4 * 6 = 24 pieces. The answer is 24. **(Incorrect: semantic understanding error)**

540B Model Output

The wire was 4 feet long. This means it was 4 * 12 = 48 inches long. It was cut into pieces 6 inches long. This means she obtained 48 / 6 = 8 pieces. The answer is 8. **(Correct)**

Question

Tom's ship can travel at 10 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 6 mph. How long does it take him to get back?

62B Model Output

He travels from 1 to 4 PM. That means he travels for 3 hours. He travels back at 6 mph. That means he travels for 3 * 6 = 18 miles. The answer is 18. **(Incorrect: semantic understanding error)**

540B Model Output

He travels at 10 mph from 1 to 4 PM. This means he travels 3 hours. 3 hours at 10 mph means he travels 3 * 10 = 30 miles. He then travels back at 6 mph. This means he travels 6 miles per hour. He has to travel 30 miles, so it takes him 30 / 6 = 5 hours. The answer is 5. **(Correct)**

Prompting

Question

Stephen placed an online order for groceries. His final bill came to $40.00. Because this was through a delivery vendor, they tacked on a 25% fee to his final total and charged him $3.00 in delivery fees. Stephen also added a $4.00 tip. After the extra fees, what was the final price of Stephen's groceries?

62B Model Output

The final price of Stephen's groceries was $40.00. The delivery vendor added 25% to the final total, so the final total was 125% of $40.00. This means the final total was 125% * $40.00 = $50.00. The delivery vendor also charged $3.00 in delivery fees. So the final price of Stephen's groceries was $50.00 + $3.00 = $53.00. The answer is $53.00. **(Incorrect: one step missing error)**

540B Model Output

Stephen's bill was $40.00. The delivery vendor added 25% to the bill. This means the bill was increased by 25% of $40.00. This is 0.25 * $40.00 = $10.00. The delivery fee was $3.00. The tip was $4.00. So the total bill was $40.00 + $10.00 + $3.00 + 540b model $4.00 = $57.00. The answer is 57. **(Correct)**

Question

There are four schools competing at a basketball tournament. Each school has sent a girls' basketball team and a boys' basketball team and each team has 5 players each. Each school has also sent a coach for each team. In total, how many people have all of the schools sent?

62B Model Output

There are 4 schools. Each school has sent a girls' basketball team and a boys' basketball team. Each team has 5 players each. That means there are 4 * 2 * 5 = 40 players. Each team has a coach. That means there are 40 + 4 = 44 coaches. The answer is 44. **(Incorrect: one step missing error)**

540B Model Output

Each school has sent 2 teams, each with 5 players. This means each school has sent 2 * 5 = 10 players. Each school has also sent 2 coaches. This means each school has sent 10 + 2 = 12 people. There are 4 schools, so in total all of the schools have sent 4 * 12 = 48 people. The answer is 48. **(Correct)**

Strategies for Writing Better Prompts

Set clear goals and objectives:

- **Tactic:** Use action verbs to specify the desired action (e.g., write a bulleted list item that summarizes the key findings of the attached research paper).
- **Tactic:** Define the desired length and format of the output (e.g., Compose a 200-word abstract summaarizing the key findings of the research paper.)
- **Tactic:** Specify the target audience (e.g., Write a 400-word essay targeting young teens concerned with global climate warming.)

Source: Wikipedia entry for prompt engineering

Strategies for Writing Better Prompts

Provide context and background information:

- **Tactic:** Include relevant facts and data. (e.g., Given that temperatures in the Tasman sea have risen 1 degree Celcius over the past decade, describe the likelihood of long term impacts caused by global warming.)
- **Tactic:** Reference specific sources or documents. (e.g., Based on the attached document, generate a forecast for the companies profitability over the next 18 months.)
- **Tactic:** Define key concepts and terms. (e.g., Explain the key ideas in quantum computing using terms suitable for a non-technical audience.)

Strategies for Writing Better Prompts

Use Few-Shot Prompting:

- **Tactic:** Provide a few examples of input-output pairs. (e.g., input: cat, output: A small hairy animal with whiskers, input: kiwi, output: A small flightless bird found in New Zealand.)
- **Tactic:** Demonstrate the desired style or tone. (e.g., (humorous) the dignitary delivered a speech that was so dull it could cure insomnia, (formal) the dignitary delivered a speech that was informative and entertaining.)
- **Tactic:** Show the desired level of detail. (e.g., (brief) The movie was about a young boy who has lunch with an alien, (detailed) The science fiction movie follows the adventures of Casper, who meets and lunch with a friendly alien.)

Strategies for Writing Better Prompts

Be Specific:

- **Tactic:** Use precise language and avoid ambiguity. (e.g., Write a persuasive essay arguing for regulated approaches to sustainability.)
- **Tactic:** Whenever possible, quantify the requests. (e.g., Write a poem with 14 lines.)
- **Tactic:** Break down complex tasks into simpler tasks. (e.g., Instead of saying develop a meeting plan, say: 1. Identify the target audience, 2 Develop key marketing messages, 3 Select appropriate marketing channels.)

Strategies for Writing Better Prompts

Iterate and Experiment:

- **Tactic:** Try a variey of phrasings and keywords. (e.g., Rephrase the prompt using a variety of synonyms or alternate sentence structures.)
- **Tactic:** Adjust the level of detail and specificity. (e.g., Add or remove information to fine-tune the output.)
- **Tactic:** Experiment with different response lengths. (e.g., Experiment with short/long output lengths to find the desired balance.)

Strategies for Writing Better Prompts

Leverage CoT Prompting:

- **Tactic:** Encourage multi-step reasoning. (e.g., Solve the problem step-by-step.)
- **Tactic:** Ask the model to explain its reasoning strategy. (e.g., Explain the thought process in determining the sentiment review of a movie, for example, the acting was superb, but the plot too predictable.)
- **Tactic:** Guide the model through a logical sequence of thought. (e.g., To classify the e-mail as spam, consider the following: 1. Is the sender known?, 2. Does the subject line contain suspicious words?, 3. Is the e-mail offerring something too good to be true?)

Context

Engineering

Context Engineering

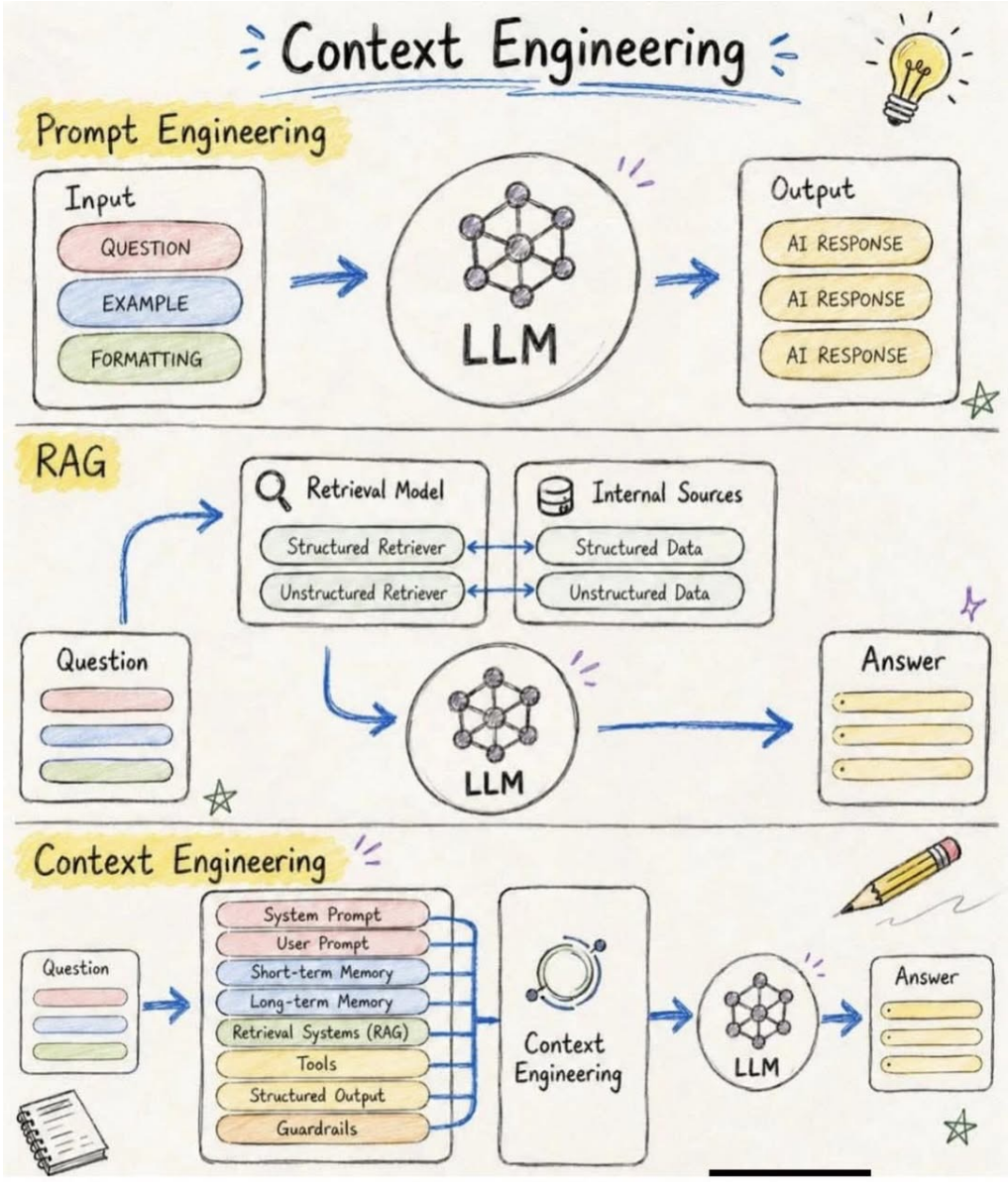
Context Engineering

Context Engineering is the practice of designing and managing dynamic systems that supply large language models (LLMs) or agents with the most relevant information, instructions and tools at right time.

Prompt vs Context vs Memory Engineering:

- Prompt engineering focusses on how a single text string or query is phrased.
- Context engineering encompasses everything the model sees during runtime before generating a response.
- Memory engineering covers the mechanisms by which an AI agent acquires, represents, stores, updates, retrieves, uses and forgets information over time.

Prompt vs RAG vs Context Engineering



Components of the Context Stack

System Instructions:

- Clear behavioral guidelines, guardrails and role definition.

User Input:

- The current prompt, question, or task assigned by the user.

Short-term State/History:

- Recent conversation turns, scratchpads, and active working memory.

Long-term Memory and RAG:

- Retrieved external documents, database lookups, and past user preferences.

Components of the Context Stack

Tools Definition:

- Available APIs, functions, and action spaces the model can execute.

Output Formatting:

- Explicit rules governing the structure of the model's response.

Strategies in Context Management

Writing:

- Allowing agents to save intermediate notes or state outside the primary context window.

Selection:

- Dynamically pull in relevant tools or facts via RAG.

Compression:

- Prune irrelevants tokens, summary history, or filtering verbose outputs.

Isolation:

- Partition context across multiple sub-agents or sandboxed environments to prevent distraction.

Context Engineering

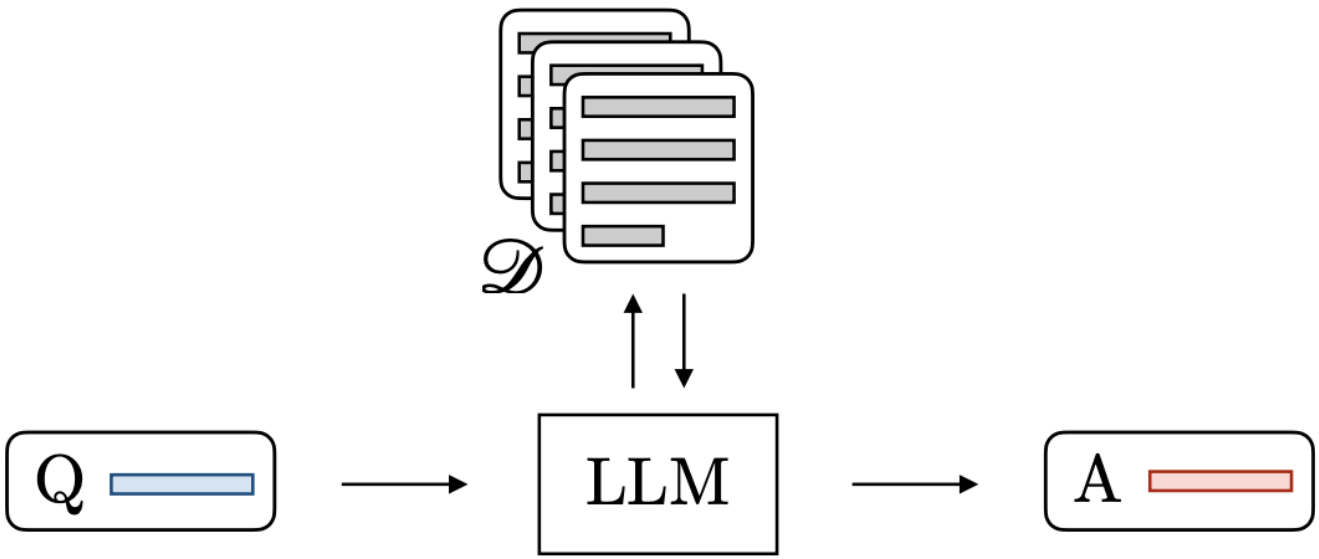
Example:

Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG)

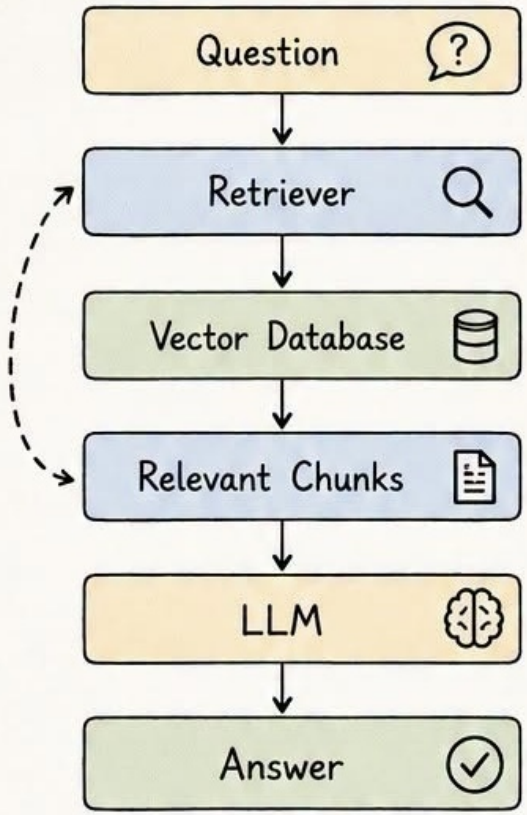
Retrieval Augmented Generation

Given a knowledge base and a question, a retriever fetches the most relevant documents, then augments the prompt with the information before generating the output.



Retrieval Augmented Generation (RAG)

RAG combines a retriever (search) with a generator (LLM) to produce more accurate and context-aware answers.



How RAG Works

1. An embedding model converts the query into a vector.
2. The retriever searches the vector database for similar chunks.
3. Most relevant chunks are retrieved from the vector database.
4. The LLM generates the final answer using the retrieved context.

Similarity Search (Cosine Similarity)

Cosine similarity measures the cosine of the angle between two vectors.

Formula:

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\| \ \|B\|}$$

Where:

- A, B = embedding vectors
- $A \cdot B$ = dot product
- $\|A\|$ = magnitude (L2 norm) of vector A
- similarity ranges from -1 (opposite direction) to 1 (most similar)

Retrieval Augmented Generation (RAG)

Embedding Example

Text:
"Artificial intelligence is transforming the world."

Embedding (example output with 5 dimensions):

[0.21, -0.34, 0.67, 0.11, -0.28]

Another text:
"AI is changing industries globally."

Embedding:

[0.19, -0.29, 0.62, 0.09, -0.31]

Cosine Similarity between the two embeddings:
similarity ≈ 0.99 (very similar)

Key Components

- **Retriever**: Retrieves relevant chunks using vector similarity search.
- **Vector Database**: Stores embeddings of document chunks.
- **Relevant Chunks**: Most relevant chunks from the vector database.
- **LLM**: Generates the answer using the retrieved context.

LangChain Code Example

```
# 1. Load documents
from langchain_community.document_loaders import TextLoader
loader = TextLoader("data.txt")
docs = loader.load()

# 2. Split documents into chunks
from langchain.text_splitter import RecursiveCharacterTextSplitter
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
chunks = splitter.split_documents(docs)

# 3. Create embeddings
from langchain_openai import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()

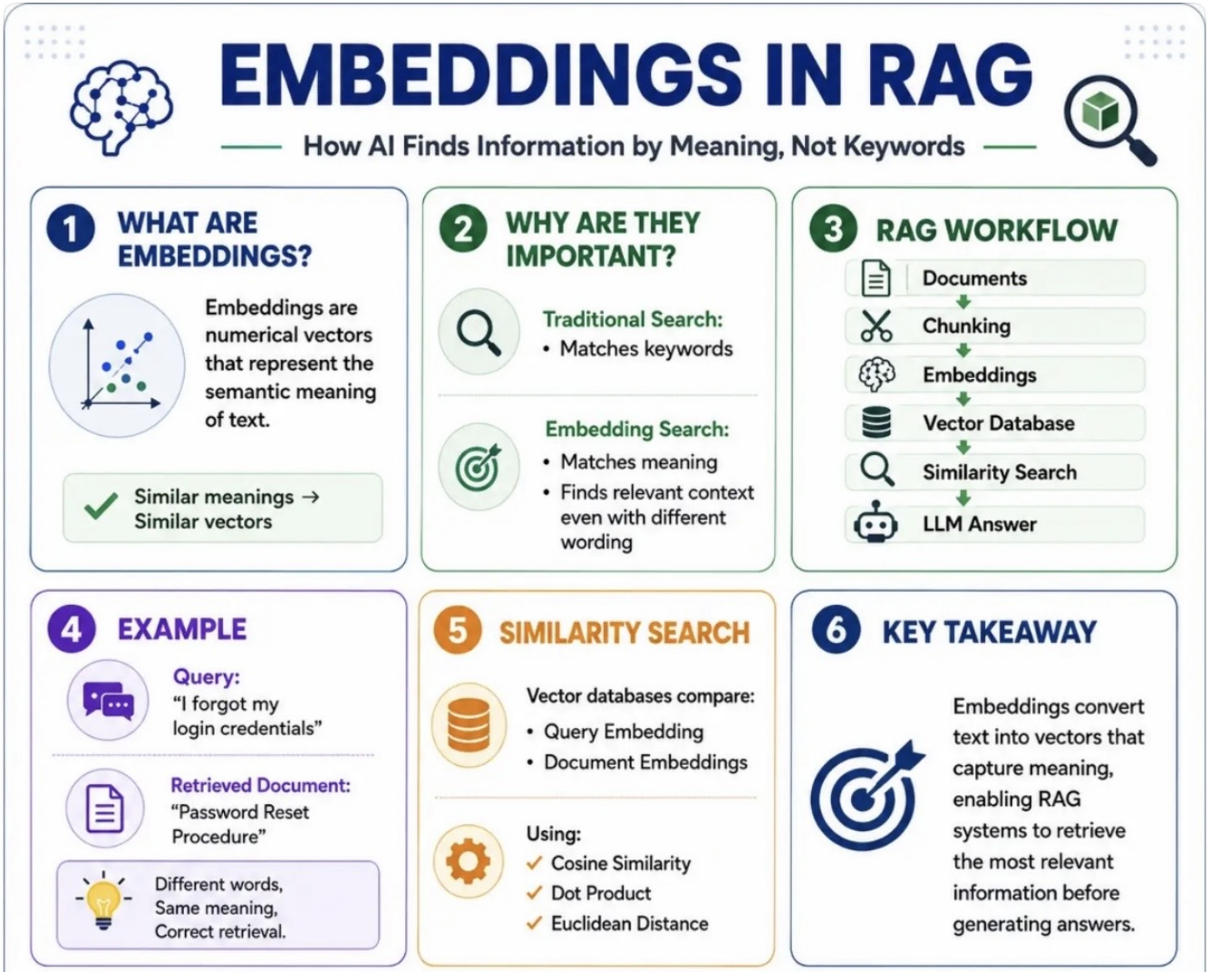
# 4. Store in vector database (FAISS)
from langchain_community.vectorstores import FAISS
vectorstore = FAISS.from_documents(chunks, embeddings)

# 5. Create retriever (top-k)
retriever = vectorstore.as_retriever(search_kwargs={"k": 4})

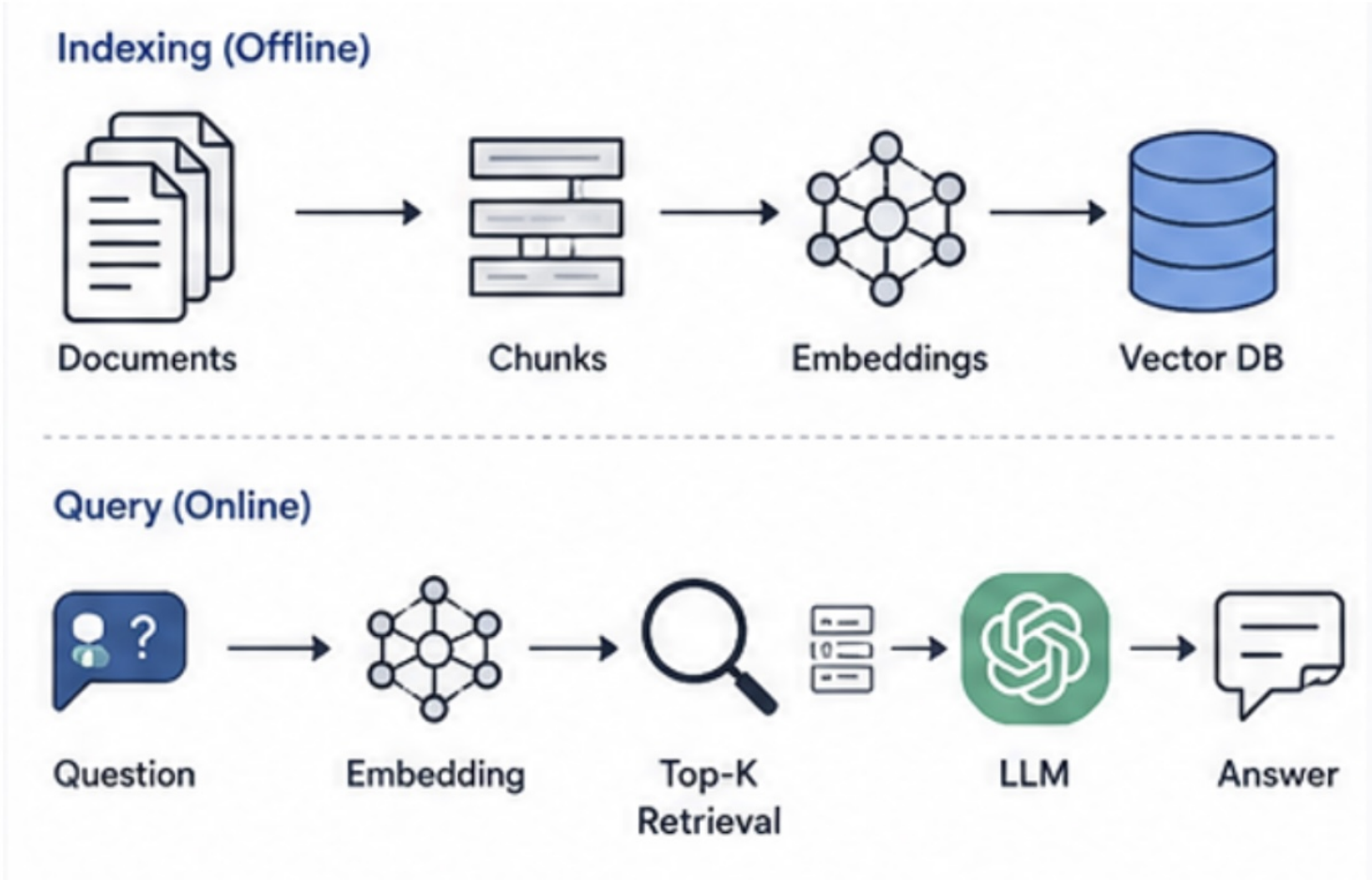
# 6. Build RAG chain
from langchain_openai import ChatOpenAI
from langchain.chains import RetrievalQA
llm = ChatOpenAI(model="gpt-4.1-mini", temperature=0)
qa = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)

# 7. Ask a question
question = "What is artificial intelligence?"
result = qa.run(question)
print(result)
```

RAG Embeddings



RAG Standard Approach



RAG Enterprise Requirements

1. Fragmented Retrieval



- Information is split across chunks
- Context gets lost

2. No Relationship Traversal



- Vector search can't follow relations
- Multi-hop reasoning is hard

3. Entity Disambiguation

NVIDIA (Company)
NVIDIA (Product)
NVIDIA (Research)

- Same words, different entities
- Similarity ≠ Identity

4. Not for Numerical Filtering



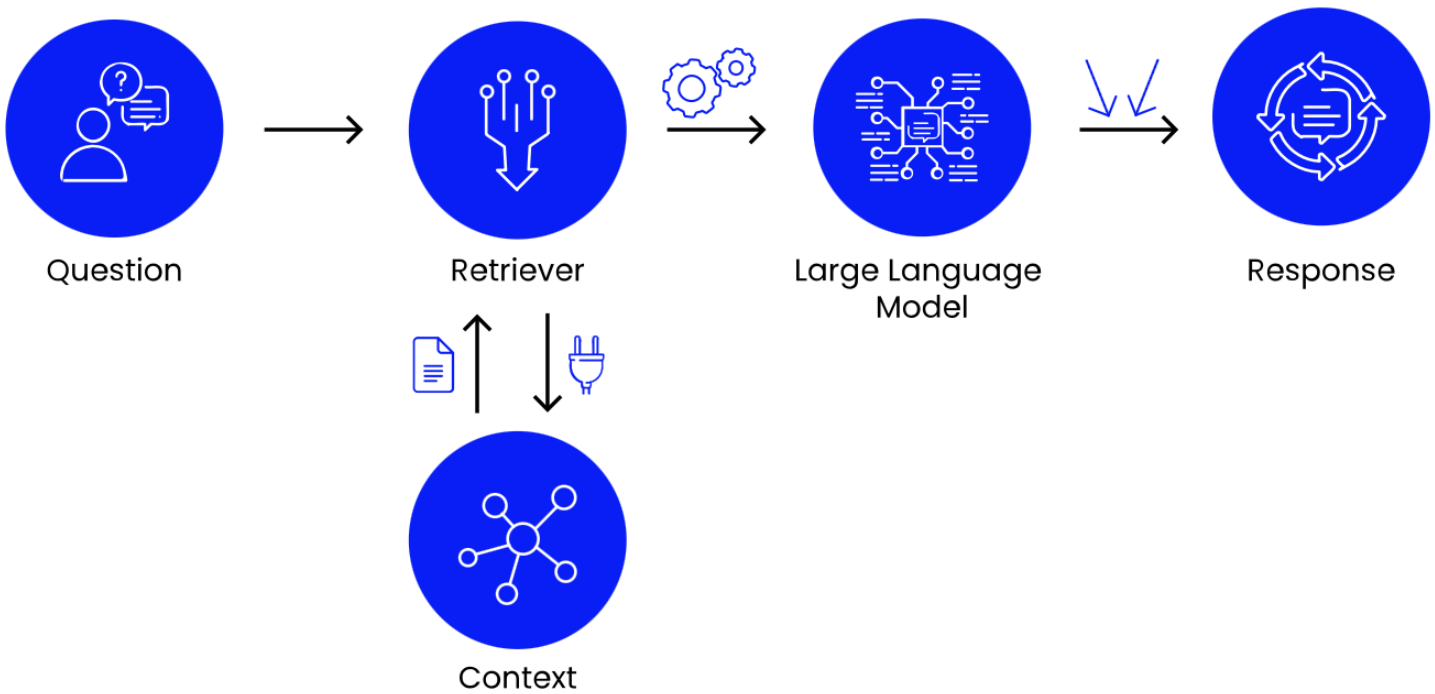
- Can't do >, <, =, SUM, ORDER BY...
- Needs structured query engines

Why pure vector RAG falls short

Graph RAG

Graph RAG

Knowledge graph retrieval method that represents entities as nodes and relationships as edges.



Graph RAG

Procedure:

- Build a graph structure (nodes and edges).
- Use graph traversal techniques.
- Feed structured multi-step knowledge to the LLM.

Benefits:

- Explicit representation of relationships between concepts.
- Multi-step reasoning ideal for complex queries.
- Works well for structured and semi-structured data.

Challenges:

- Requires structured graph data (e.g., RDF, Neo4j).
- More difficult to implement than traditional RAG.

Traditional RAG vs Graph RAG

Feature	Traditional RAG	Graph RAG
Retrieval Method	Vector Similarity Search	Graph Traversal + Embeddings
Best For	Simple Q&A , Document Retrieval	Complex reasoning , multi-hop queries
Data Type	Unstructured text (docs , FAQs, articles)	Structured + unstructured(linked entities, graphs)
Scalability	Easier to scale	More complex but powerful
Use Case Example	Finding a relevant research paper	Connecting multiple research papers to derive insights

Source: Ali S.M., "Traditional RAG vs Graph RAG", medium.com, February, 2025.

Traditional RAG vs Graph RAG

Use Traditional RAG:

- Data is text-based and loosely related (independent).
- Problem domain only requires simple similarity-based retrieval.
- You need scalable, easy-to-implement retrieval.

Use Graph RAG:

- Data involves complex relationships among entities.
- Problem domain requires multi-step reasoning.

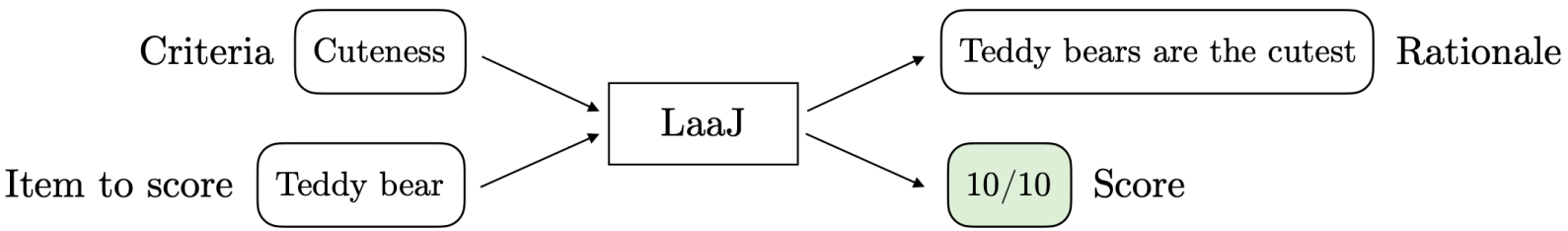
Hybrid approaches are possible: use vector search for quick lookups, then graph reasoning for deeper insights.

LLM-as-a-Judge

LLM-as-a-Judge

LLM-as-a-Judge

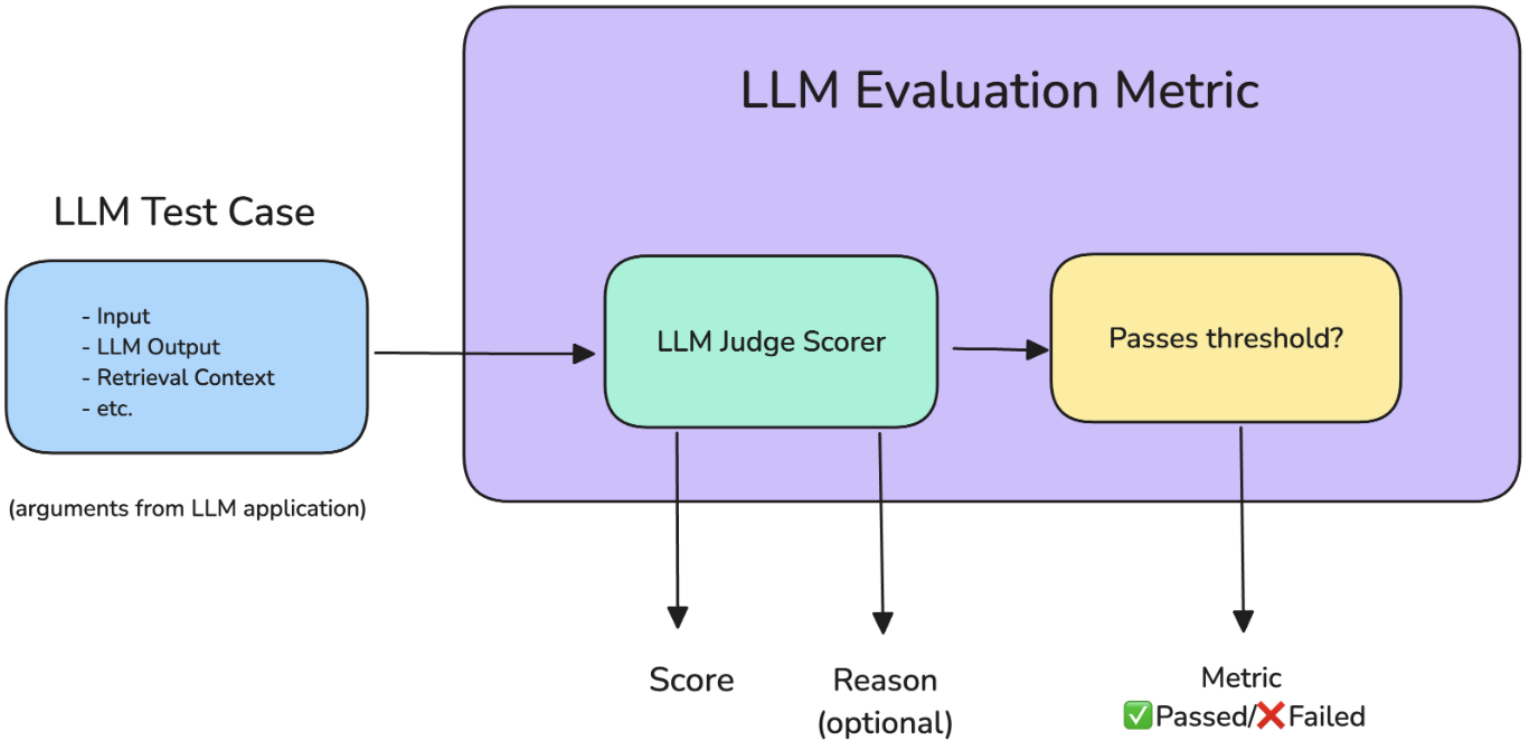
LLM-as-a-Judge is an evaluation framework where a high-performing language model (e.g., GPT 5.5) assesses the outputs of other LLMs or AI Agents based on specific criteria.



The approach aims to replace manual scoring by humans with automated scoring against user-defined evaluation criteria.

LLM-as-a-Judge

Evaluation Metrics/Criteria:



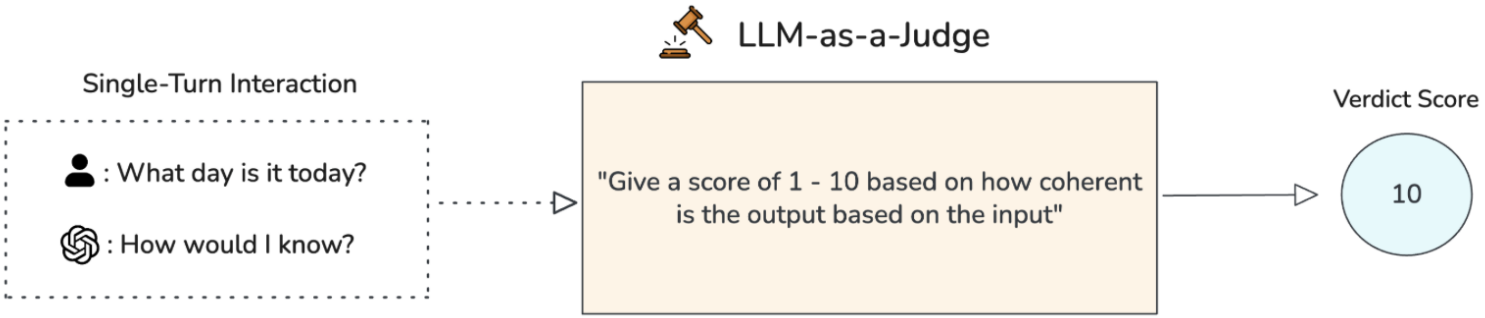
LLM-as-a-Judge

Evaluation Criteria:

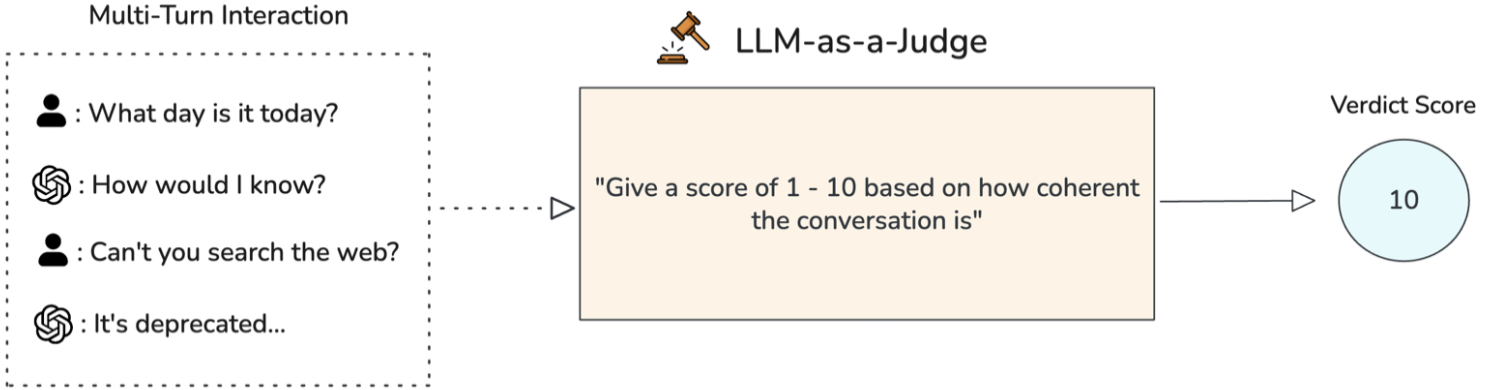
- **Answer Relevancy:** Did the response actually address the user's request?
- **Helpfulness:** Was the answer genuinely useful? Or just pretty sentences, no substance?
- **Faithfulness:** Did it stick to the facts? Or did it wander into hallucinations?
- **Bias:** Was the answer balanced, or did some skew creep in?
- **Correctness:** Was the answer accurate start to finish?

LLM-as-a-Judge

Single-Turn LLM-as-a-Judge

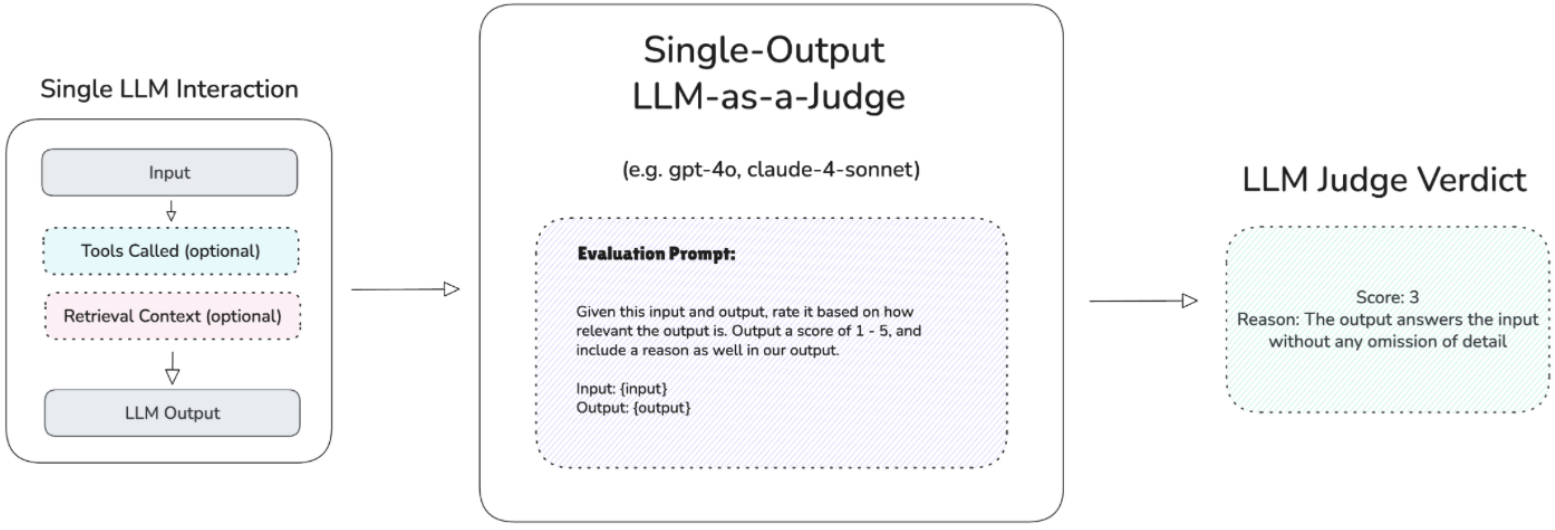


Multi-Turn LLM-as-a-Judge



LLM-as-a-Judge

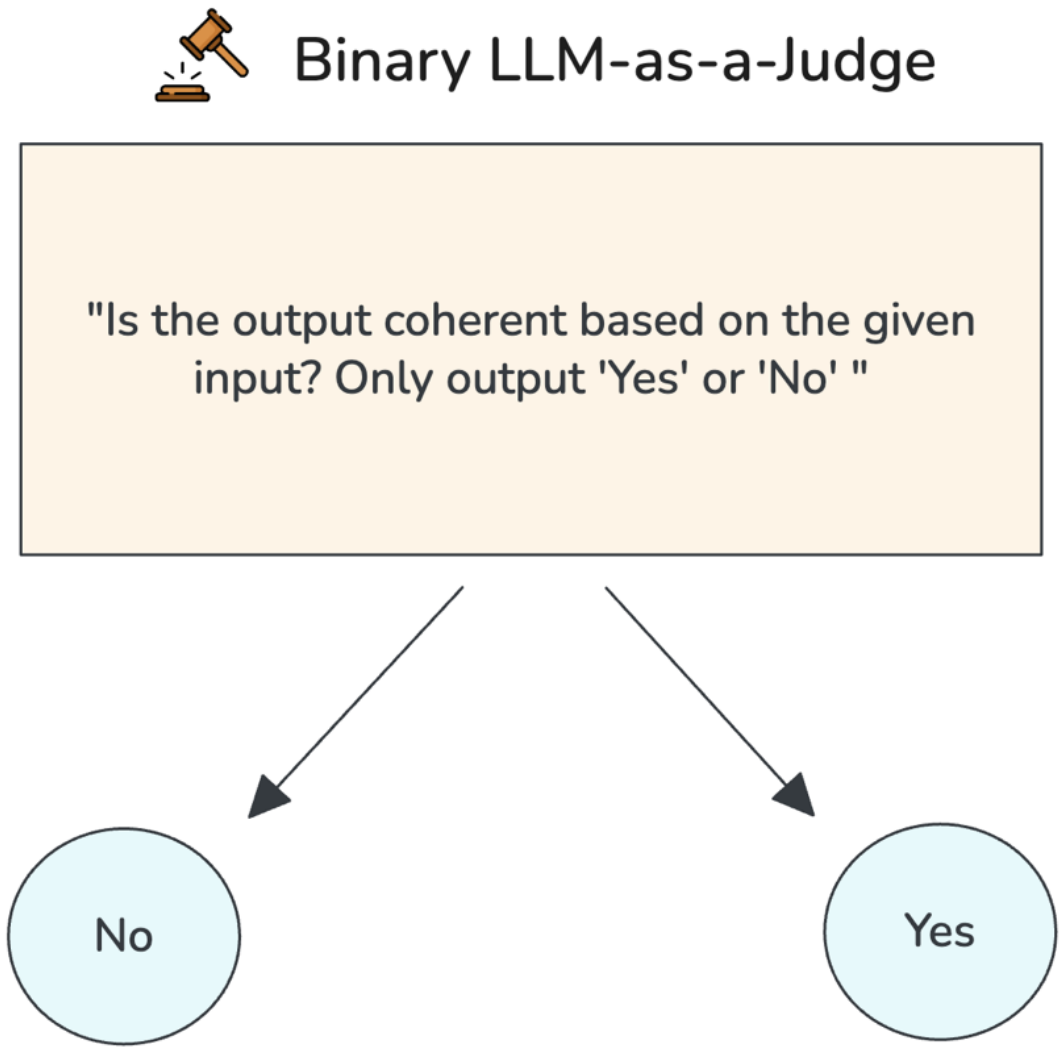
Single-Output LLM as a Judge:



Single-Output LLM-as-a-Judge

LLM-as-a-Judge

Binary LLM-as-a-Judge



LLM-as-a-Judge

Multi-Turn Dialogues between a User and two AI Assistants:

Question: If the FED buys bonds in the secondary market
(A) the money supply will increase. (B) the money supply will decrease. (C) the money supply will not be affected. (D) the discount rate would be affected. Output the answer only.

Assistant A:

```
The Federal Reserve buys
bonds in the secondary
market to increase the
money supply.
```

Assistant B:

```
(A) the money supply will increase.
```

Follow-up Question: How does it affect my daily life? Give 3 examples.

Assistant A:

```
The Federal Reserve buys
bonds in the secondary
market to increase the
money supply. This
affects your daily life
by:
1. Increasing the money
supply.
2. Increasing the money
supply and lowering the
discount rate.
3. Increasing the money
supply and lowering the
discount rate and
increasing the money
supply.
```

Assistant B:

```
When the Federal Reserve buys bonds in the secondary market, it can have
several effects on the economy that can impact your daily life. Here are
three examples:

1. Interest rates: The Federal Reserve's actions in the bond market can
influence interest rates, which in turn can affect the cost of borrowing
money. For example, if the Federal Reserve buys bonds and drives up demand
for them, interest rates on things like mortgages, car loans, and credit
cards may go up, making it more expensive to borrow money.
2. Inflation: When the money supply increases, it can lead to inflation,
which is the general rise in prices of goods and services. This can impact
your daily life by making it more expensive to buy things like groceries,
gas, and other necessities.
3. Employment: The Federal Reserve's actions in the bond market can also
have an impact on employment. For example, if the Federal Reserve's
actions lead to economic growth and increased demand for goods and
services, it can create more job opportunities and boost employment.
```

GPT-4 Judgment:

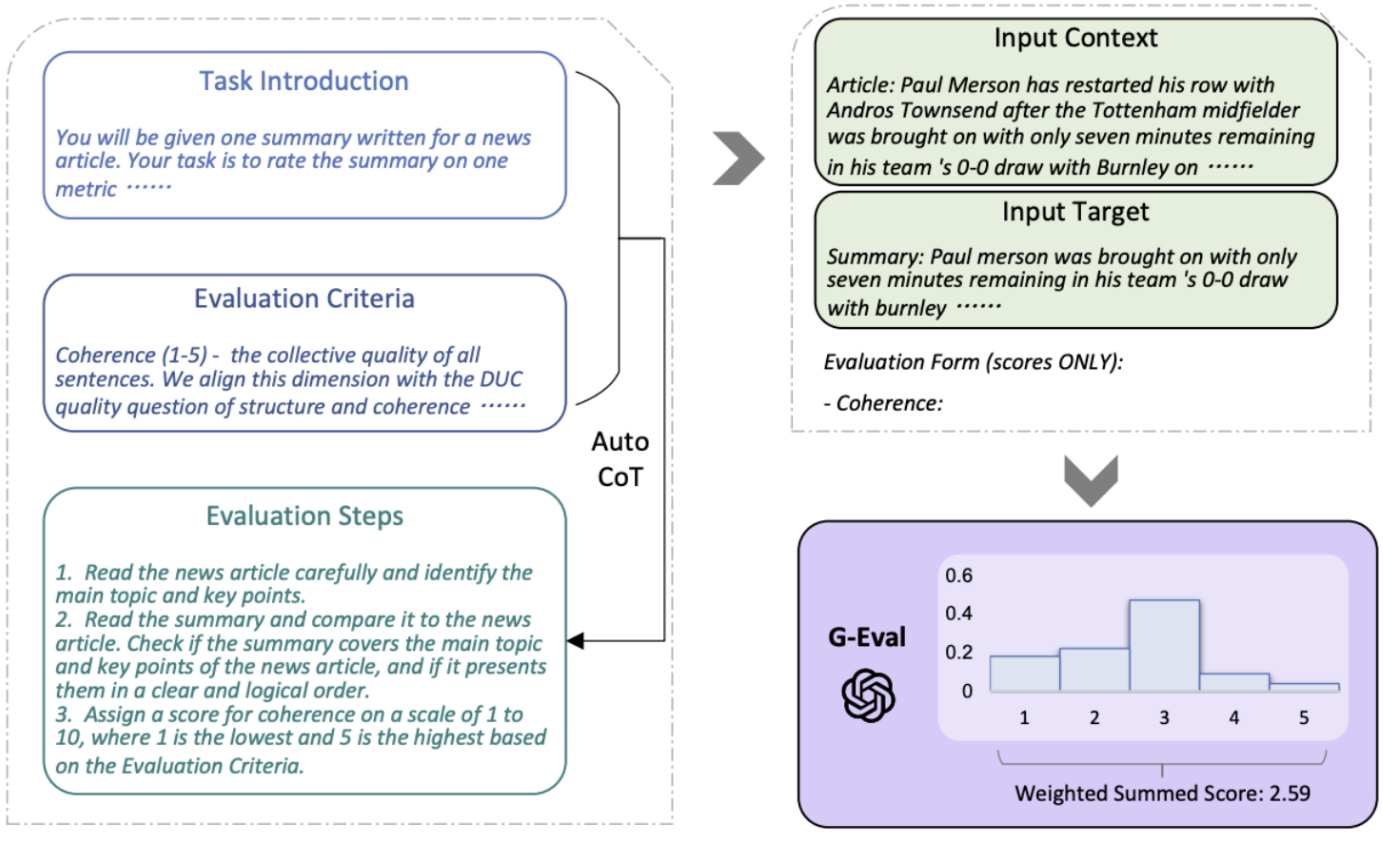
Assistant A provided an incorrect response to the user's question about how the Federal Reserve buying bonds in the secondary market affects daily life. The answer given is repetitive and lacks clear examples of how the action impacts daily life.
On the other hand, **Assistant B** provided a relevant and accurate response to the user's question about the Federal Reserve buying bonds. The answer includes three clear examples of how the action impacts daily life, such as interest rates, inflation, and employment.

Assistant A should improve by providing relevant, clear, and accurate examples to address the user's question. **Assistant B**'s response is more helpful, relevant, and detailed.

Figure 1: Multi-turn dialogues between a user and two AI assistants—LLaMA-13B (Assistant A) and Vicuna-13B (Assistant B)—initiated by a question from the MMLU benchmark and a follow-up

LLM-as-a-Judge

G-Eval Algorithm:



G-Eval algorithm

LLM-as-a-Judge

Weaknesses of LLM as a Judge:

- **Can't Make Up Their Minds:** Their scores are non-deterministic, which means that for a given LLM output their are evaluating, the scores might be different depending on the time of day. You'll need a good way such as **DAG** to make them deterministic if you want to rely fully on them.
- **Narcissistic Bias:** It has been shown that LLMs may favor the answers generated by themselves. We use the word "may" because research has shown that although GPT-4 and Claude-v1 favors itself with a 10% and 25% higher win rate respectively, they also favor other models and GPT-3.5 does not favor itself.
- **More is More:** We humans all know the phrase less is more, but LLM judges tend to prefer more verbose text over more concise ones. This is a problem in LLM evaluation because LLM computed evaluation scores might not accurately reflect the quality of the LLM generated text.
- **Not-so-Fine-Grained Evaluation Scores:** LLMs can be reliable judges when making high-level decisions, such as determining binary factual correctness or rating generated text on a simple 1—5 scale. However, as the scoring scale becomes more detailed with finer intervals, LLMs are more likely to produce arbitrary scores, making their judgments less reliable and more prone to randomness.
- **Position Bias:** When using LLM judges for pairwise comparisons, it has been shown that LLMs such as GPT-4 generally prefer the first generated LLM output over the second one.

Working with

LangChain

Working with LangChain

LangChain

LangChain (2022) is a framework that makes it easier to build applications using large language models (LLMs) by connecting them with data, tools and APIs.

Philosophy and Software Overview:

- Rather than treat LLMs as standalone services, LangChain's philosophy centers on composability and modularity.
- Provides a full stack for application development: build with a composable framework, run with LangGraph platform.
- The core framework is open source. Additional enterprise features are commercially available.

Working with LangChain

Challenge 1. Context Window Limitations:

- Every LLM has a fixed context window – the max no tokens it can process at once – typically, 2k to 128k tokens.
- Practical challenges, e.g., long documents must be chunked to fit within the context window.

Challenge 2. Limited Tool Orchestration:

- Modern LLMs may lack the tools to organize and execute complex workflows.

Challenge 3. Task Coordination:

- Complex workflows require structured control mechanisms.

Source: Generative AI with LangChain, Chapter 1.

Working with LangChain

Benefit 1. Composable Workflows:

- The LangChain Expression Language (LCEL) provides for complex tasks to be broken down into modular components, which can be assembled and reconfigured.
- Composability enables systematic reasoning through orchestration of multiple processing steps.

Benefit 2. Integration Ecosystem:

- Abstract interfaces for all generative AI components (LLMs, embeddings, vector databases, document loaders, search engines.

Working with LangChain

Benefit 3. Memory and State Management

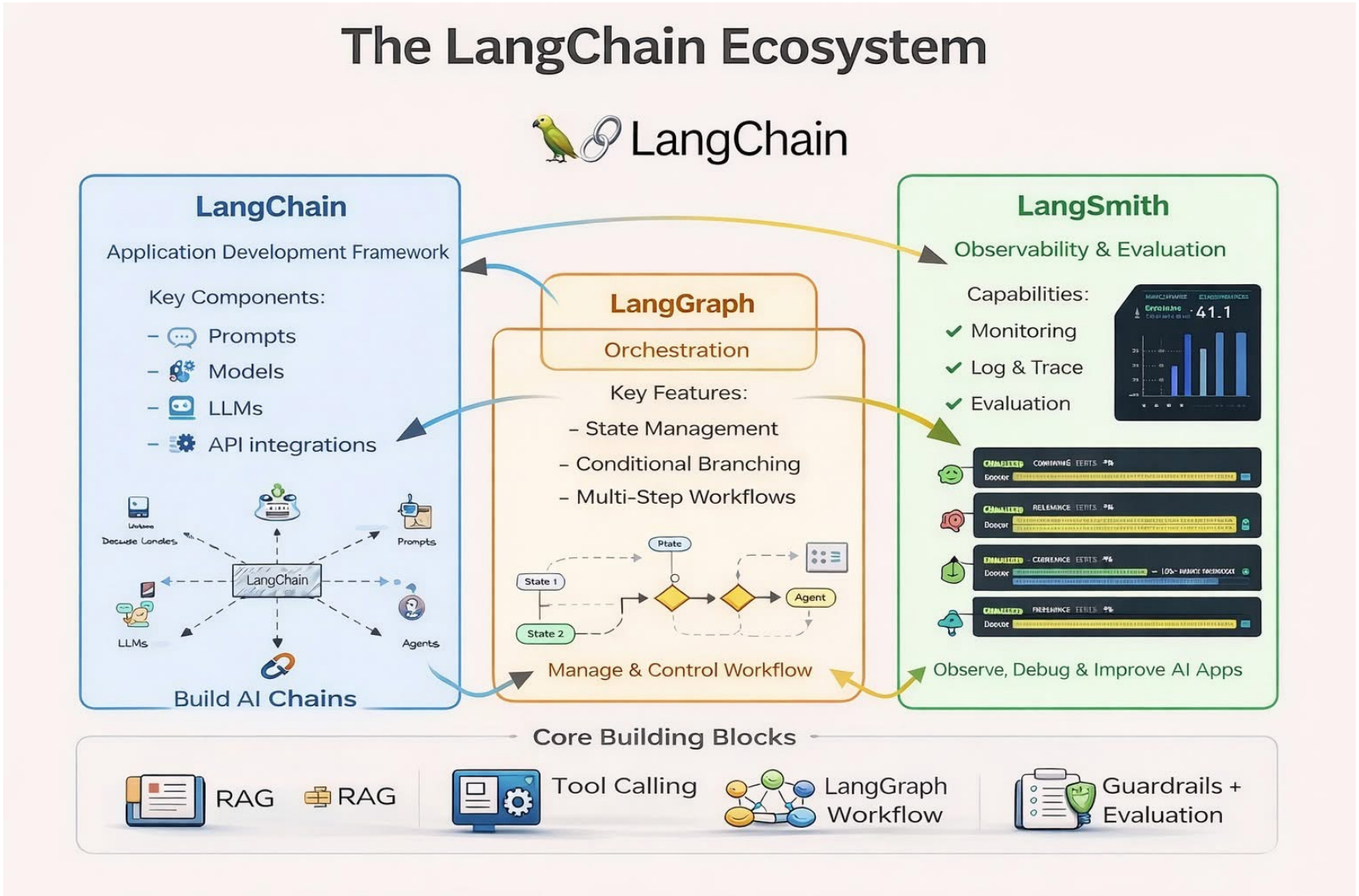
- Driven by need for persistent context across interactions.
- Maintain conversation history and application state.

Benefit 4. Unified Model Access:

- Consistent interfaces to diverse languages and embedding models.
- Support for seamless switching among providers while maintaining application logic.

Source: Generative AI with LangChain, Chapter 1.

Working with LangChain



Working with LangChain

Support for Application Development:



Working with LangChain

Key Components:

- **Chains:** Sequence of steps where each step can use an LLM, process data or call tools. Multi-step chains combine actions.
- **Prompt Management:** Uses templates to simplify design and mangement of prompts.
- **Agents:** LLM driven components that decide which actions to take, such as calling tools or APIs.
- **Vector Database:** Stores data as vectors to enable similarity search, helping retrieve relevant information for tasks like document search and RAG.
- **Models:** Supports LLMs like OpenAI and Hugging Face.
- **Memory Management:** Maintains context from past interactions, enabling better responses in conversations/tasks.

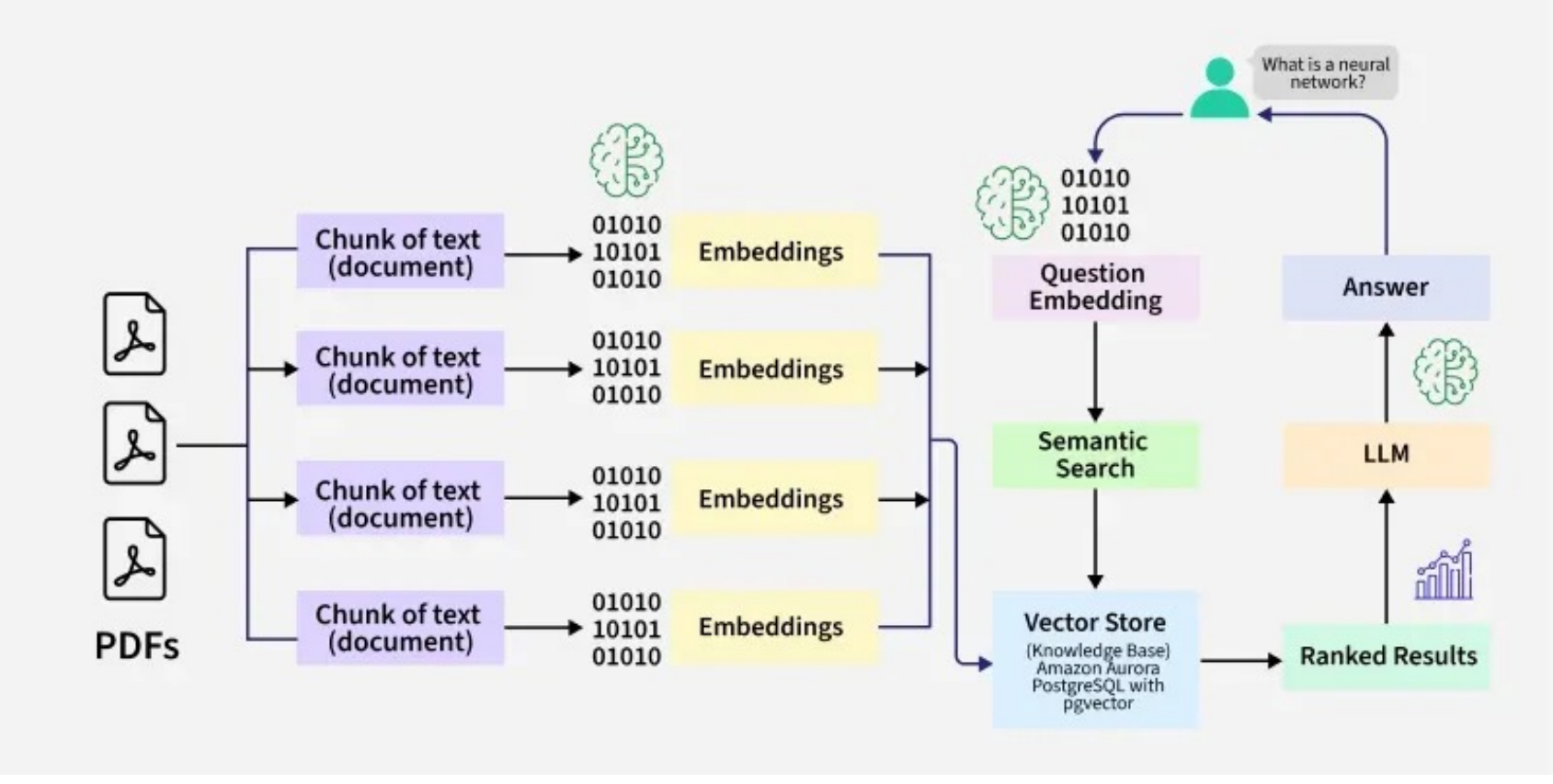
Working with LangChain

LangChain Use Cases:

1. Question and Answer with RAG
2. Chatbots and Conversational Agents.
3. Query Analysis.
4. Knowledge Graphs.
5. Code Generation and Programming Assistance.
6. Summerization.
7. Interacting with APIs.
8. Multi-Agent Systems.

Working with LangChain

LangChain Workflow: Enables RAG by combining document processing, vector storage, and LLMs to generate accurate, context aware responses.



Working with LangChain

Step-by-step Procedure:

1. **Document Processing:** Split pdf documents into chunks for efficiency of processing.
2. **Create Embeddings:** Each chunk is converted into embeddings that capture its semantic meaning.
3. **Vector Store:** These embeddings are stored in a vector database, creating a searchable knowledge base.
4. **User Query:** The process starts when a user submits a question or request as input.

   For example, a user might ask, "What's the weather like today?" This query serves as the input to the LangChain pipeline.

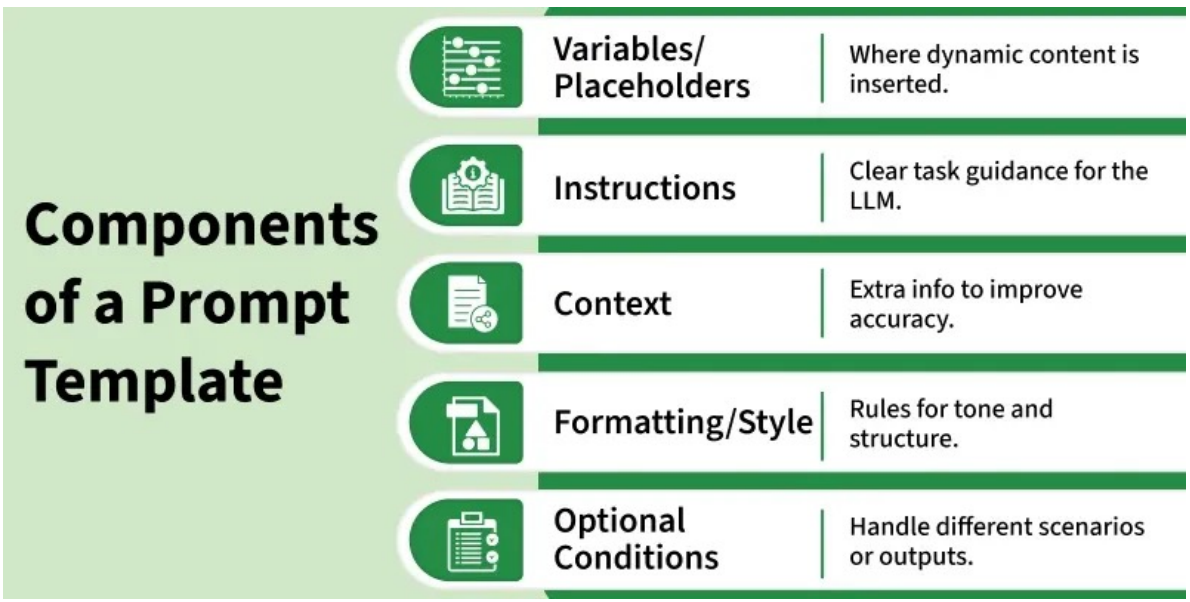
Working with LangChain

1. **Vector Representation:** Once the query is received, LangChain converts it into a vector representation using embeddings.
2. **Similarity Search:** This vector is compared with vectors stored in a database to find the most relevant matches based on meaning.
3. **Fetch Relevant Information:** The system retrieves the most relevant data or context from the vector database to support the response.
4. **Generate a Response:** The retrieved context is passed to a language model, which processes it and generates a meaningful answer.

Working with LangChain

Prompts and Prompt Templates

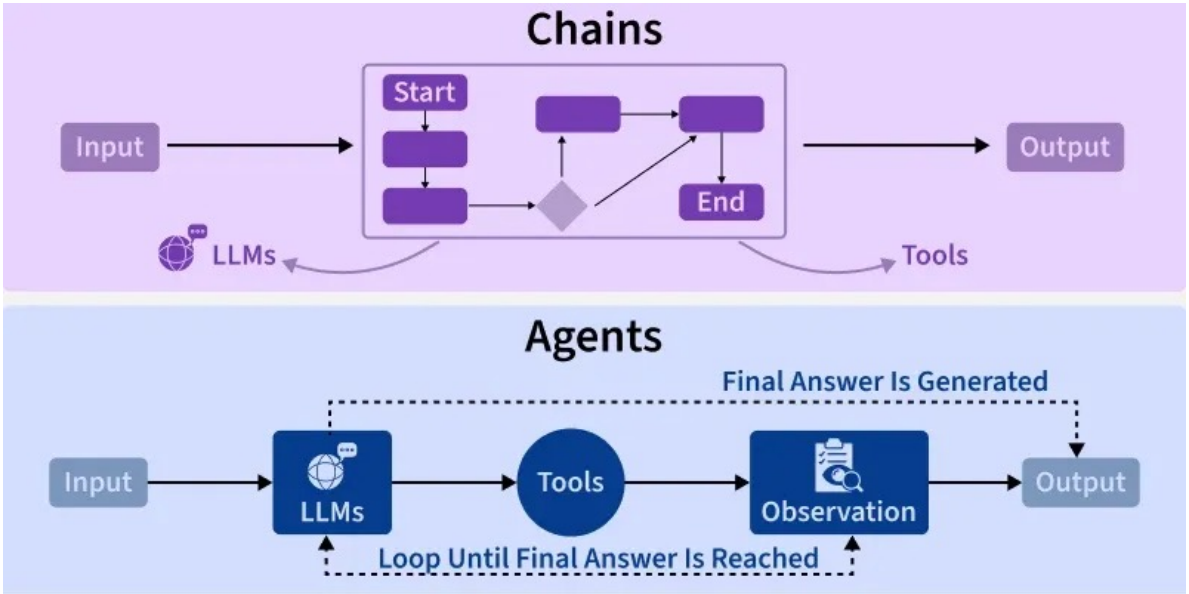
Prompts are the foundation of LLM interaction. Prompt templates are reusable formats for instructing LLMs using placeholders to insert variable information.



Working with LangChain

Building Chains and Agents

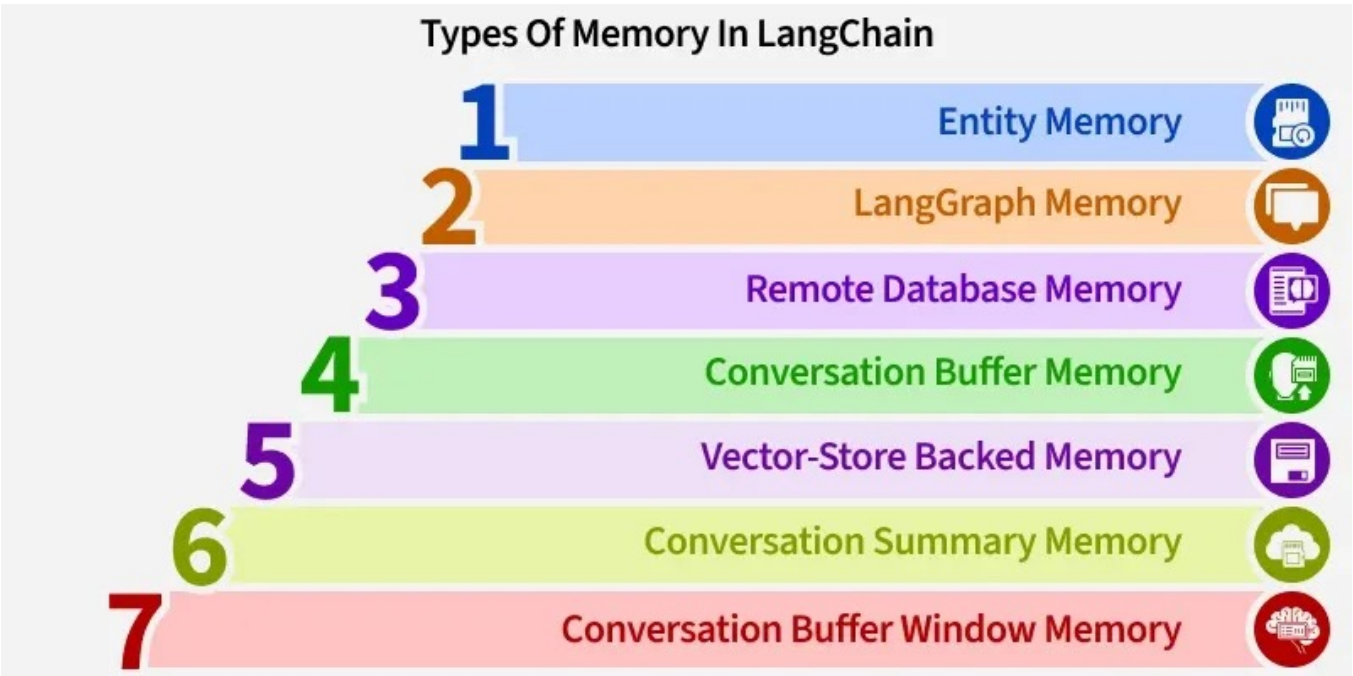
Chains connect multiple steps of a solution procedure into workflows. Agents make decisions to build task-oriented multi-step AI applications.



Working with LangChain

Memory

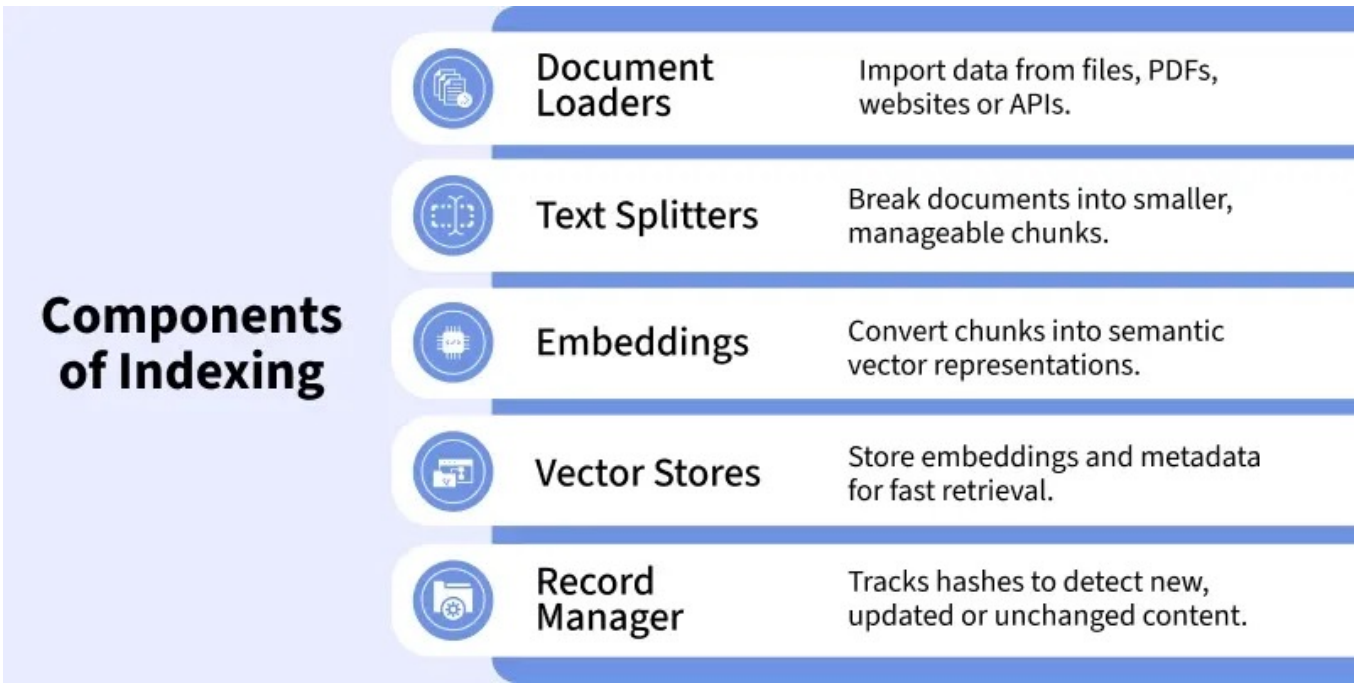
Memory makes applications more human-like by retaining context across coversations.



Working with LangChain

Indexes

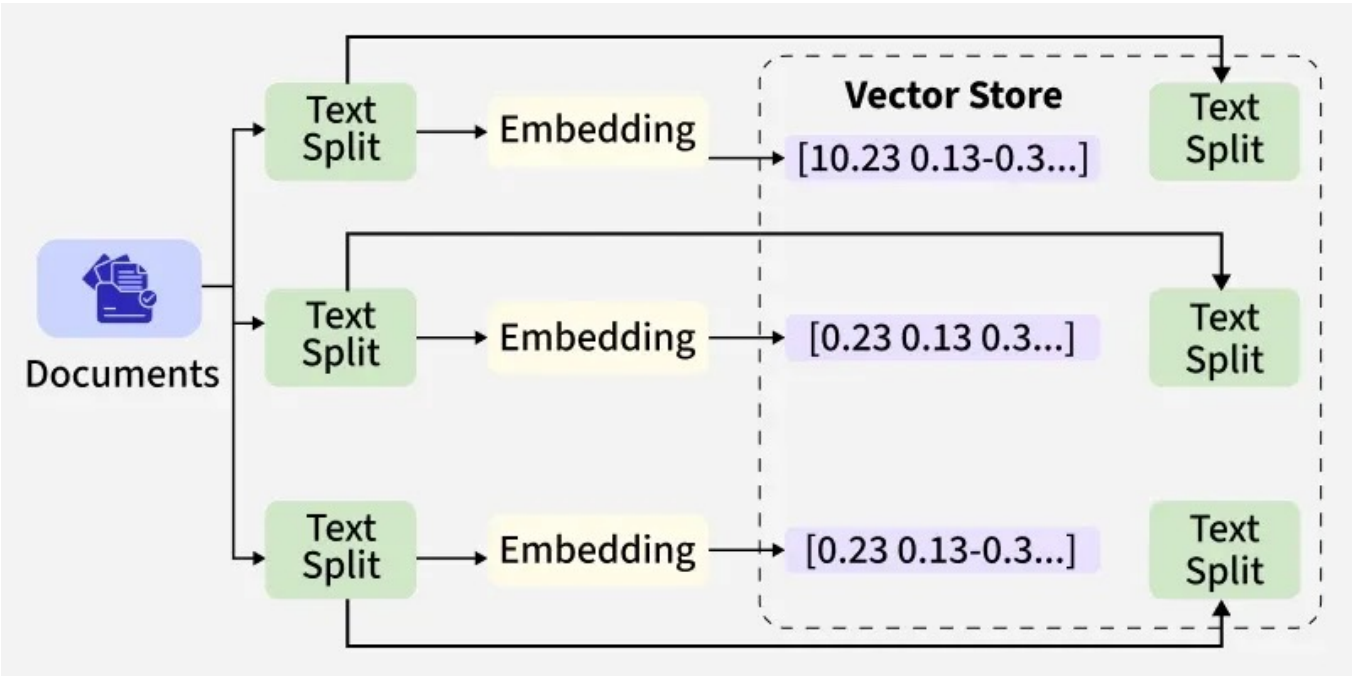
Indexing is the proces of organizing documents in a vector database such that the language model can quickly find and use them.



Working with LangChain

Vector Stores

Vector stores are specialized databases that store embeddings and provide fast similarity search.



Applications

LLM Applications

LangChain Applications

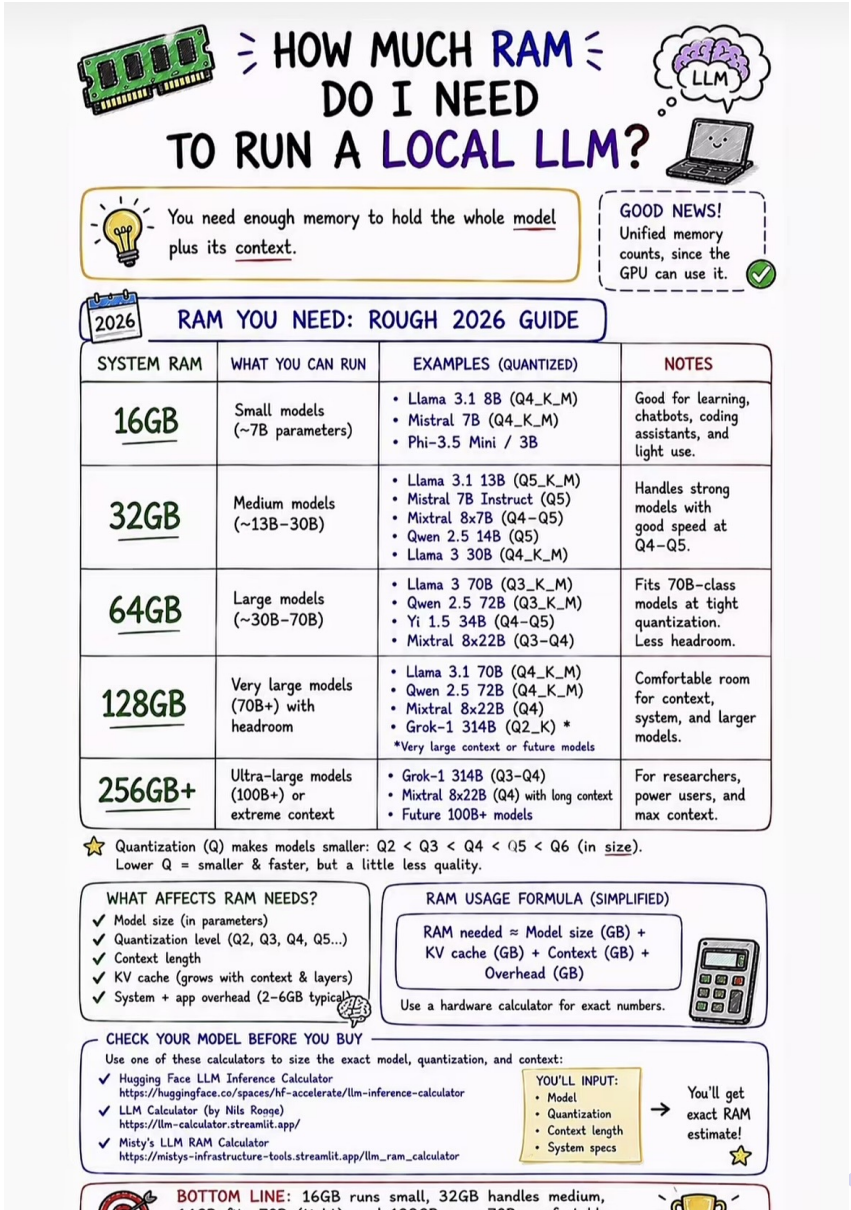
References

- Amidi A., and Amidi S., Super Study Guide: Transformers and Large Language Models (First Edition), ISBN 979-8-836693-31-2, 2024.
- Auffarth B., and Kuligin L., Generative AI with LangChain (Second Edition), Packt Publishing, 2025.
- Glassner A., Deep Learning: A Visual Approach, Starch Press, San Francisco, 2021.
- Lakshmanan V., and Hapke H., Generative AI Design Patterns: Solutions to Common Challenges when building GenAI Agents and Applications, O'Reilly, 2026.
- Negro A., Futia G., Kus V., and Montagna F., Knowledge Graphs and LLMs in Action, Manning Publications, 2026.
- Raieli S., and Luculano G., Building AI Agents with LLMs, RAG and Knowledge Graphs, Packt Publishing, 2025.
- Shetty R., Modeling Intelligence: The Hidden Mathematical Structures behind Modern AI, Independently Published, 2026.

Appendix A

Cheatsheets

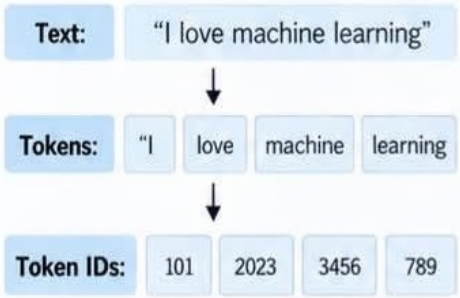
LLM Hardware Requirements



LLM Formulae Explained

1 Tokenization
(Convert Text to Tokens)

Splits text into smaller units called tokens which are then converted into numbers (token IDs).

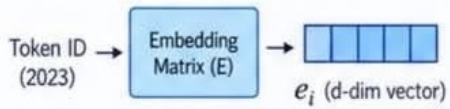


2 Token Embedding
(Convert Tokens to Vectors)

Maps each token ID to a dense vector using an embedding matrix.

$$e_i = E[t_i]$$

Where:
- t_i = token ID
- $E \in \mathbb{R}^{V \times d}$ = embedding matrix (V = vocabulary size, d = embedding dimension)
- $e_i \in \mathbb{R}^d$ = token embedding vector



3 Positional Encoding
(Add Word Position Information)

Adds information about the position of each token in the sequence.

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

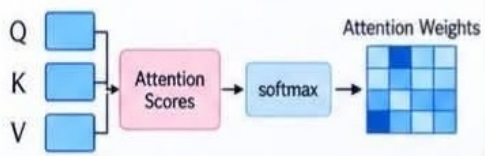
Where:
- pos = position of the token
- i = dimension index
- d = embedding dimension

4 Self-Attention
(Find Important Tokens)

Computes how much each token should attend to every other token in the sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:
- Q = Query matrix
- K = Key matrix
- V = Value matrix
- d_k = dimension of keys/queries



5 Softmax
(Convert Scores to Probabilities)

Converts attention scores or model outputs into a probability distribution.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Where:
- z_i = input score for token i
- Output values are between 0 and 1 and sum to 1.



6 Cross-Entropy Loss
(Train the Model)

Measures how different the predicted distribution is from the true distribution.

$$L = -\sum_{i=1}^{V} y_i \log(\hat{y}_i)$$

Where:
- y_i = true probability (one-hot, 0 or 1)
- $\hat{y}_i$ = predicted probability
- V = vocabulary size

Token	True (y_i)	Predicted ($\hat{y}_i$)	Loss ($-\log(\hat{y}_i)$)
I	0	0.1	–
love	0	0.2	–
machine	1	0.6	0.51
learning	0	0.1	–

LLM Formulae Explained

7 Next Token Prediction
(Autoregressive Modeling)

Predicts the next token based on previous tokens in the sequence.

$$P(w_t \mid w_1, \ldots, w_{t-1}) = \text{softmax}(z_t)_{w_t}$$

Where:

- w_t = next token
- $w_1, \ldots, w_{t-1}$ = previous tokens
- z_t = model output (logits) at position t

Input: "I love machine"

↓

Model predicts: "learning"

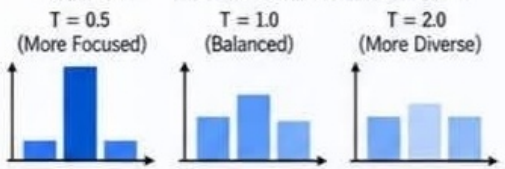
8 Temperature Sampling
(Control Output Randonmess)

Adjusts the sharpness of the probability distribution during generation.

$$P_i = \frac{e^{z_i/T}}{\sum_{j=1}^{n} e^{z_j/T}}$$

Where:

- z_i = model logits
- T = temperature (T > 0)
- Lower T → more focused (less random)
- Higher T → more diverse (more random)



9 Perplexity
(Evaluate Model Performance)

Measures how well the model predicts a sequence. Lower perplexity generally indicates better next-token prediction.

$$\text{Perplexity} = \exp\left(\frac{1}{N}\sum_{i=1}^{N} L_i\right)$$

Where:

- L_i = cross-entropy loss for token i
- N = total number of tokens

Lower perplexity generally indicates better next-token prediction.



LLM GenAI Formulae

1. What is Generative AI?

Generative AI learns an approximation of the data distribution from training data and generates new samples based on the patterns it has learned.

$$x \sim p_\theta(x)$$

The model learns $p_\theta(x)$ and can generate new data x based on the patterns in the training data. The generated samples are not guaranteed to be exact copies of the training data or always correct.

Examples:



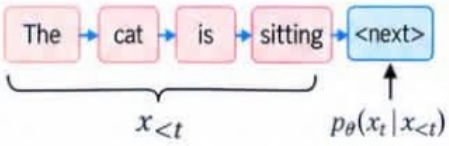
2. Language Model

An autoregressive language model predicts the next token based on the preceding tokens in a sequence.

$$p_\theta(x_1, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t})$$

The probability of a sequence is factorized into next-token probabilities.

Example (next-token prediction):

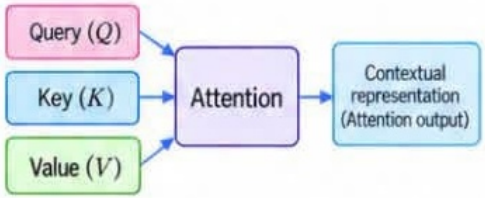


3. Transformer Attention

Self-attention lets the model focus on relevant tokens when generating each token.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

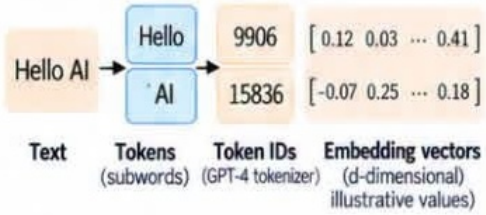
- Q : Query (what we are looking for)
- K : Key (what we can match against)
- V : Value (information to use)



4. Tokenization & Embeddings

Text is split into tokens. Each token is assigned an ID, which is mapped to a dense embedding vector.

Example (using GPT-4 tokenizer):



5. Training Objective

Language models are usually trained using a cross-entropy loss to predict the next token.

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t})$$

- This is the average next-token negative log-likelihood loss.
- It measures the difference between the true next token and the model's predicted distribution.
- The model adjusts parameters θ to minimize this loss (e.g., using gradient descent).

6. Temperature & Sampling

Before applying softmax, logits are scaled by a temperature T to control randomness.

$$p_i = \text{softmax}\left(\frac{z_i}{T}\right) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

- Lower T (e.g., 0.2 – 0.7): concentrates probability on higher-scoring tokens (more predictable output).
- Higher T (e.g., 1.0 – 2.0): produces a flatter distribution and can increase output diversity, but may reduce coherence.
- The effect also depends on the model's logits and the sampling strategy.

LLM GenAI Formulae

7. Diffusion Models

Forward diffusion gradually adds noise to training data. Generation starts from noise and uses a learned reverse process to produce a sample.

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$$

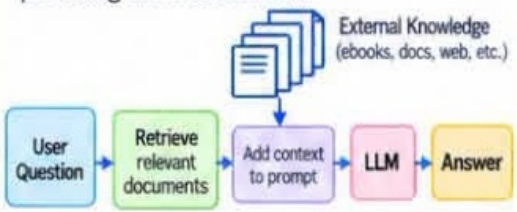
$$\epsilon \sim \mathcal{N}(0, I)$$



Random noise (x_T) → Partially denoised sample (x_t) → Generated image (x_0)

8. RAG (Retrieval-Augmented Generation)

Combines an LLM with external knowledge by retrieving relevant documents and providing them as context.



RAG grounds the model's answers in retrieved content, but it does not guarantee correctness. Always verify important information.

9. Fine-tuning, Prompting & Evaluation

Prompting (Use instructions)

- Provide clear instructions, examples or context in the input.
- No model weights are updated.
- Works well for many general tasks.

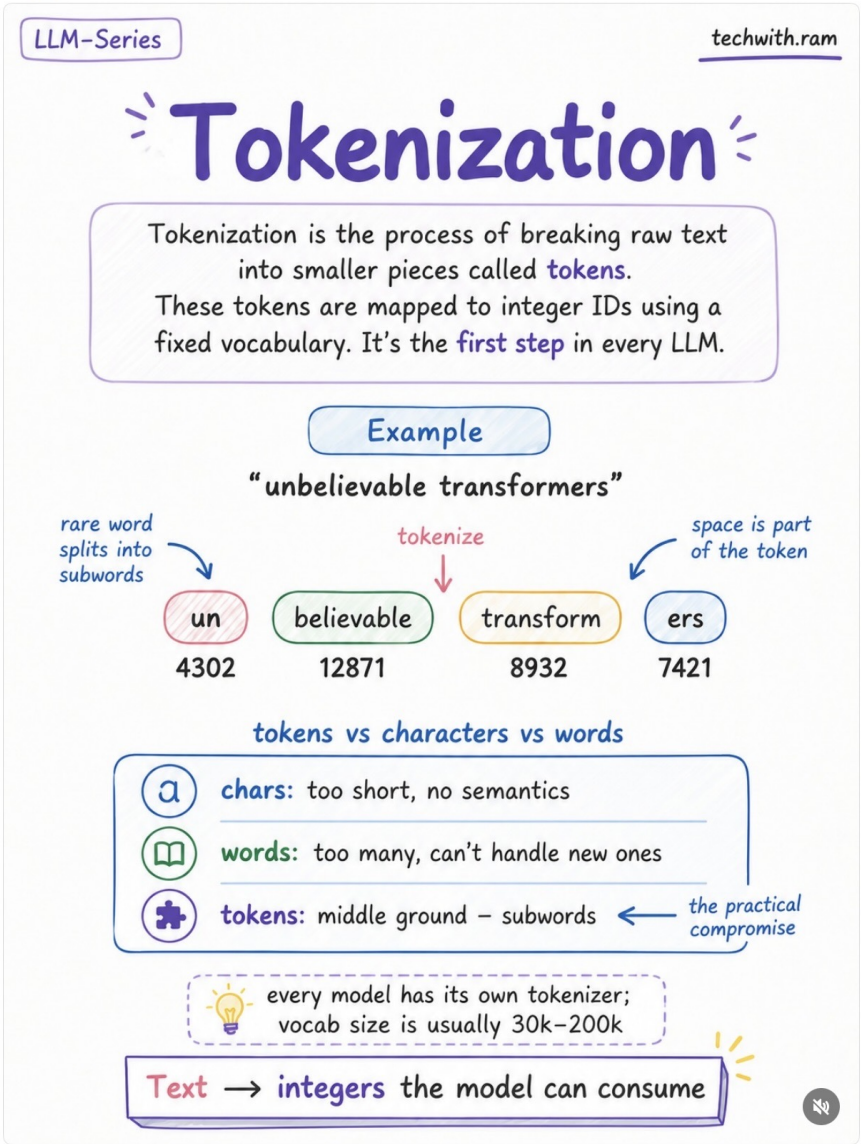
Fine-tuning (Update model weights)

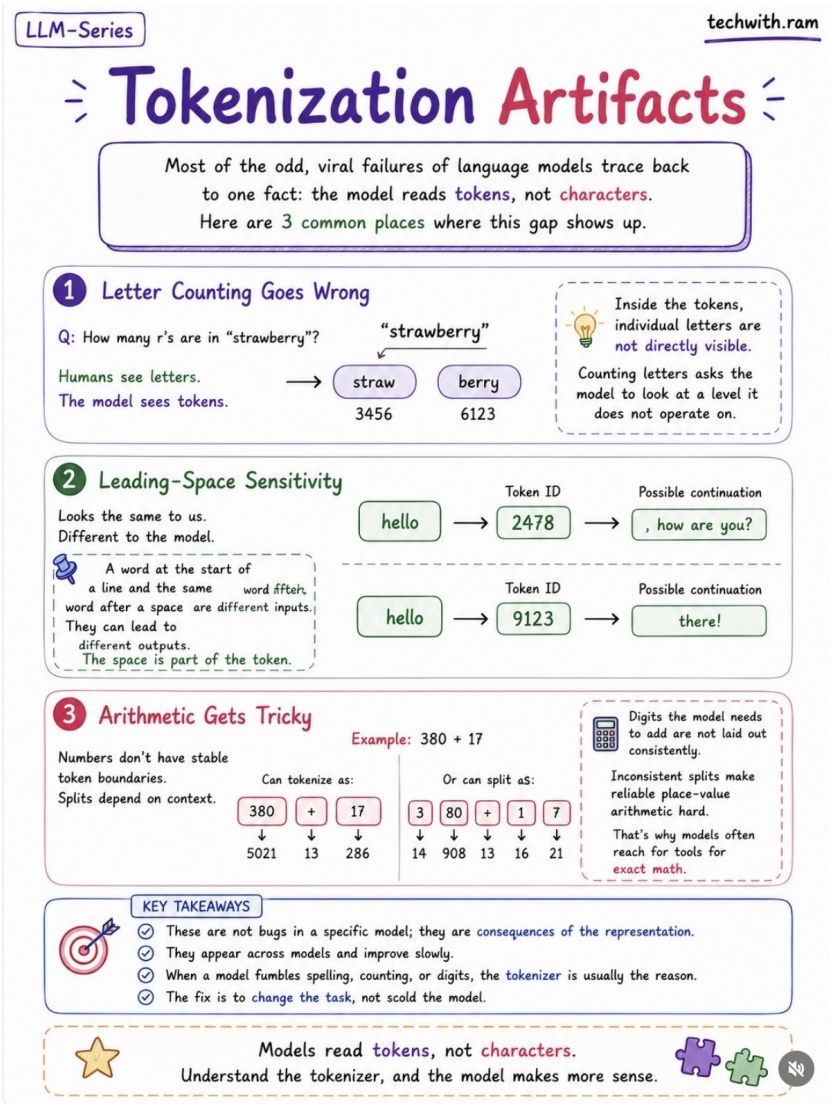
- Further train the model on domain-specific data.
- Helps the model learn your specific style or knowledge.
- Requires labeled data and compute.

Evaluation (Test quality, factuality, safety)

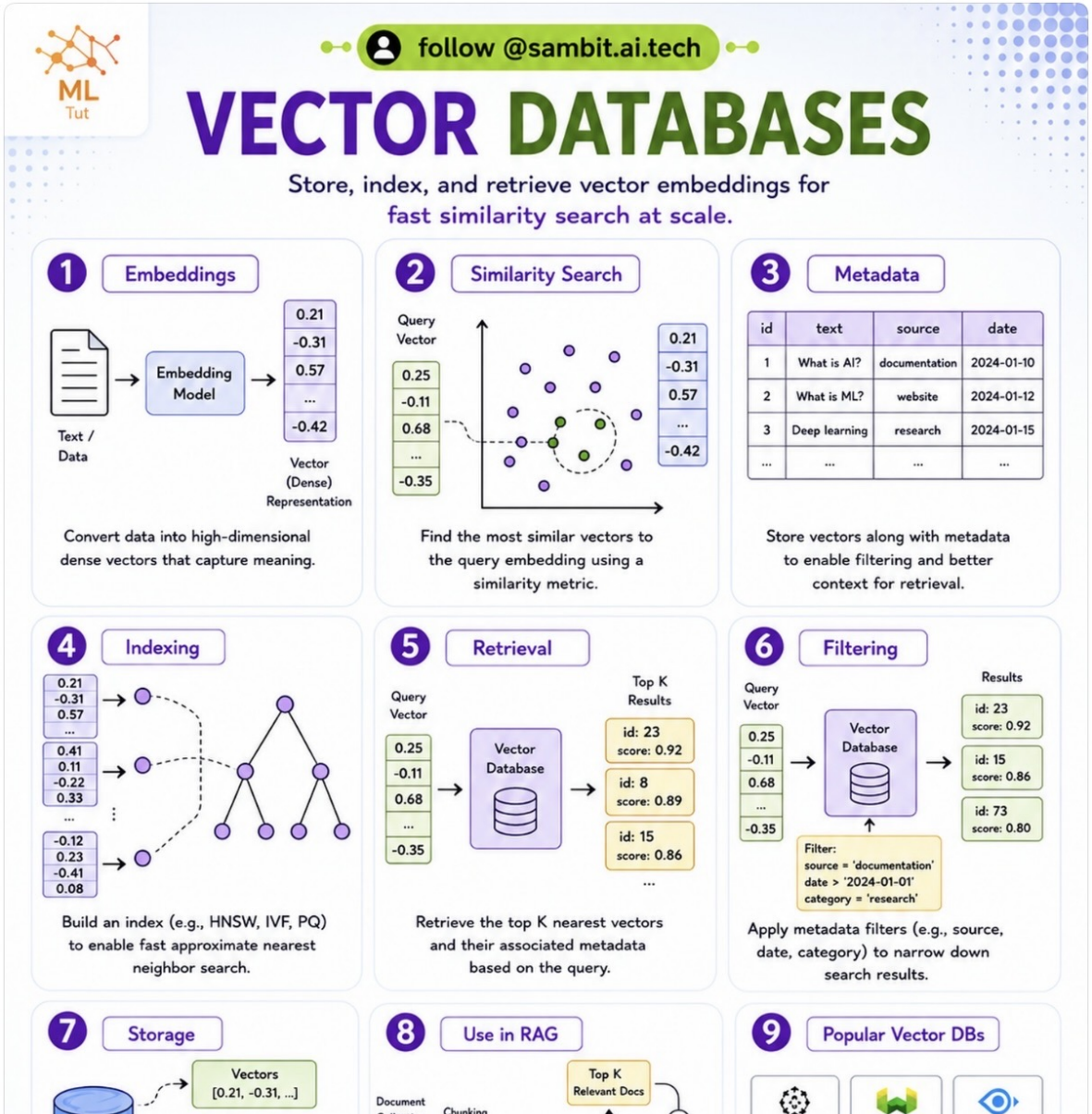
- Check helpfulness, relevance and accuracy.
- Test factuality and consistency.
- Evaluate safety (e.g., harmful or biased content).
- Use benchmarks and human review.

LLM Tokenization

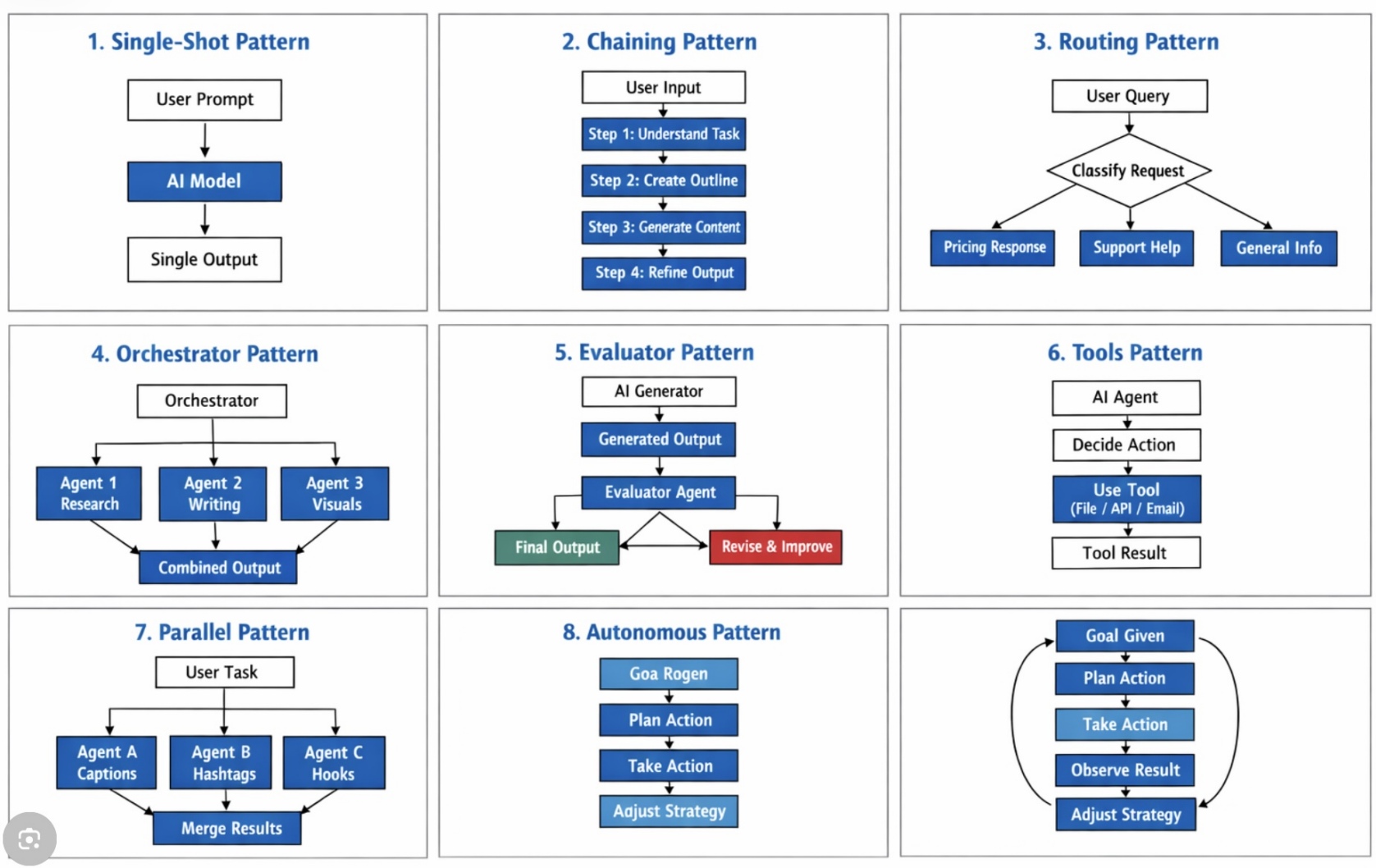




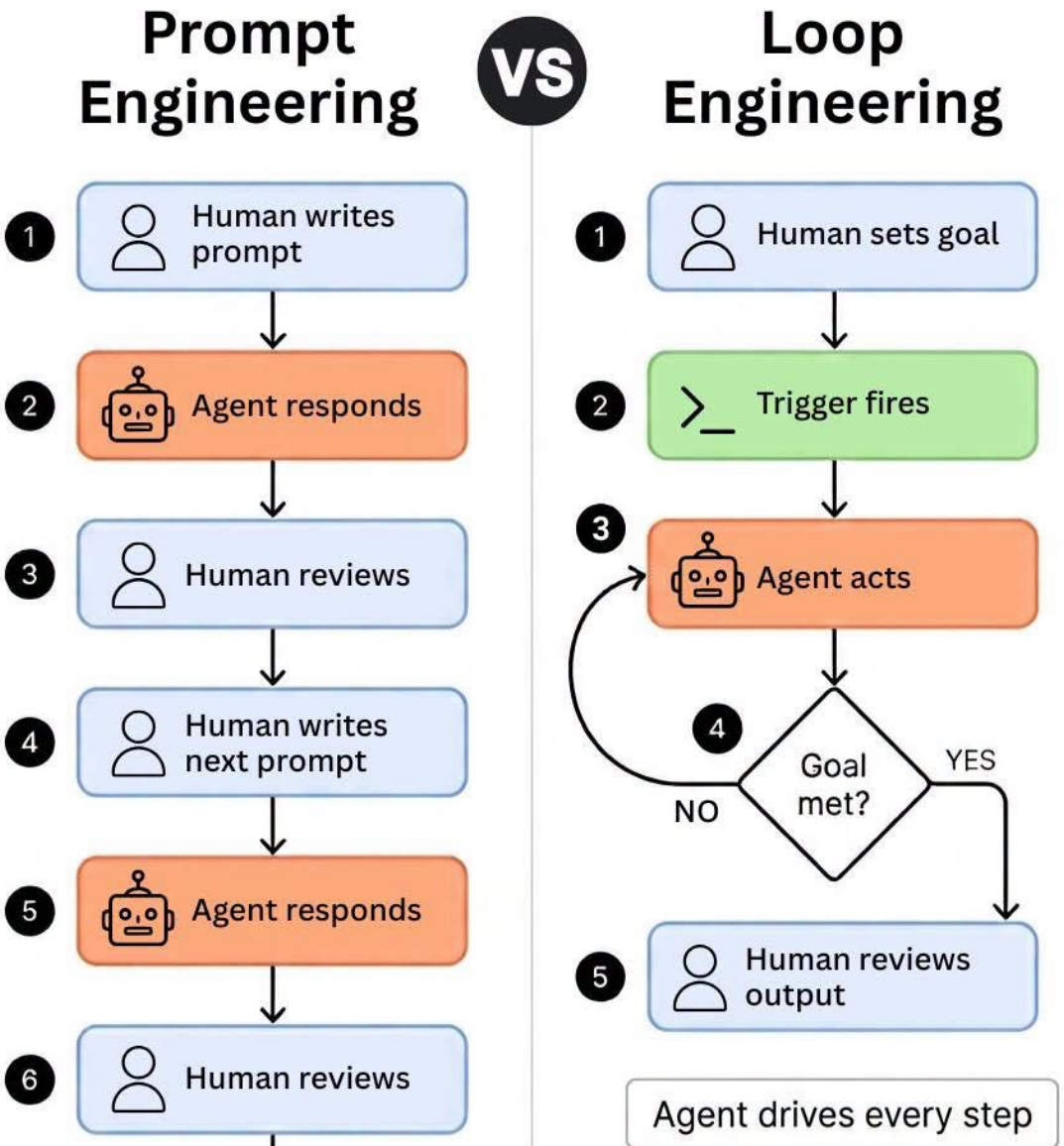
LLM Vector Databases



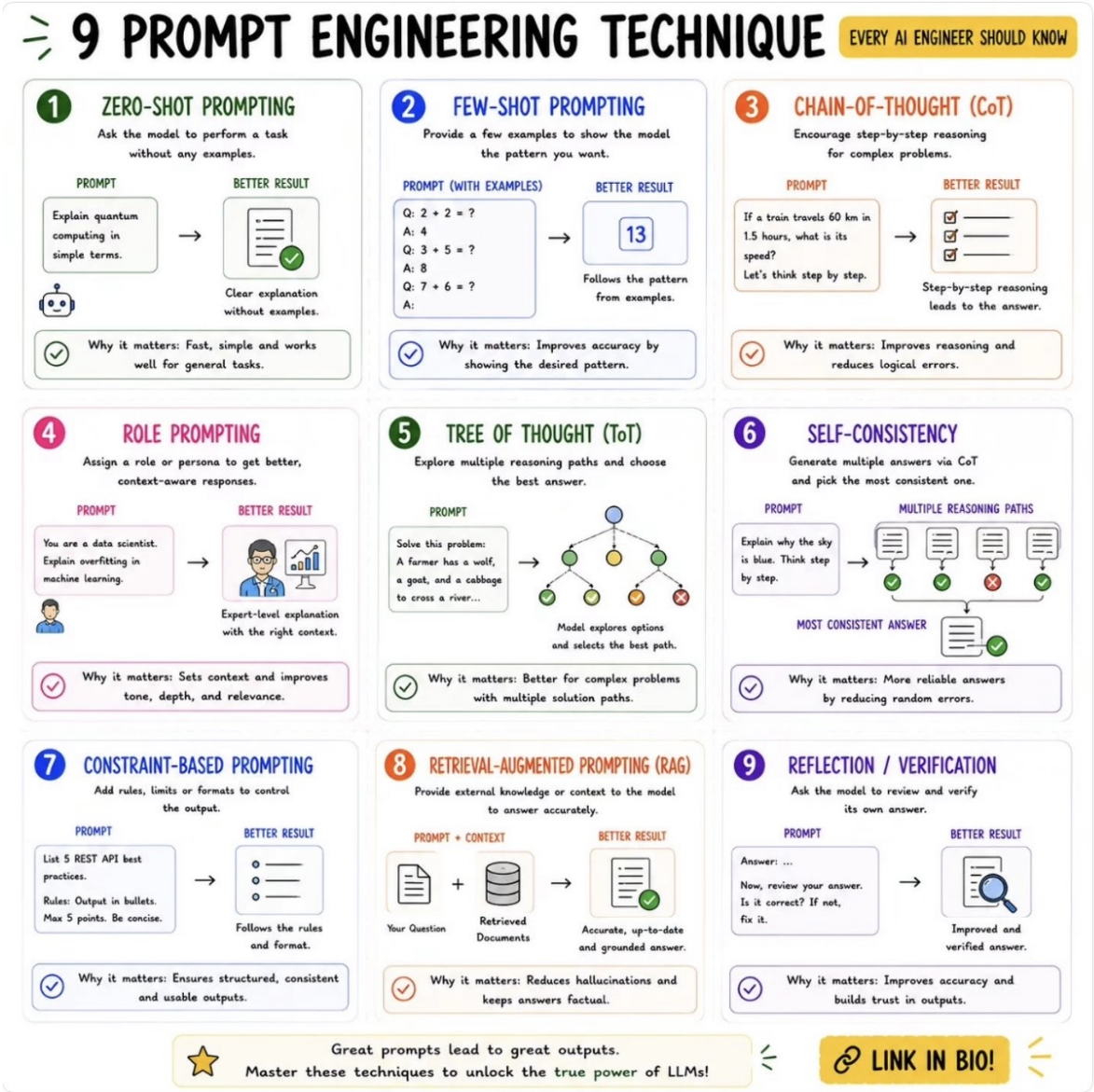
LLM Prompt Engineering



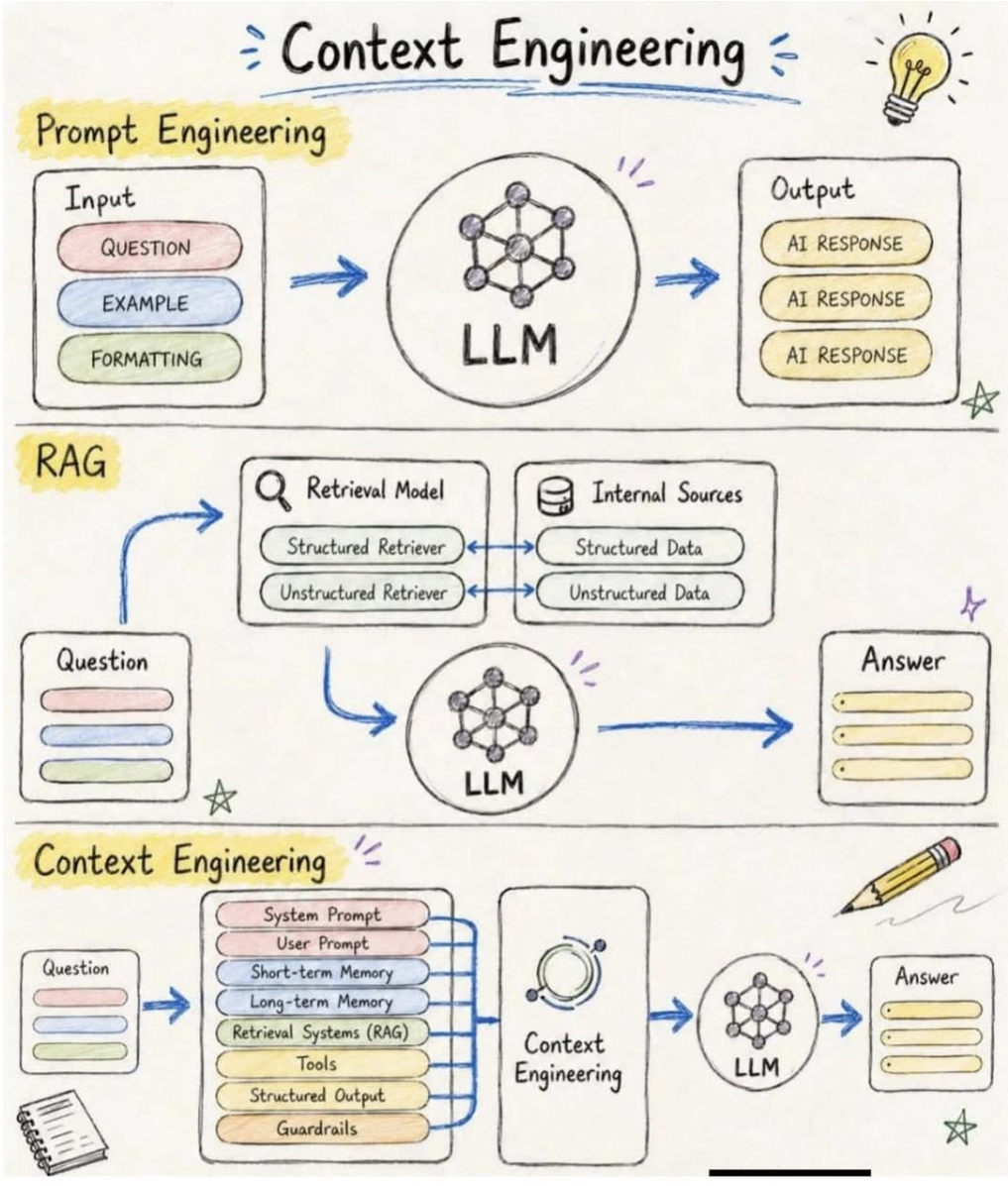
LLM Prompt Engineering



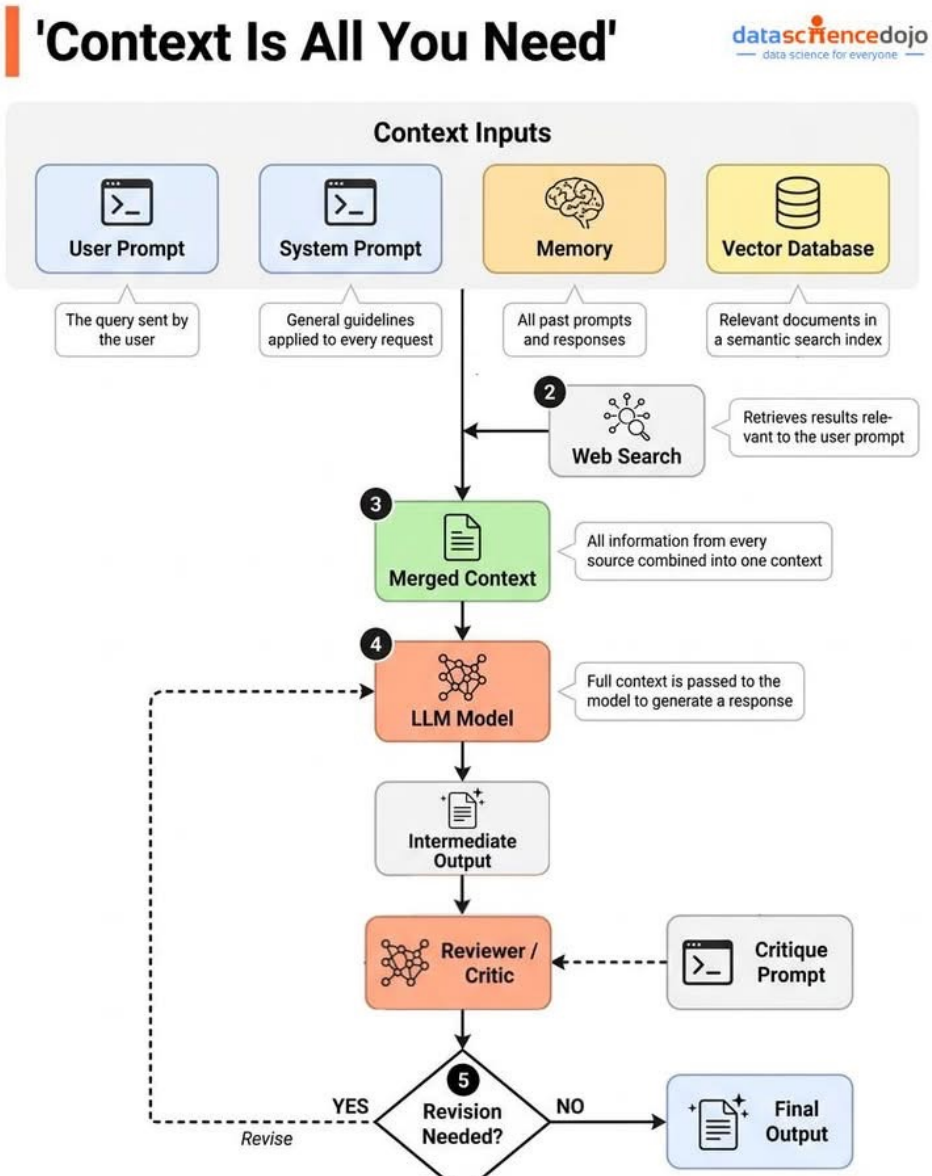
LLM Prompt Engineering



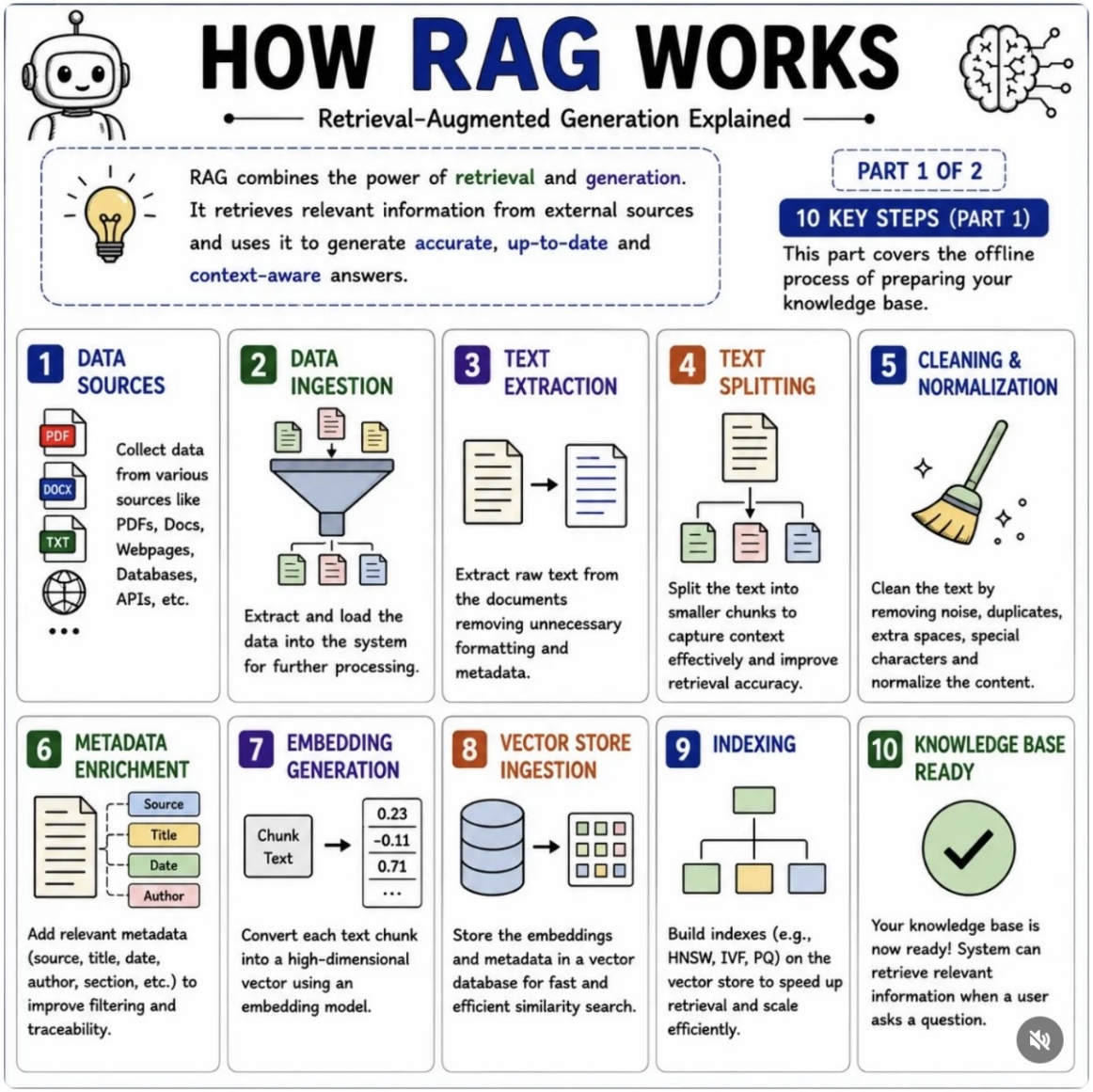
LLM Context Engineering



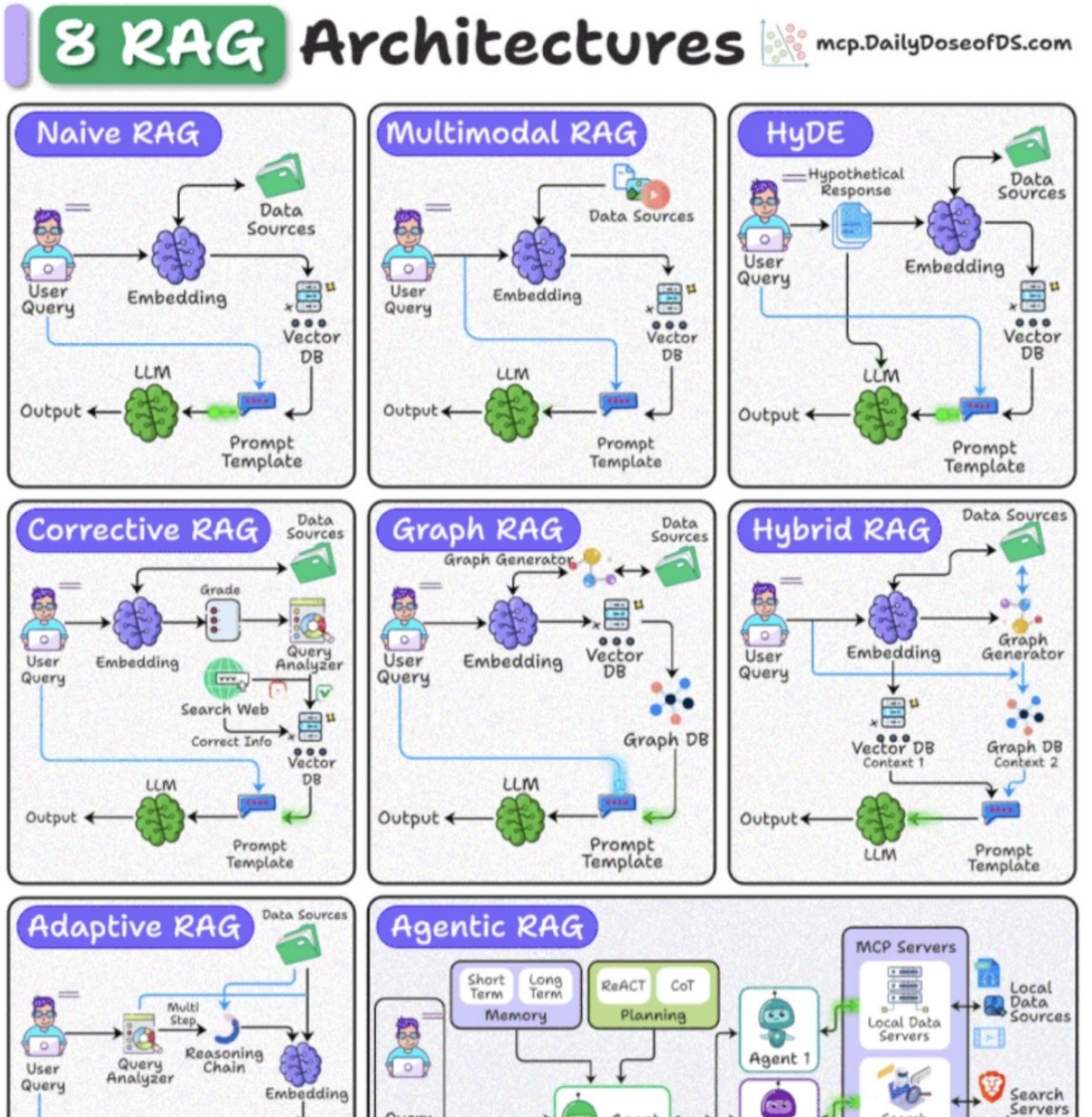
LLM Context is all you need



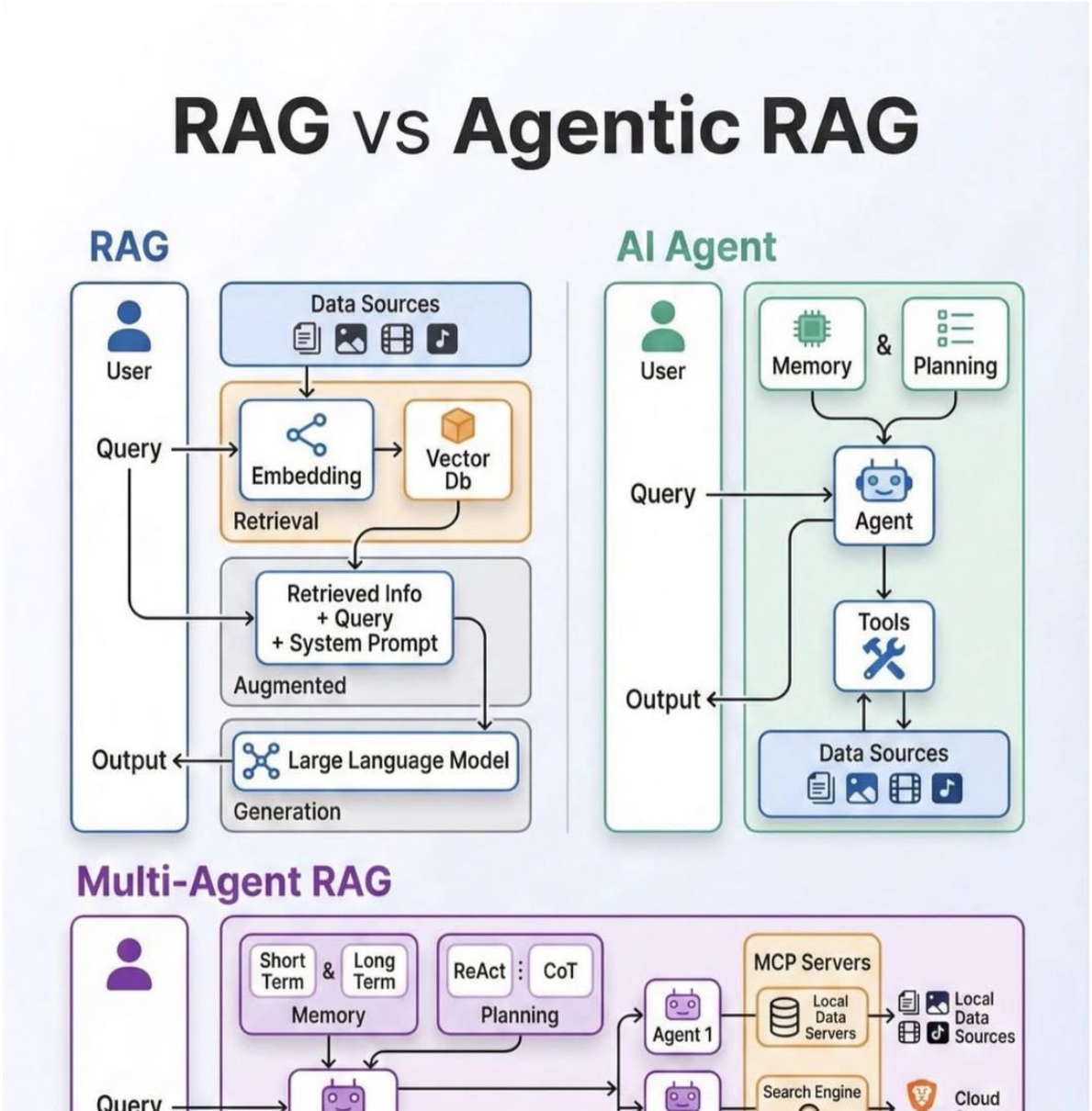
LLM RAG Implementation



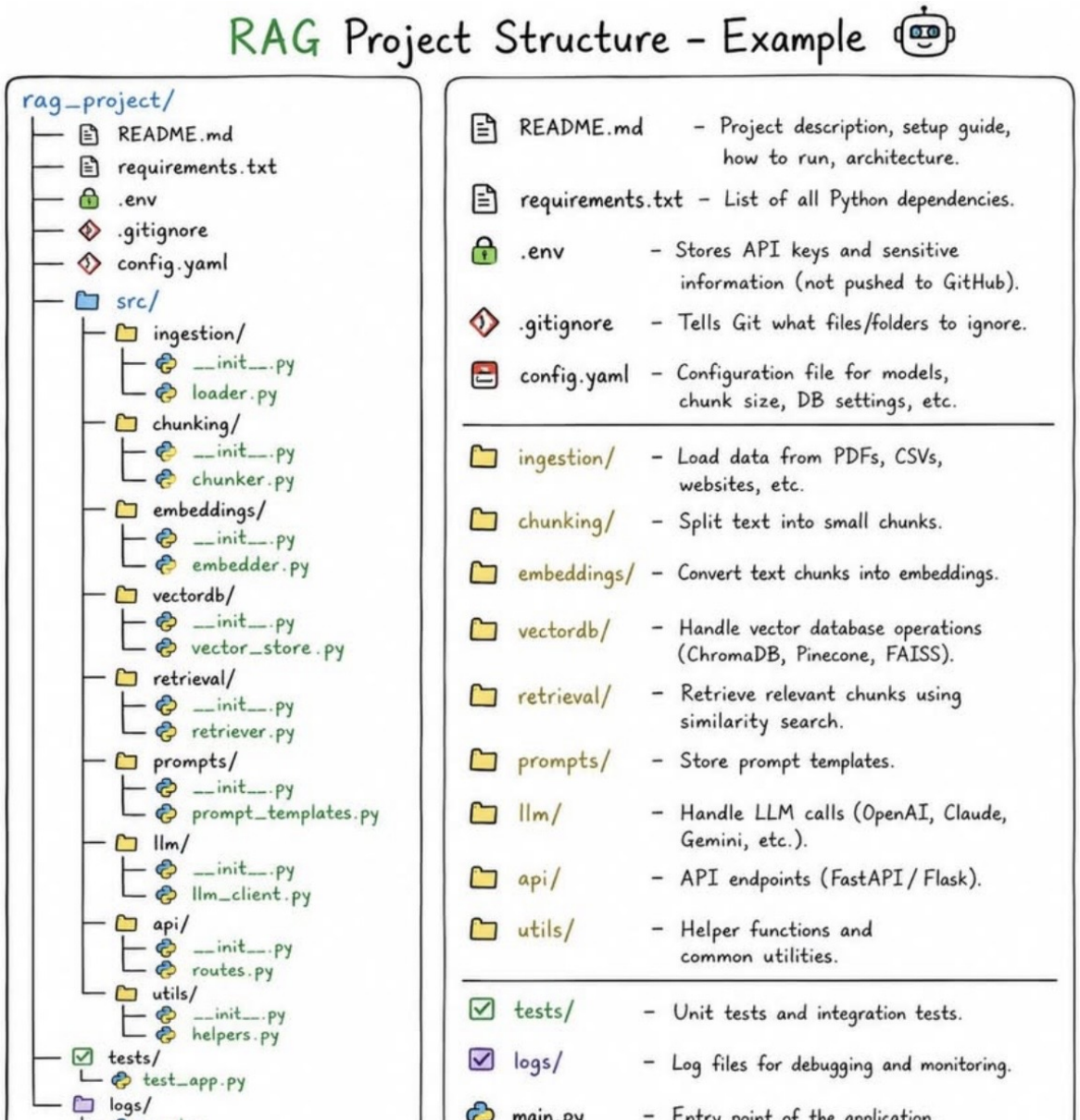
LLM RAG Architectures



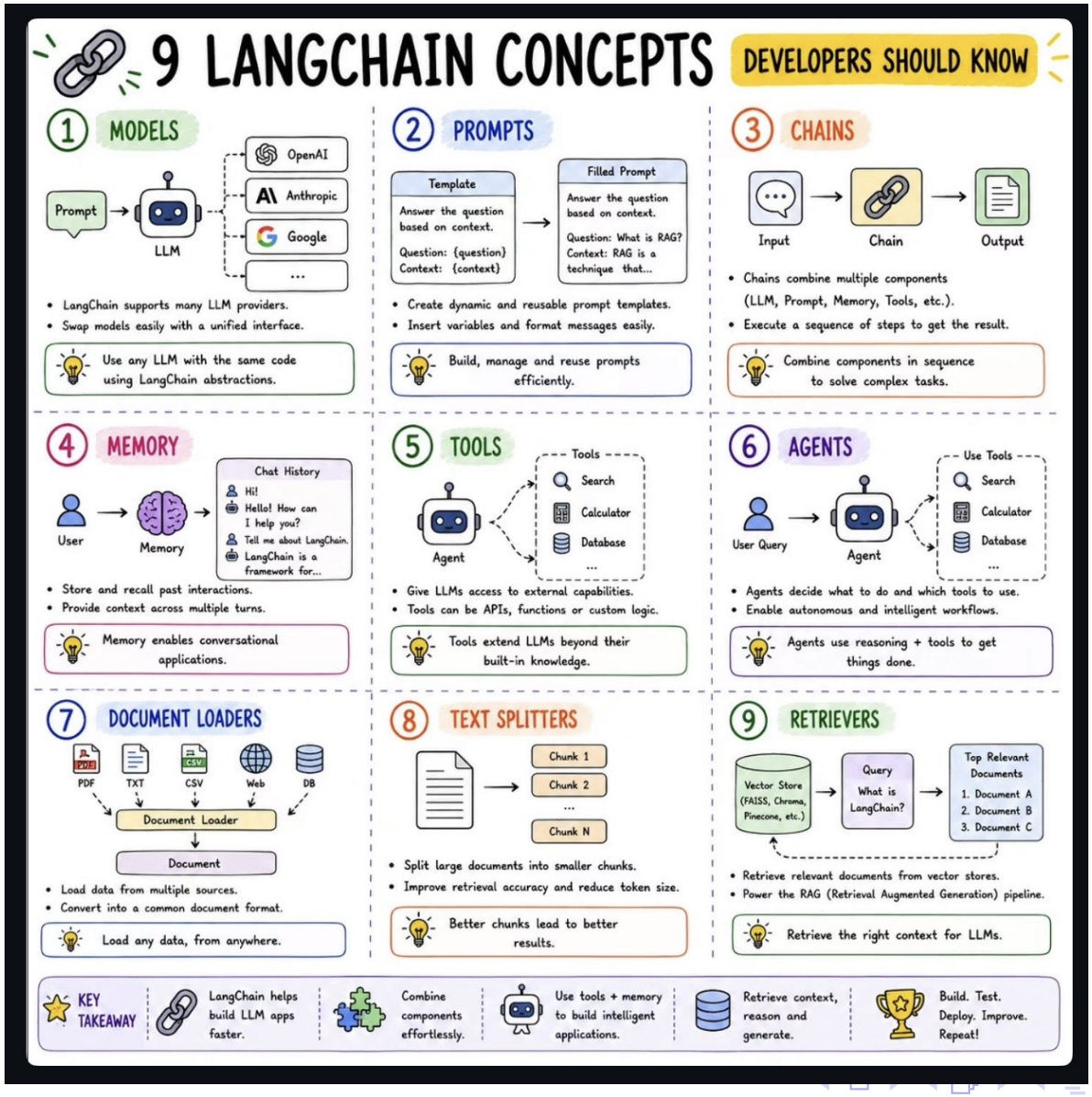
LLM RAG vs Agentic RAG



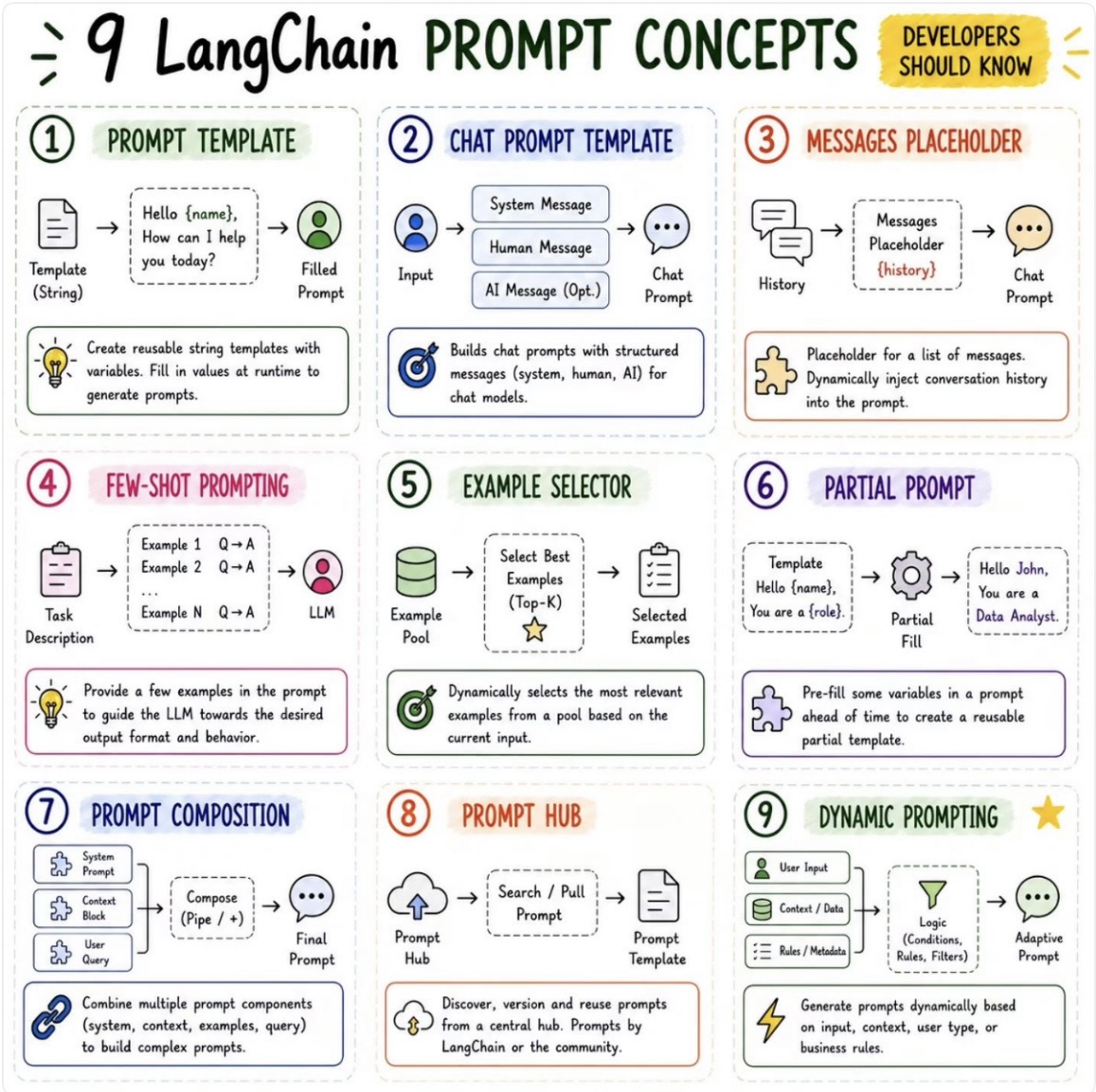
LLM RAG Project Structure



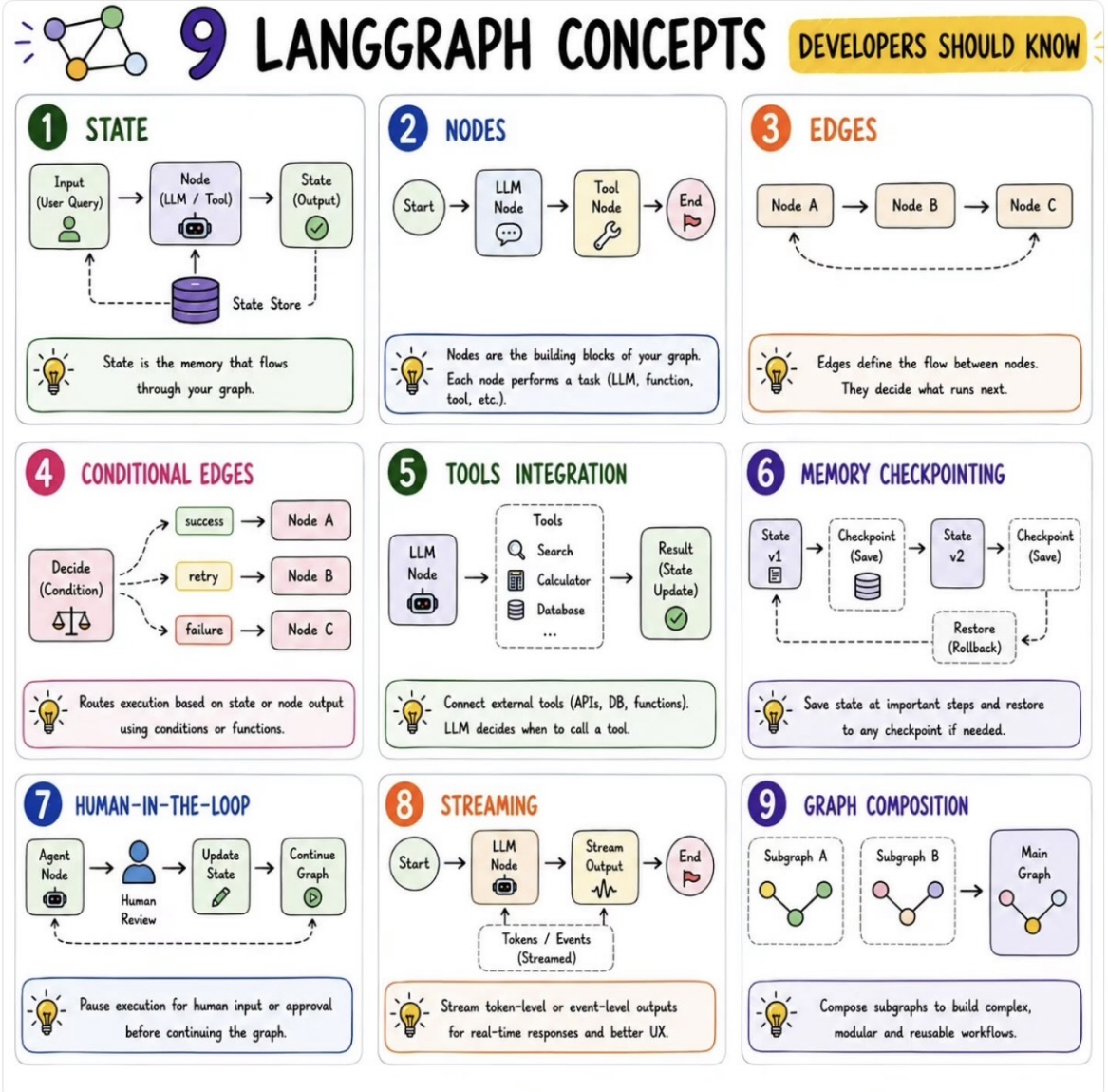
LangChain Concepts



LangChain Prompt Concepts



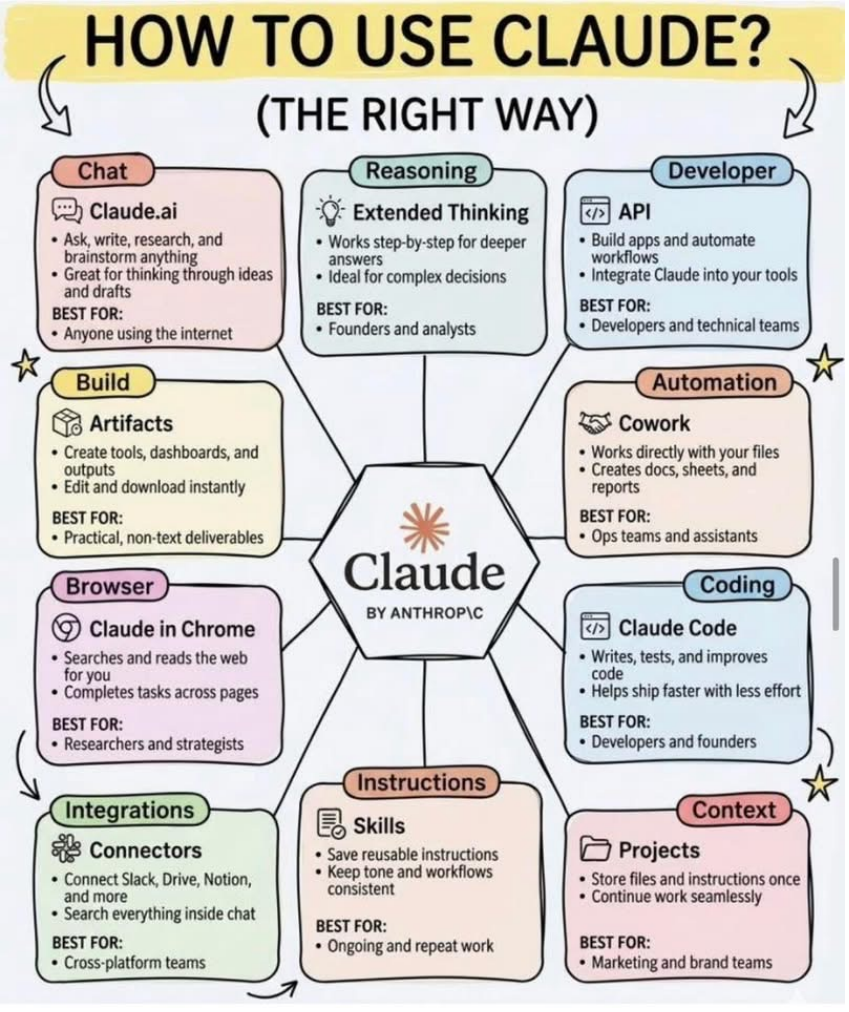
LangGraph Concepts



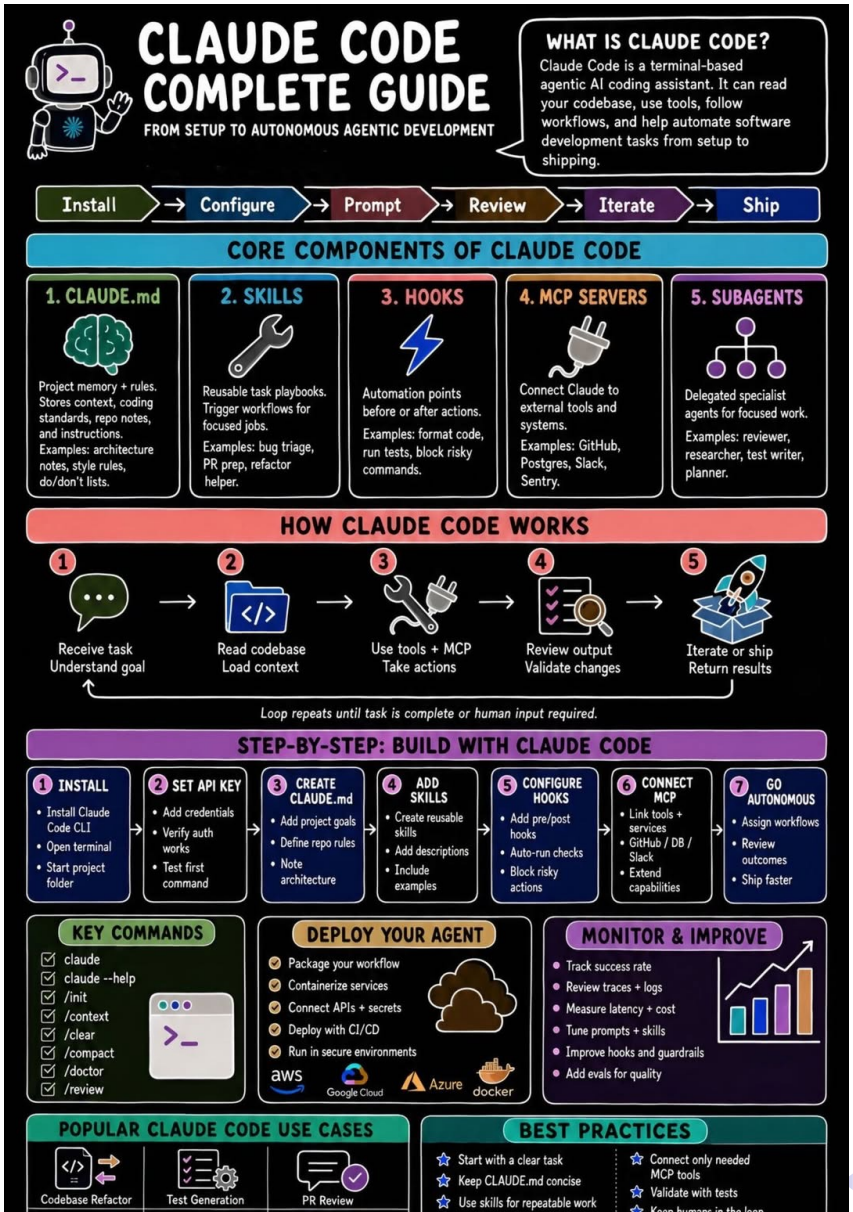
Appendix B

Anthropic Claude

Anthropic Claude



Anthropic Claude



Anthropic Claude

