

Essential AI Terms

25 Foundational AI Terms
Every Tester Should Know



— A Handwritten Guide to Understanding the Language of AI —

≡ Manjunath N P ≡

Artificial Intelligence (AI)

1. SIMPLE DEFINITION



Artificial Intelligence is the broad field of building non-human systems or models that can perform sophisticated tasks such as understanding language, recognizing patterns, making predictions, generating content, or taking actions.

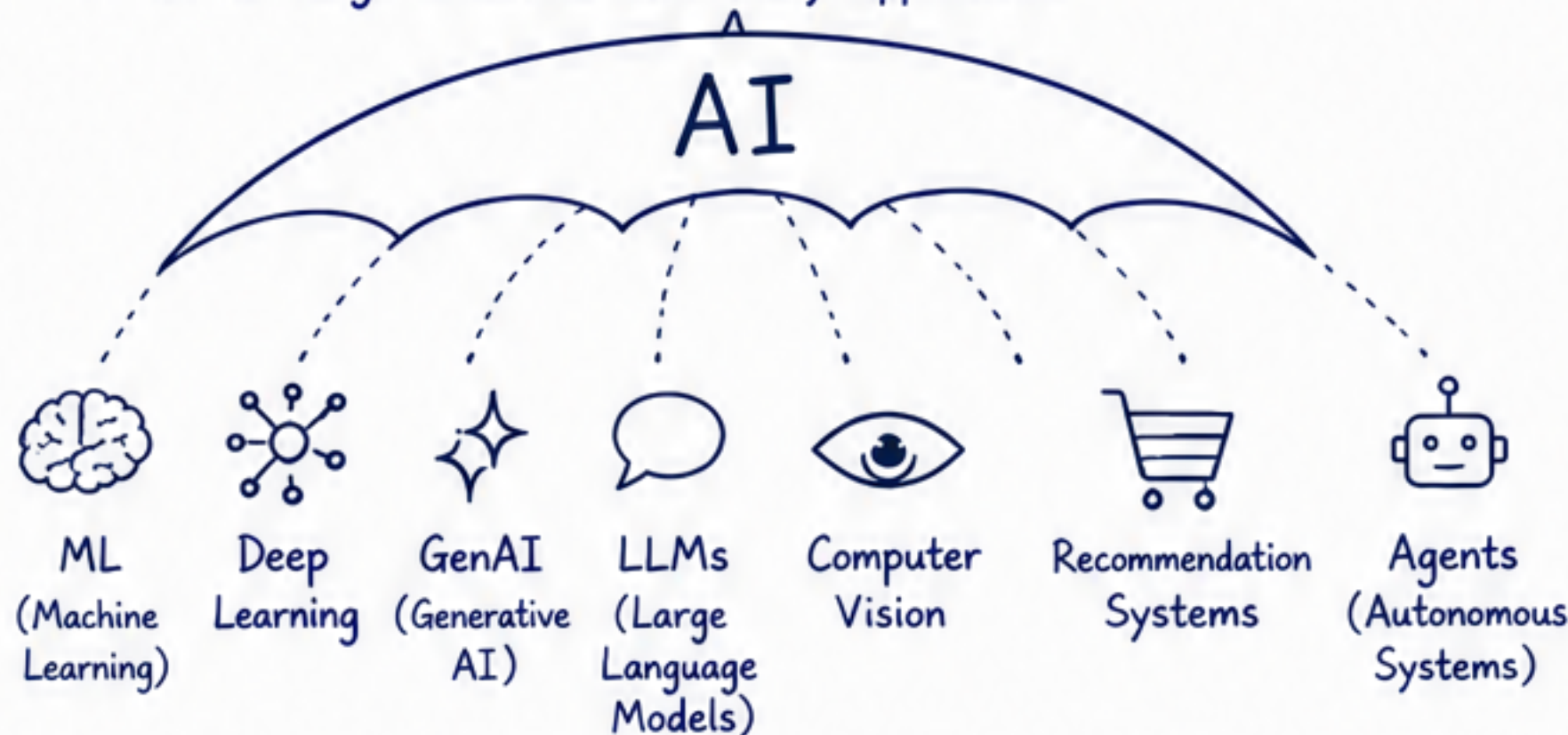
2. WHY IT MATTERS



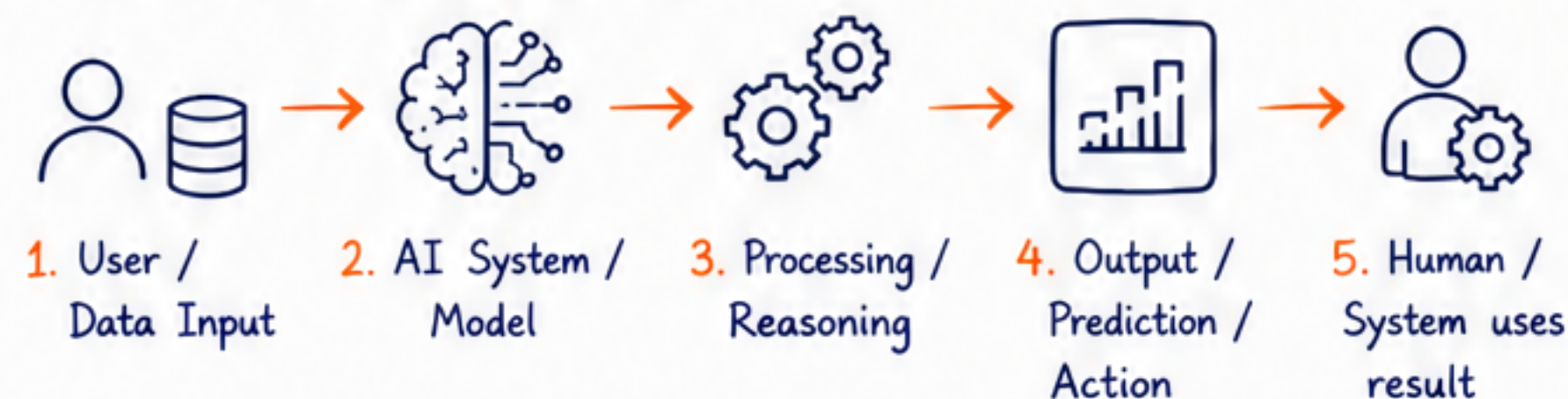
- ✓ AI is the umbrella concept; ML and Generative AI sit inside it.
- ✓ A tester must know the AI capability before choosing test data and quality checks.
- ✓ AI output may be probabilistic or context-dependent, so exact expected-result thinking is not always enough.

3. EASY MENTAL MODEL

AI as a large umbrella over many approaches.



4. FLOW (HIGH-LEVEL)

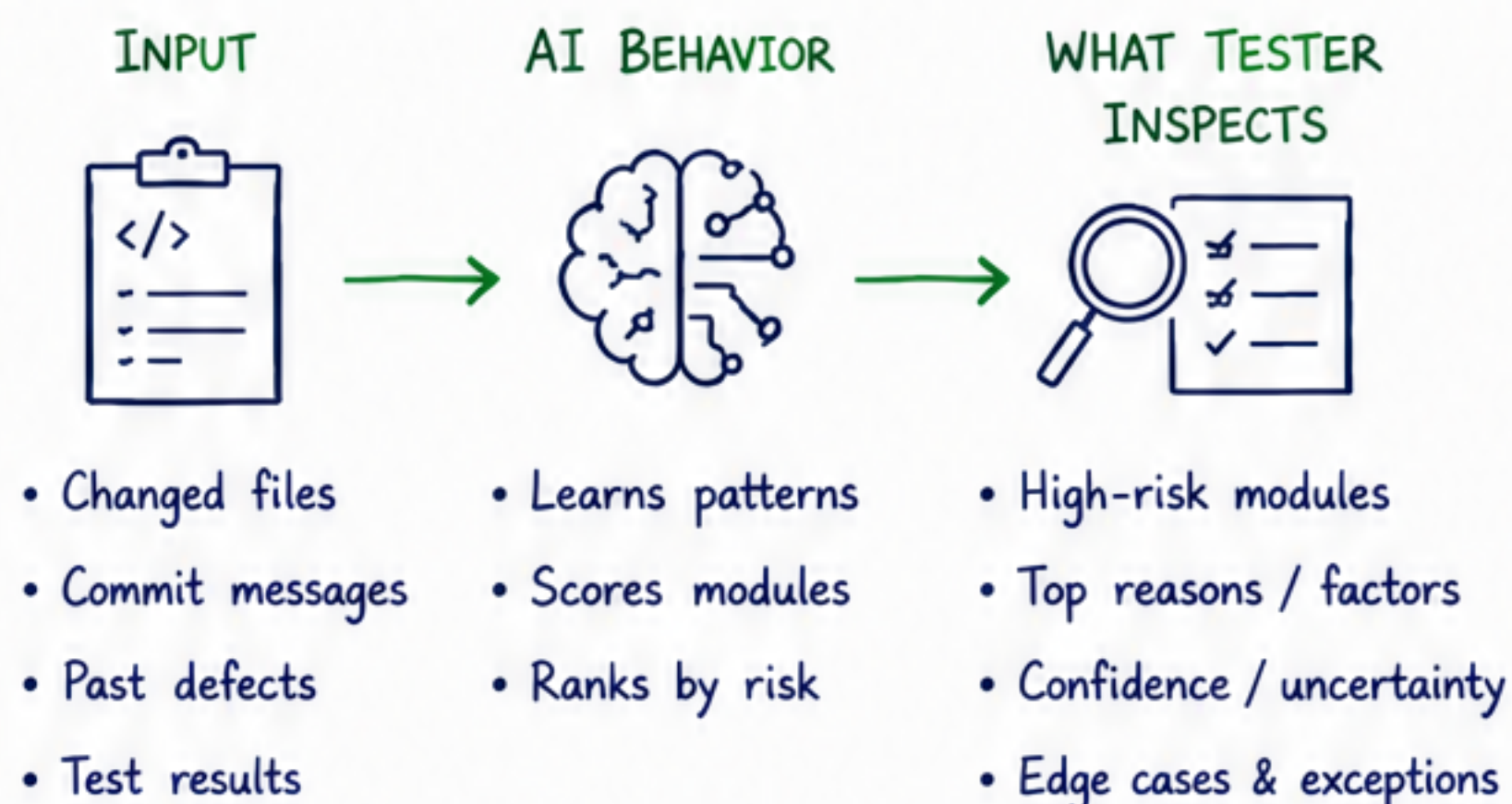


5. CORE IDEAS (AT A GLANCE)

Term	Simple meaning	Tiny example
 AI	Broad field of systems that show intelligent behaviour.	An AI assistant that helps prioritize support tickets.
 ML	AI built by learning patterns from data.	Model learns from past defect data to predict risk.
 GenAI	AI that generates new content.	Model writes test cases or summaries.
 LLM	A generative AI model focused mainly on language.	An LLM answers questions or explains code.

6. PRACTICAL QA EXAMPLE

Defect-risk assistant reads code-change information and predicts which modules are more likely to contain defects.



7. IMPORTANT DISTINCTION

- AI AI is the broad capability.
- ML ML is one way to build AI by learning from data.
- GenAI GenAI generates new content.
- LLM An LLM is a generative AI model focused mainly on language.

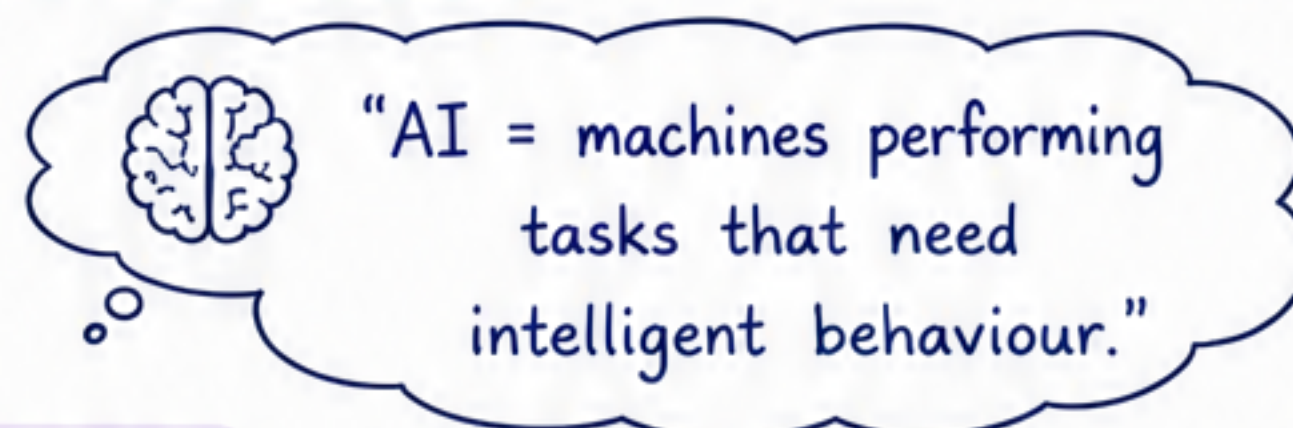
8. TESTER CHECKS (BEFORE & AFTER USING AI)

- ✓ Is the task clearly defined and appropriate for AI?
- ✓ Are inputs relevant, representative, and high quality?
- ✓ Is "good output" clearly defined and measurable?
- ✓ How does the model express uncertainty or low confidence?
- ✓ What are the potential harms, biases, or security risks?

9. COMMON MISTAKES

- ✗ AI = LLM Thinking AI and LLM are the same.
- ✗ Assuming AI output is deterministic (always the same).
- ✗ Assuming fluent output = correct. Fluency is not proof of correctness.
- ✗ Testing only happy paths, ignoring edge cases, uncertainty, and failures.

10. MEMORY LINE



11. NEXT

→ Machine Learning

Machine Learning (ML)

1. DEFINITION

Machine Learning is a way of building systems that learn patterns from data so they can make predictions or decisions on new inputs without being explicitly programmed with every rule.

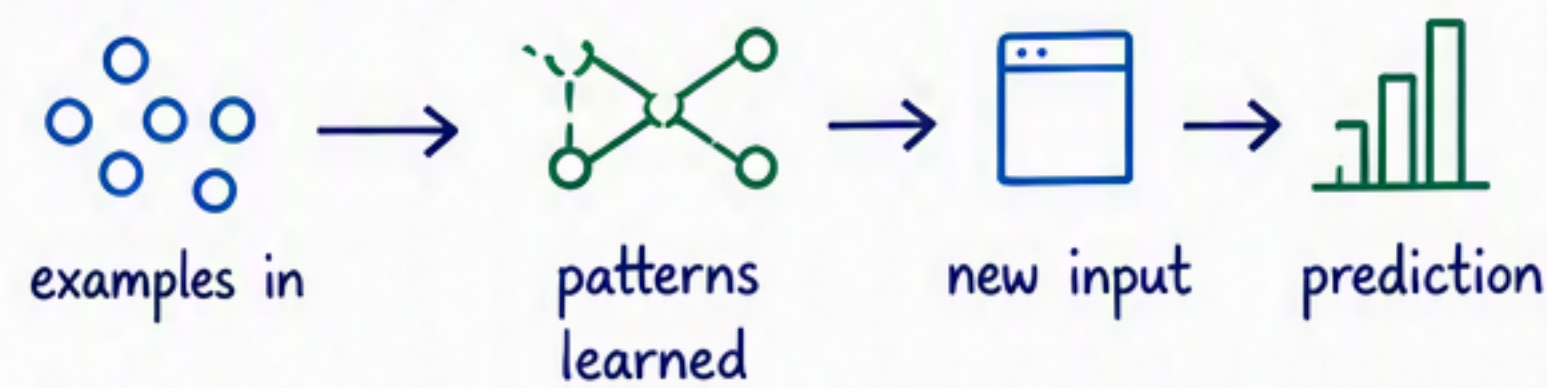


2. WHY IT MATTERS

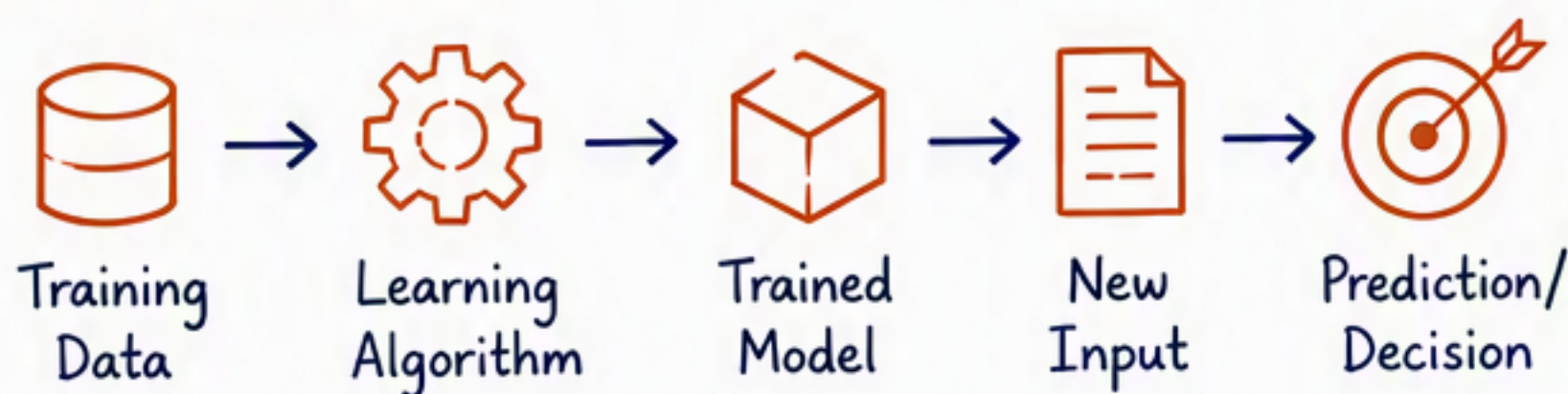
- ✓ Powers prediction
- ✓ Powers classification
- ✓ Powers recommendation
- ✓ Powers anomaly detection

QA must test **unseen data**, **data quality**, and **generalization**.

3. MENTAL MODEL



4. ML FLOW



5. CORE IDEAS

Training Data	Examples the model learns from. Quality and representativeness are crucial.
Feature / Input	Information given to the model to make a prediction.
Label / Target	The correct answer in training data the model tries to predict.
Model	Learned patterns and parameters captured from training data.
Inference	Using the trained model on new data to produce a prediction or decision.

6. QA EXAMPLE



Historical defect data (modules, code churn, complexity, past bugs, tests, etc.) trains a model to **predict which modules are likely to be risky**.

- ✓ Tester checks: **data quality**, **data leakage**, **class imbalance**, and performance on **unseen code**.

7. TRAINING vs INFERENCE

TRAINING	INFERENCE
Learning model parameters from training data. 	Using the trained model on new, unseen data to make predictions.

Training happens offline. Inference happens in use.

8. WHAT TESTERS SHOULD CHECK

- ✓ Representative and sufficient data.
- ✓ Overfitting: model performs well on training data but poorly on unseen data.
- ✓ Unseen cases: test with data the model hasn't seen before.
- ✓ Consistent preprocessing between training and inference.
- ✓ Performance across important groups (e.g., teams, modules, environments).

9. COMMON MISTAKES

- ✗ Testing only on training data.
- ✗ Assuming more data is always better.
- ✗ Ignoring bias in data and model outputs.
- ✗ Treating correlation as causation.



10. MEMORY HOOK



"ML = learn patterns from data, then use them on new data."

Next: Generative AI. →

Generative AI (GenAI)

1. DEFINITION

- Generative AI is AI designed to create new content such as text, images, audio, video, or code based on patterns learned from training data and current input/context.



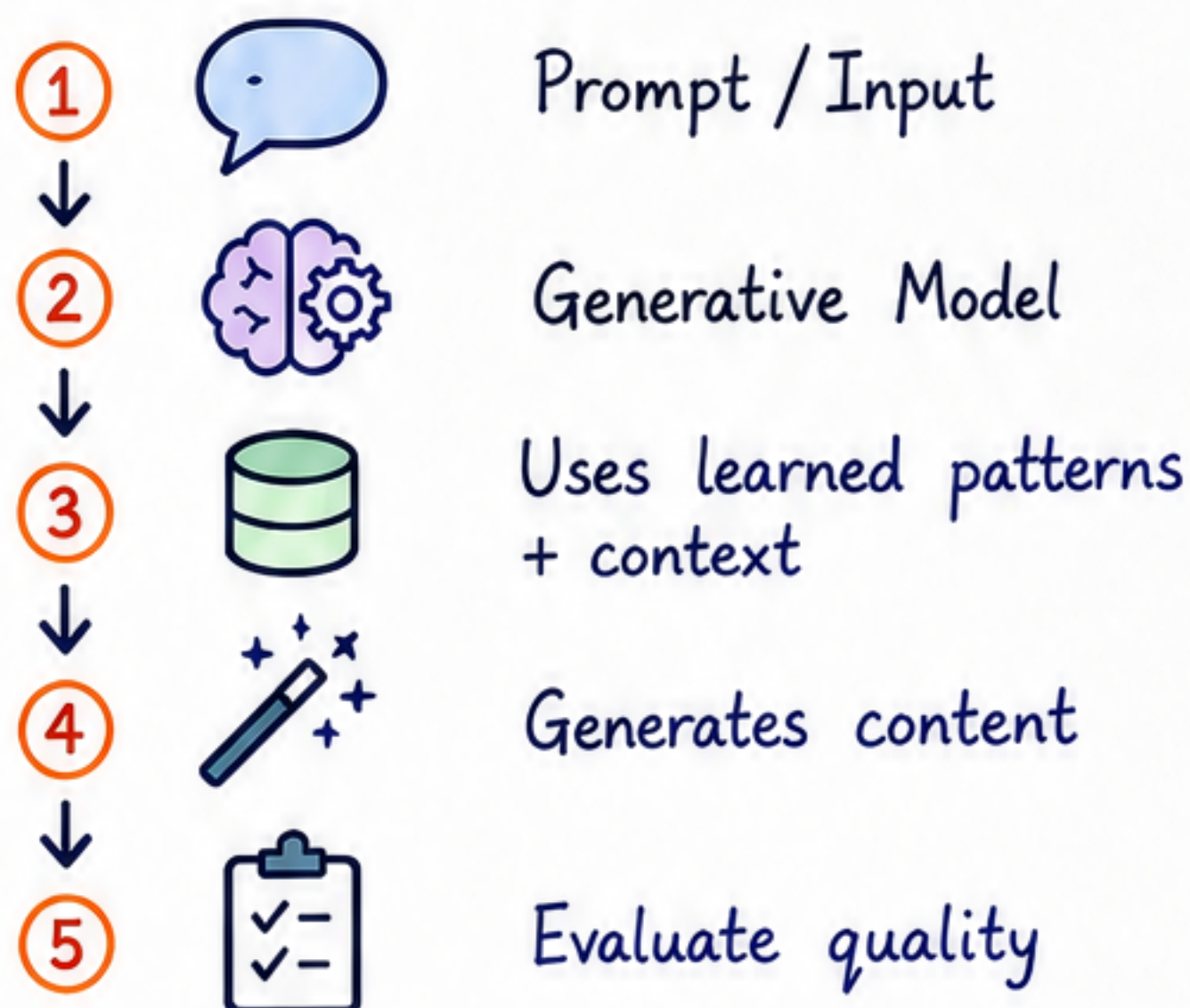
2. WHY IT MATTERS

- ✓ Outputs are open-ended.
- ✓ Can vary across runs.
- ✓ Need multidimensional quality checks rather than only exact matches.

3. MENTAL MODEL



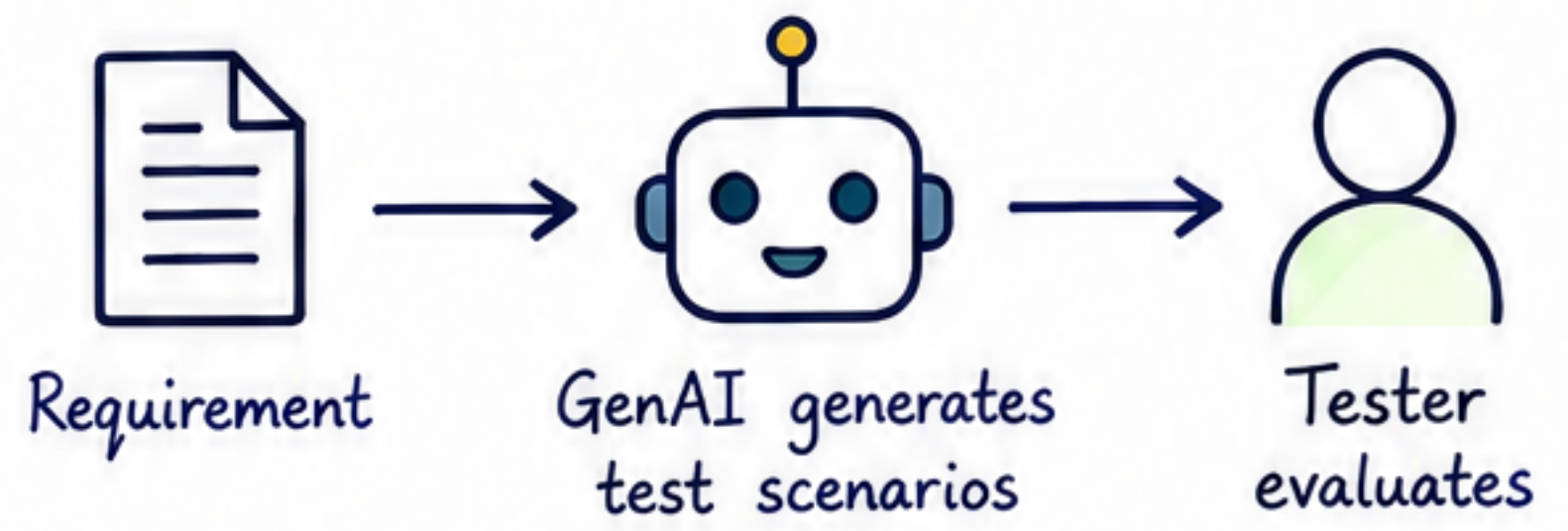
4. BASIC FLOW



5. CORE IDEAS

- Prompt:** The instruction or question you give.
- Context:** Additional information that guides the model.
- Model:** The generative AI system.
- Generated Output:** The new content created by the model.

6. QA EXAMPLE



Tester evaluates correctness, completeness, duplication, relevance, and unsafe assumptions.

7. GENAI vs TRADITIONAL ML

Traditional Predictive ML	Generative AI (GenAI)
Often predicts a label/value.	Creates new content.
Outputs are usually closed-set.	Outputs are open-ended and diverse.
Typically evaluated with accuracy, precision, etc.	Needs multidimensional quality evaluation.

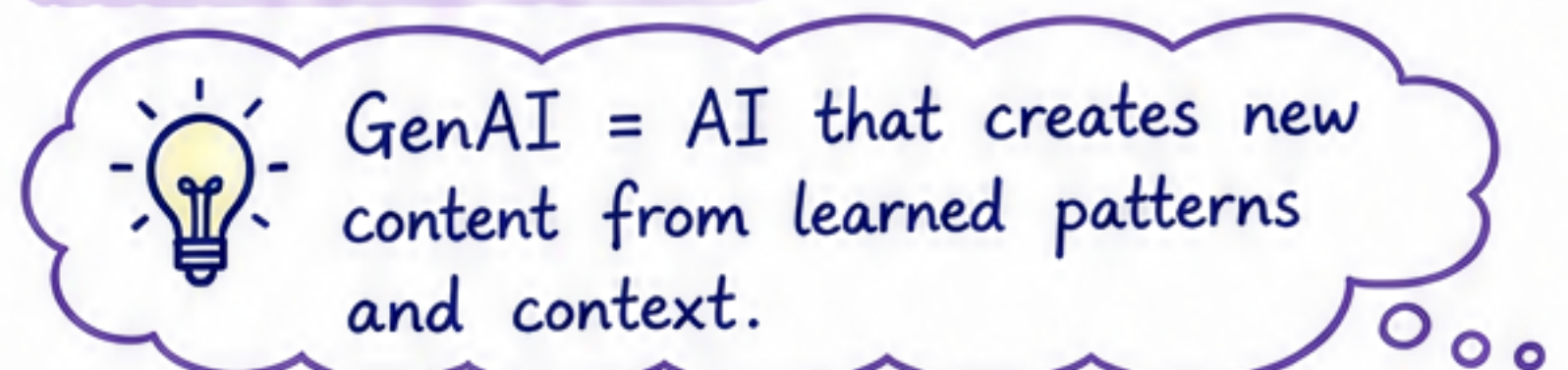
8. WHAT A TESTER SHOULD CHECK

- ✓ **Instruction Following:** Did the model follow the instructions?
- ✓ **Factuality:** Is the information factually correct?
- ✓ **Relevance:** Is the output relevant to the context?
- ✓ **Safety:** Is the output safe and non-harmful?
- ✓ **Consistency:** Are outputs reasonably consistent across multiple runs?

9. COMMON MISTAKES

- ✗ Treating fluent output as correct.
- ✗ Expecting identical responses.
- ✗ Evaluating with one metric only.
- ✗ Ignoring prompt changes.
- ✗ Not checking safety and assumptions.

10. MEMORY HOOK





Model



1. DEFINITION

A model is the learned computational component that maps inputs to outputs using parameters learned during training or configured behavior.



2. WHY IT MATTERS

In AI systems, the model is only one part of the complete application.

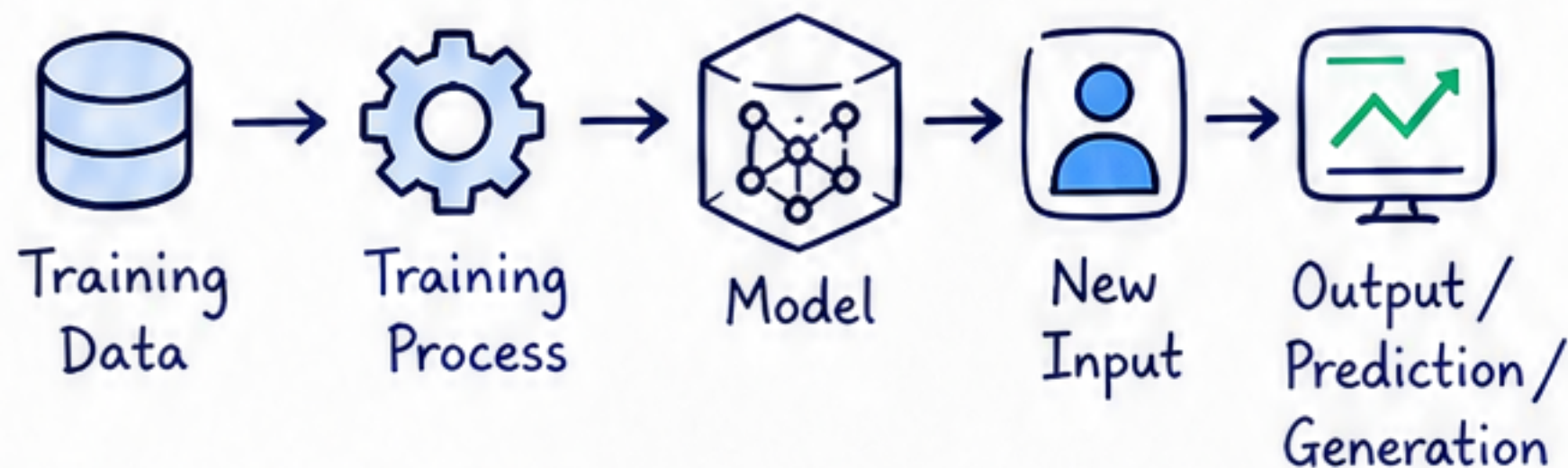


3. MENTAL MODEL

Input goes into the learned engine, output comes out.



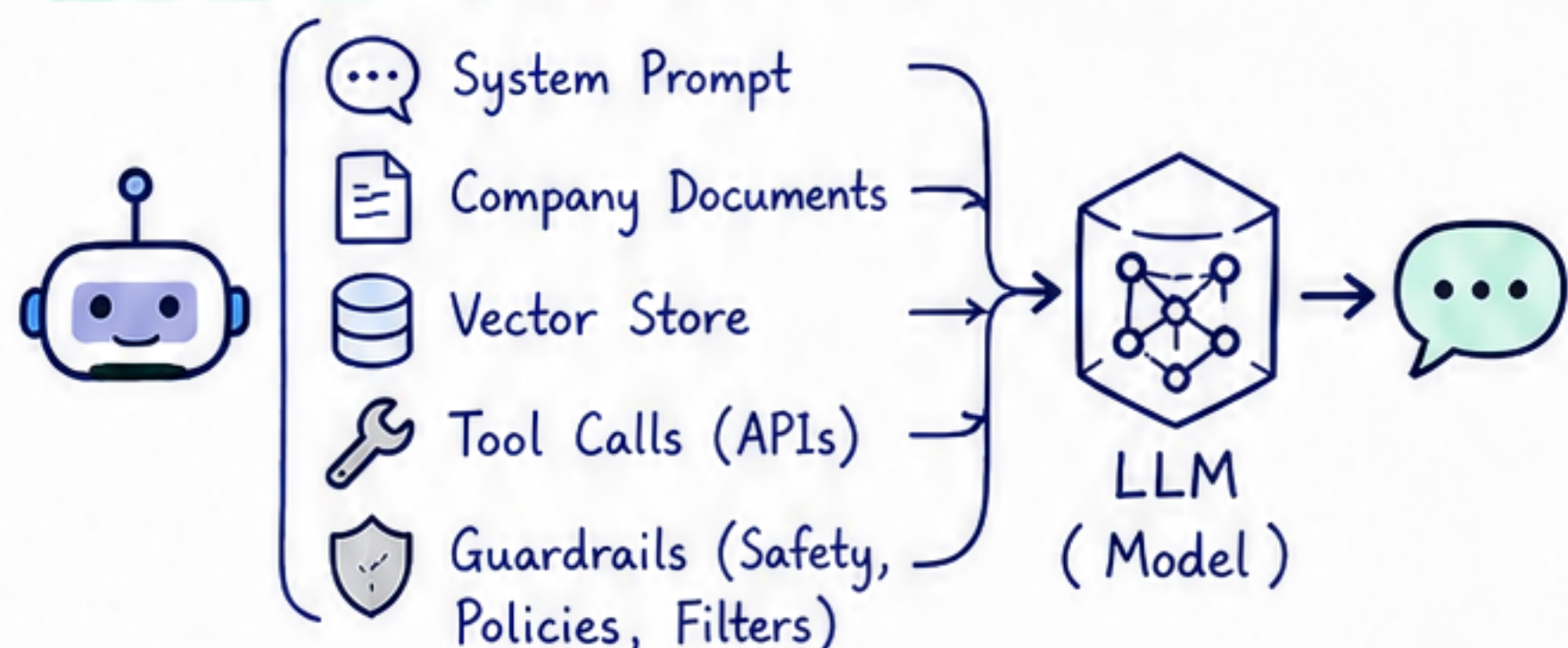
4. FLOW (HIGH LEVEL)



5. CORE IDEAS

- ✓ Input: data, prompt, query, context.
- ✓ Learned parameters: weights, embeddings, rules, probabilities, etc.
- ✓ Inference: the model uses learned knowledge to produce a result.
- ✓ Output: prediction, classification, summary, answer, code, image, etc.

6. QA EXAMPLE (CHATBOT)



Testing only the model misses the system.

7. DISTINCTION

MODEL



Learned engine that maps input to output.

AI APPLICATION



The application includes many components around the model.

Application may:

- Transform inputs
- Enrich with data
- Filter or guard
- Combine model output with other sources
- Reject or override output

8. WHAT A TESTER CHECKS

- ✓ Which model is used (name, version, provider).
- ✓ Exact inputs given to the model.
- ✓ Preprocessing and prompt/template used.
- ✓ Post-processing, formatting, filters.
- ✓ Impact of model version or parameter changes.

9. COMMON MISTAKES

- ✗ Calling the entire app "the model".
- ✗ Blaming the model for retrieval, tool, or integration failures.
- ✗ Ignoring configuration or prompt/template changes.
- ✗ Assuming one model fits every task.

10. MEMORY HOOK

Model = learned engine that maps input to output.

11. NEXT



Large Language Model. ----->

12. ACCURACY



Separate model behavior from application behavior.



Large Language Model (LLM)

1. DEFINITION



A Large Language Model is a large pre-trained language model that processes and generates language, typically by predicting tokens based on available context.

2. WHY IT MATTERS



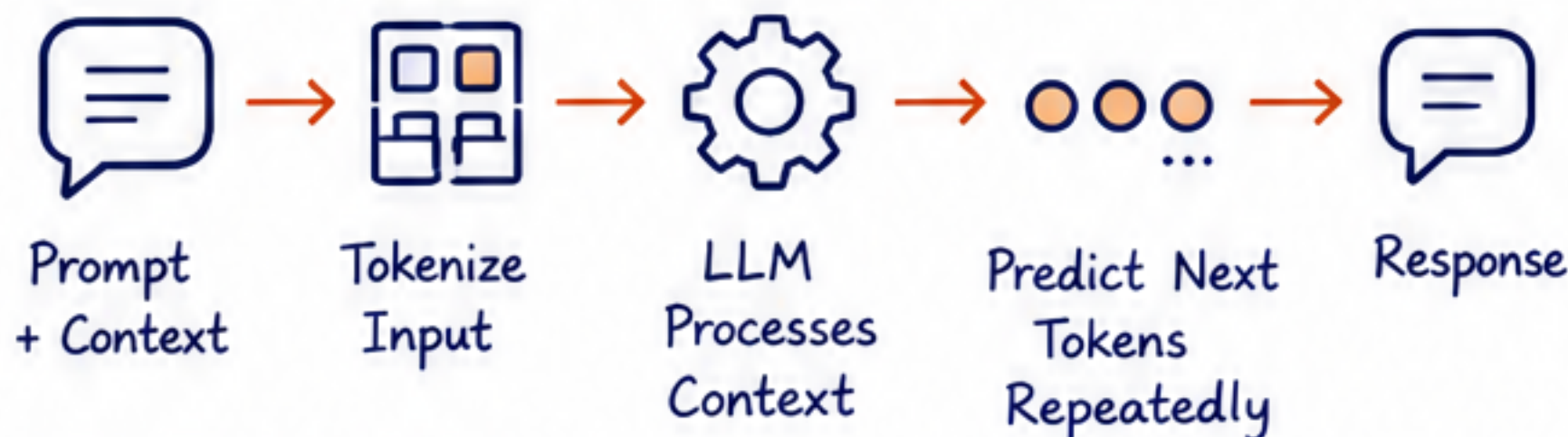
LLMs power chat, summarization, code/test generation, RAG, and agents; outputs are probabilistic and can be fluent but wrong.

3. MENTAL MODEL



A powerful pattern-based language engine, not a guaranteed-facts database.

4. HOW IT WORKS (HIGH-LEVEL FLOW)



5. CORE IDEAS

- Prompt** : Your instruction or question.
- Token** : Smallest unit the model uses. One token is not one word.
- Context** : All input the model can use (prompt, history, retrieved info, etc.).
- Model** : The pre-trained engine that finds patterns in data.
- Output** : Generated token sequence (the response).

6. QA EXAMPLE (TEST AUTOMATION CONTEXT)



Requirement: Users shall log in with valid email and password. Account locks after 5 failed attempts.

Prompt : Generate negative test scenarios for this login requirement.

As a Tester, I will:

- ✓ Verify completeness (coverage of rules/edges)
- ✓ Check correctness (expected results)
- ✓ Look for duplication and overlaps
- ✓ Identify assumptions and missing details

7. LLM ≠ CHATBOT

LLM



The core model that predicts and generates tokens from context.

CHATBOT (EXAMPLE)



A system that may include

- LLM
- State / Memory
- Retrieval (RAG)
- Tools / APIs
- Guardrails / Safety
- User Interface

Key point: Not every application using an LLM is a chatbot.

8. WHAT A TESTER SHOULD CHECK

- ✓ **Relevance**: Is the response relevant to the prompt and context?
- ✓ **Instruction Following**: Did it follow all instructions and constraints?
- ✓ **Hallucination**: Are there unsupported or made-up claims?
- ✓ **Ambiguity Handling**: How does it handle unclear or ambiguous prompts?
- ✓ **Multi-run Behavior**: Is the output consistent (or acceptably variable) across multiple runs?

9. COMMON MISTAKES

- ✗ Thinking LLM searches the internet by default.
- ✗ Assuming LLM stores every conversation permanently.
- ✗ Believing fluent output proves understanding or correctness.
- ✗ Relying on one-run testing and calling it done.

10. IMPORTANT NOTES

- LLMs do not "know" facts. They predict the most likely next tokens from patterns.
- Accuracy depends on model, data, context, prompt quality, and constraints.
- Outputs can be incorrect or outdated. Always verify critical information.

11. MEMORY (ONE-LINER)

LLM = language model that predicts and generates token sequences from context.



Next: Prompt.



Prompt



1. DEFINITION



A prompt is input supplied to a generative model to influence what it should do and how it should respond; it can contain a question, instruction, context, examples, constraints, or requested format.

2. WHY IT MATTERS



- ✓ Prompt quality affects behavior.
- ✓ Wording changes can reveal ambiguity, instruction-following, and safety issues.
- ✓ Prompts should be versioned and regression-tested.

3. MENTAL MODEL



Test step + requirement + test data combined into natural-language input.

4. FLOW (HIGH LEVEL)



5. CORE IDEAS IN A PROMPT

- ✓ Instruction – what to do
- ✓ Context – background information
- ✓ Constraints – rules, limits, scope
- ✓ Examples – few-shot or reference
- ✓ Output format – how to respond

6. QA EXAMPLE

WEAK PROMPT (LESS USEFUL)



Give test cases.

VS

STRONG PROMPT (MORE USEFUL)



You are a QA engineer. For the “Password Reset” feature of a web application, create test cases.

Include positive, negative, boundary, and security scenarios. Consider email not received, expired link, token reuse, weak password, rate limiting, etc.

Provide output in a table with columns:
Test Case ID | Scenario | Test Data | Steps | Expected Result | Type.

7. PROMPT ≠ MODEL (IMPORTANT DISTINCTION)



A prompt is not the model. Better prompts improve odds but cannot guarantee missing knowledge or eliminate hallucination.

8. WHAT A QA/TESTER SHOULD CHECK

- ✓ Clarity of intent and expected output
- ✓ Explicit constraints and assumptions
- ✓ Sufficient context and test data
- ✓ Required output format is specified
- ✓ Resistance to conflicting instructions

9. COMMON MISTAKES

- ✗ Vague or unclear instructions
- ✗ Mixing multiple goals in one prompt
- ✗ Missing business or domain context
- ✗ Changing prompt without regression evaluation

10. MEMORY HOOK



Prompt = instructions + context + constraints given to the model.

Token

1. Definition

A token is a unit of text used by a language model during processing; a token can be a whole word, part of a word, punctuation, or another text fragment depending on the tokenizer.



2. Why it matters

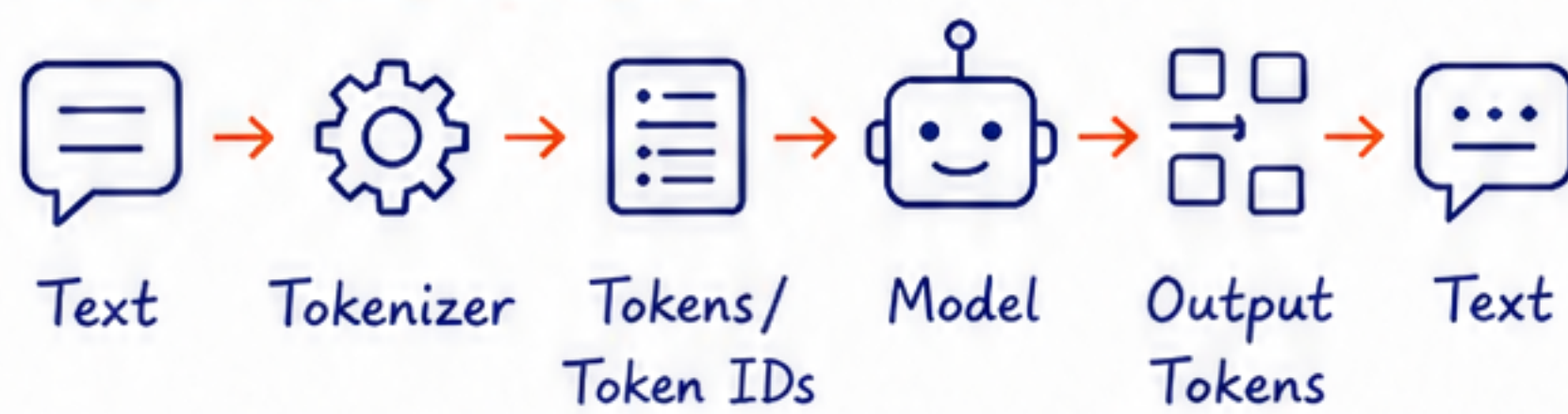
- ✓ Token counts affect context limits, latency, and usage cost.
- ✓ Long prompts and outputs consume more tokens.
- ✓ Staying within limits ensures complete responses.

3. Mental model

Text is broken into puzzle pieces before the model processes it.



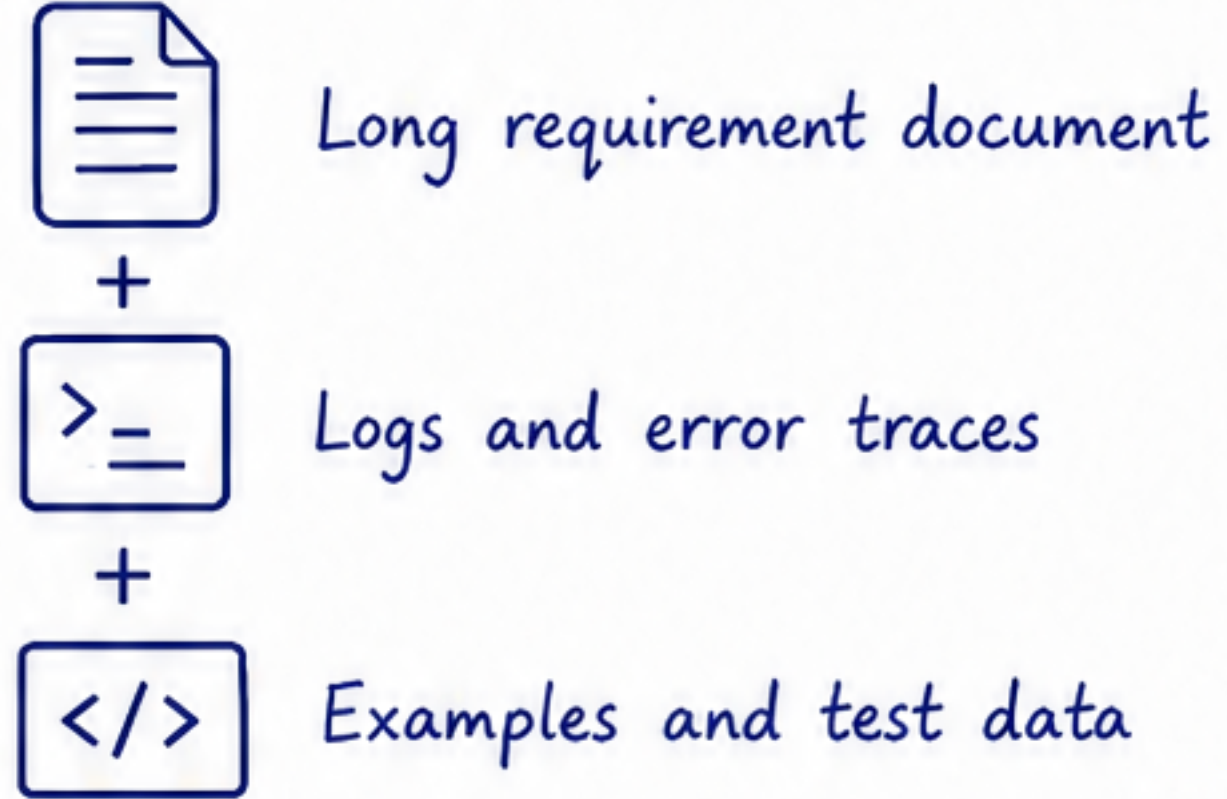
4. Flow (High level)



5. Core ideas

- Token: The basic unit of text for a model.
- Token ID: The numeric ID assigned to a token.
- Input tokens: Tokens in your prompt, instructions, files, examples, etc.
- Output tokens: Tokens generated by the model in its response.
- Tokenizer: The rules/algorithm that splits text into tokens.

6. QA Example



All of the above are input tokens.
More input tokens → fewer tokens left for the model's response.



7. Distinction

- Token $\neq$ Word Tokens may be shorter or longer than words.
- Tokenization $\neq$ Fixed It varies by tokenizer/model.

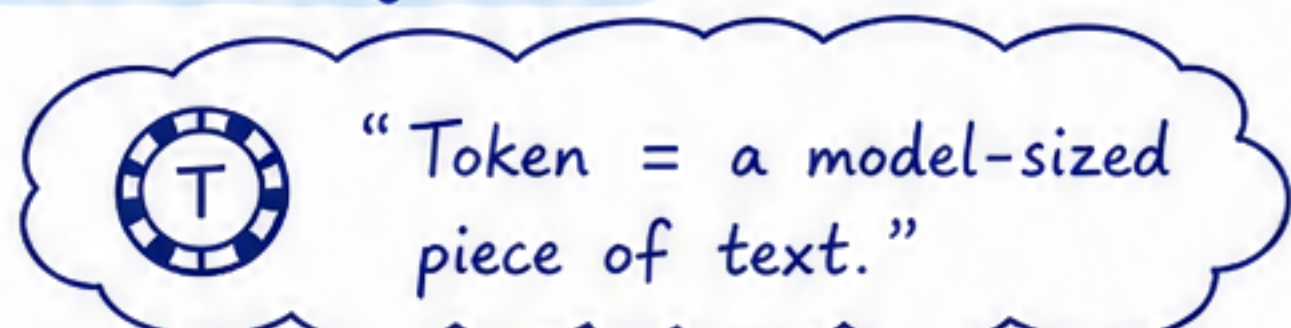
8. Tester Checks

- ✓ Are prompts token-heavy (verbose, repetitive)?
- ✓ Are inputs (docs, logs, attachments) very long?
- ✓ Is truncation happening? Any content dropped?
- ✓ Is output length appropriate for the task?
- ✓ How does the model behave near context limits?

9. Common Mistakes

- ✗ Assuming one token equals one word.
- ✗ Counting characters as tokens.
- ✗ Ignoring hidden/system content (instructions, tool messages, metadata).
- ✗ Ignoring output tokens in capacity planning.

10. Memory Hook



→ Next: Tokenization

Tokenization

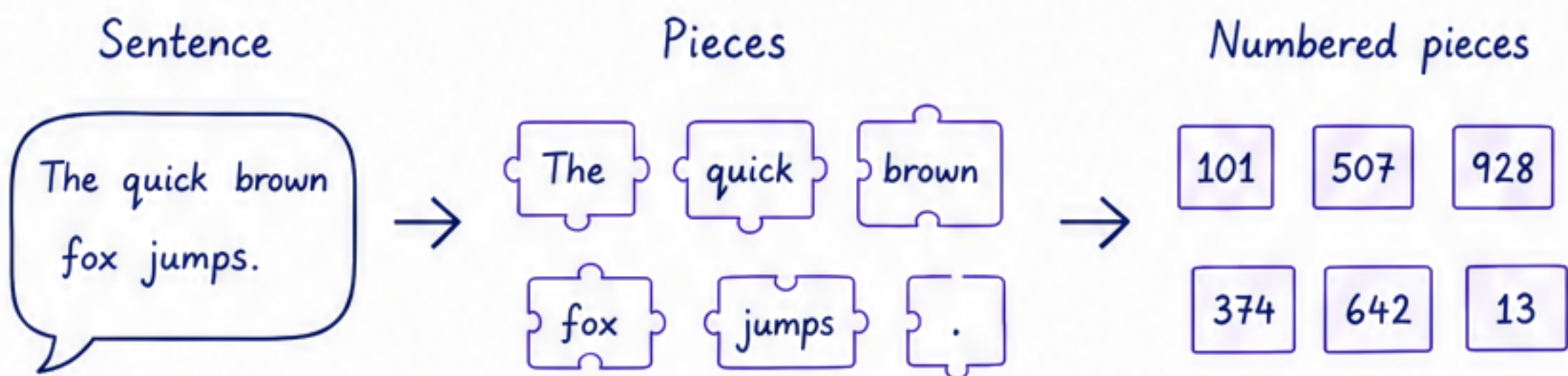
1. Definition

Tokenization is the process of converting text into tokens and usually token IDs that a language model can process.

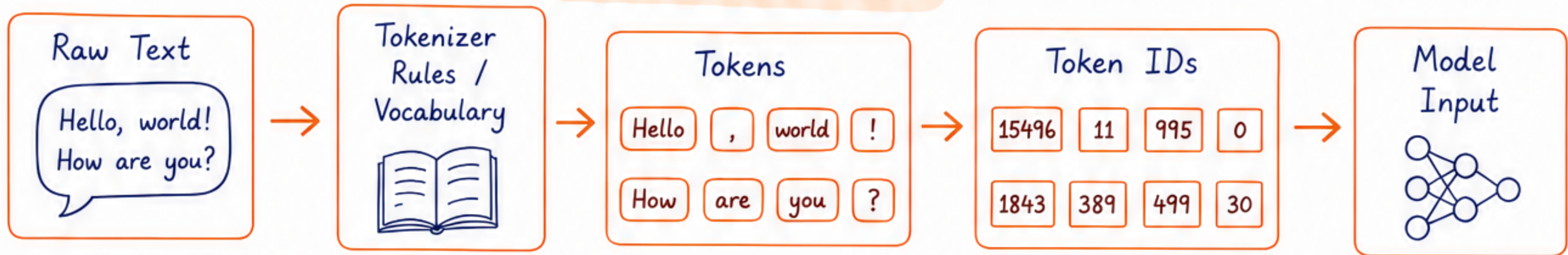
2. Why it matters

- ★ Different text can split into different numbers of tokens, affecting limits and cost.
- ★ Unusual strings, code, emojis, languages, and punctuation may tokenize differently.

3. Mental model



4. Tokenization Flow



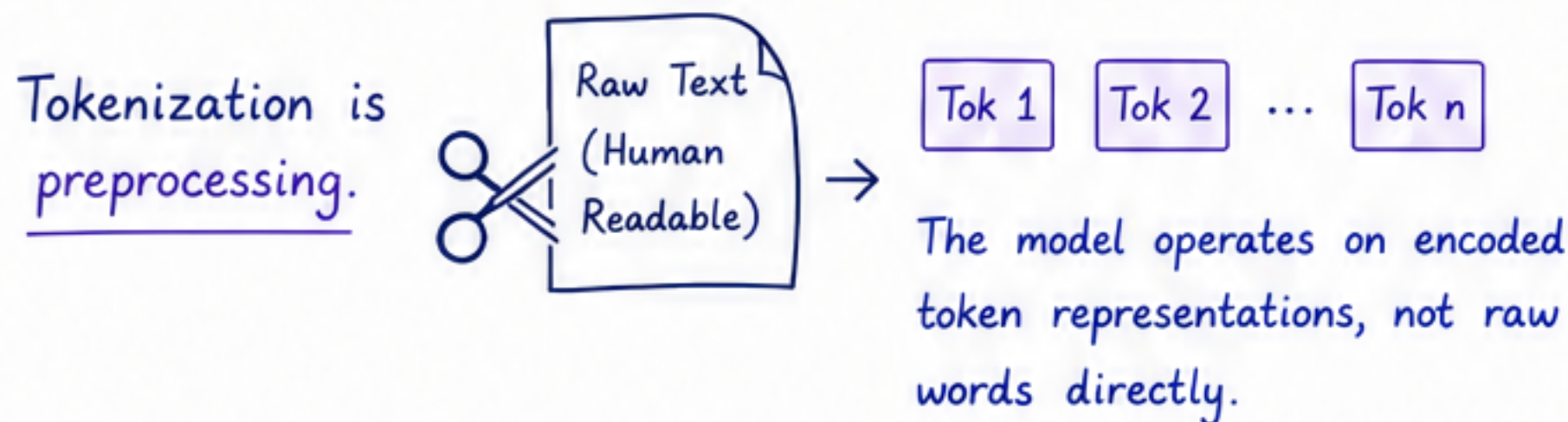
5. Core Ideas

Concept	What it means
Raw Text	The original string you provide.
Vocabulary	The set of tokens (subwords, words, symbols) the tokenizer knows.
Token	A piece of text (could be a word, part of a word, or even a single character).
Token ID	The integer ID assigned to a token from the vocabulary.
Decode	Converting token IDs back to text (may not be perfectly reversible).

6. QA Example

- ✓ API payloads with long JSON keys or values may consume unexpectedly many tokens.
- ✓ Stack traces with paths and line numbers can explode token count.
- ✓ UUIDs (e.g., 550e8400-e29b-41d4-a716-446655440000) often split into many tokens.
- ✓ Code snippets: indentation, symbols, and snakeCase may tokenize inefficiently.
- ✓ Multilingual text (e.g., Chinese, Arabic) may use more tokens than expected.

7. Key Distinction



8. Tester Checks

- Test with multilingual text and different scripts.
- Include emojis and special symbols.
- Use long identifiers, GUIDs, and hashes.
- Check code snippets and logs.
- Verify behavior near context limits (truncation, errors, or silent drops).

9. Common Mistakes

- ✗ Assuming spaces define tokens.
- ✗ Assuming all models tokenize identically.
- ✗ Ignoring special tokens (BOS, EOS, system tokens, etc.).
- ✗ Hard-coding token estimates without verifying.

10. Memory Hook



Context Window

1. DEFINITION

The context window is the limited amount of tokenized information a model can consider in a single interaction or inference request, including relevant instructions, conversation, retrieved context, and output budget depending on the system.



2. WHY IT MATTERS

- Long conversations or documents can exceed the limit.
- Important details may get pushed out of the window.
- Effective attention on key information can be reduced.



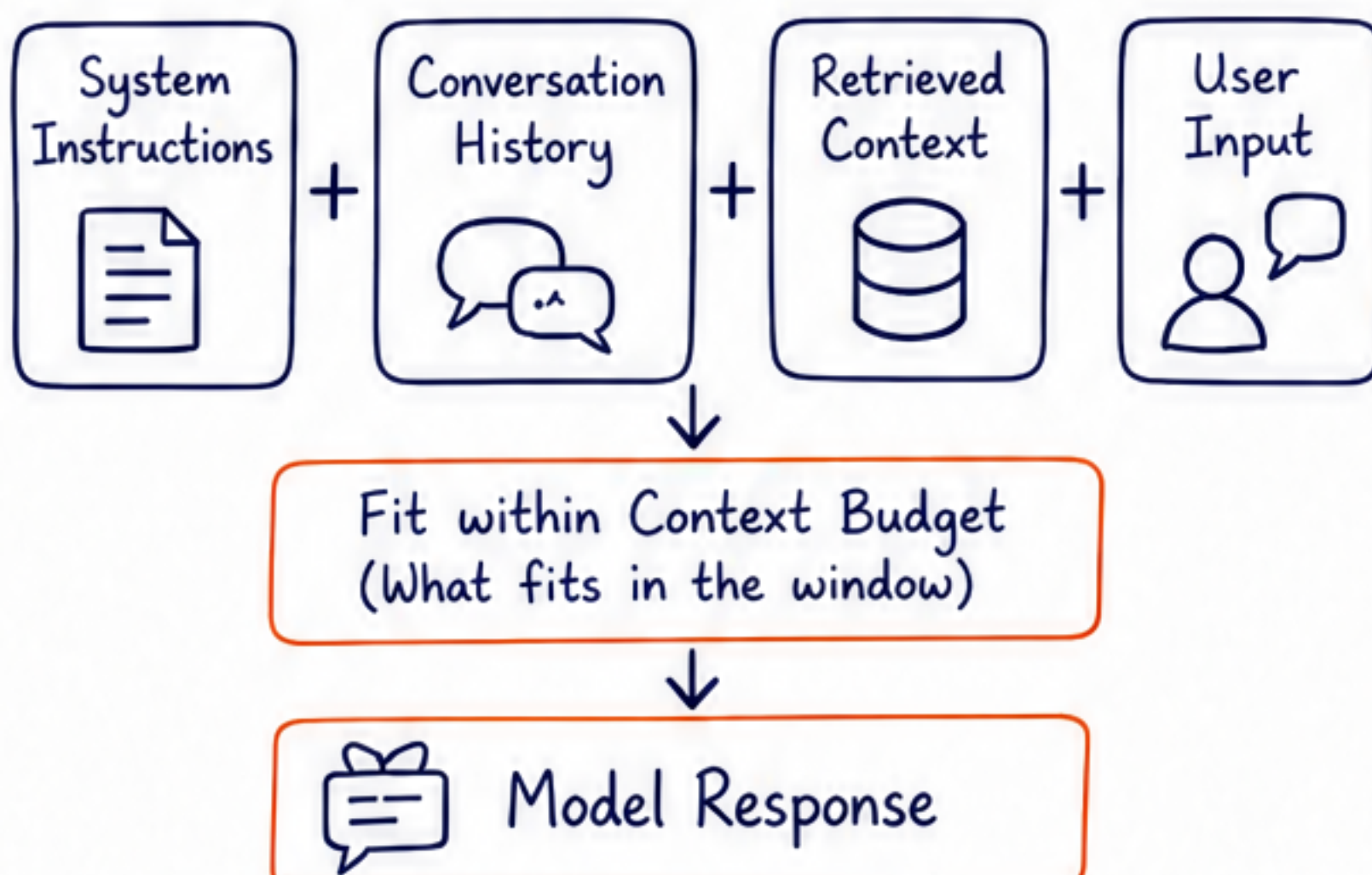
3. MENTAL MODEL

Think of the context window as your desk with limited working space.



You can only work with what fits on the desk right now.

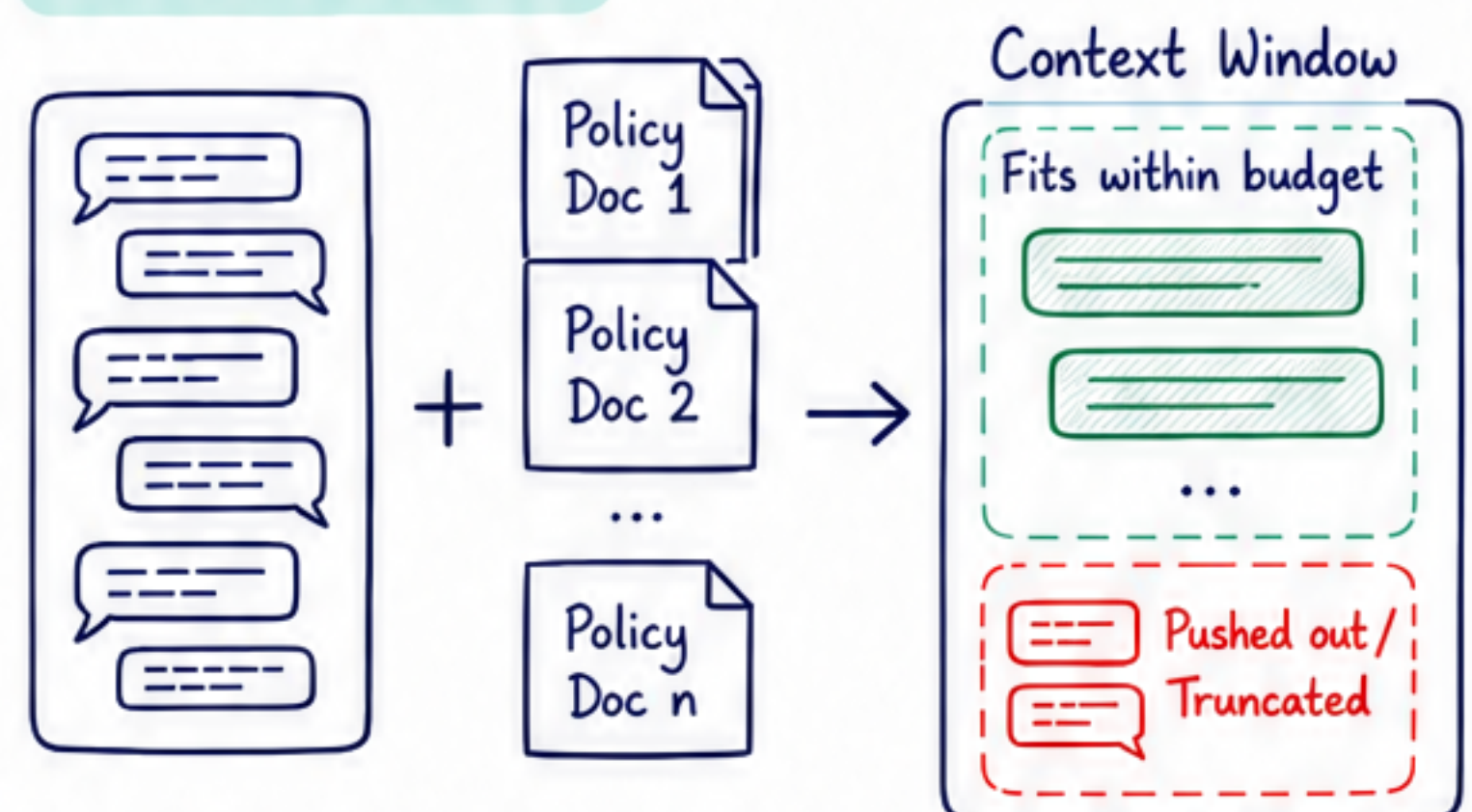
4. FLOW: FROM INPUT TO RESPONSE



5. CORE IDEAS

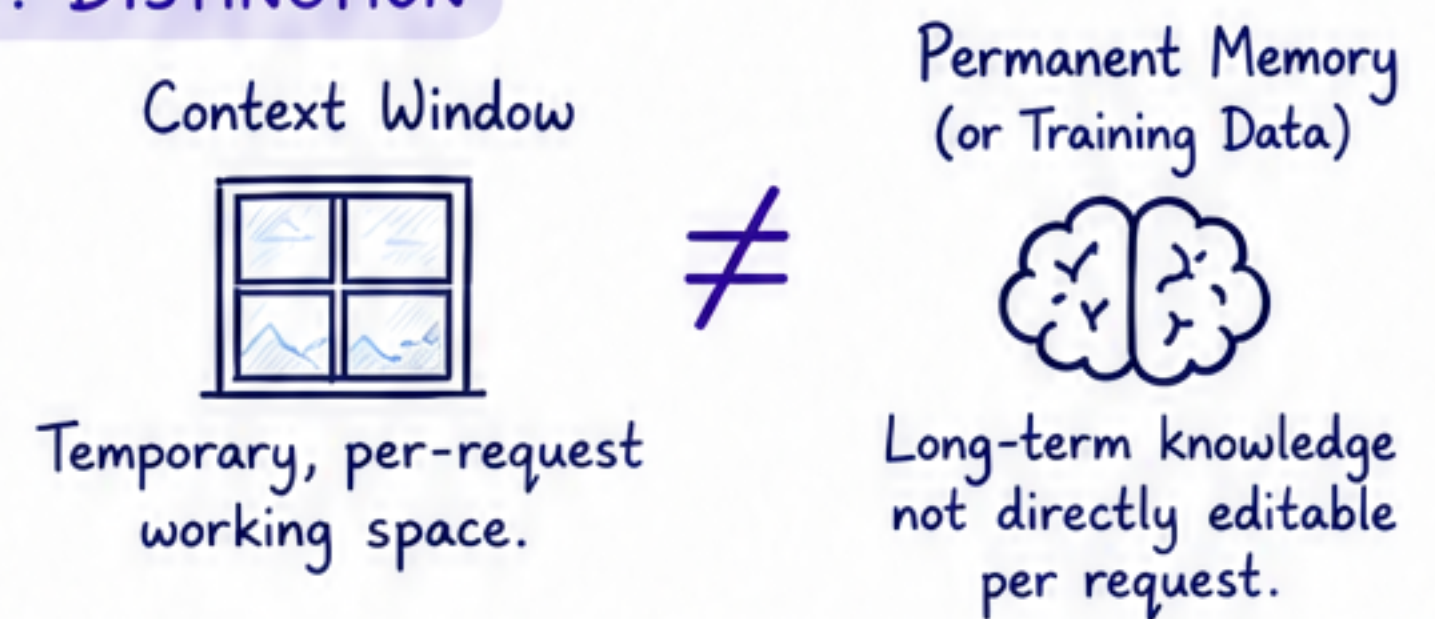
- System Instructions: Rules, role, and behavior guidance.
- Conversation History: Past turns in the dialogue.
- Retrieved Context: External documents or knowledge brought in.
- Token Budget: The total space available for this request.
- Output Budget: Portion reserved for the model's response.

6. QA EXAMPLE



A long support conversation plus several policy documents may push earlier constraints out of usable context or cause truncation.

7. DISTINCTION



8. TESTER CHECKS

- ✓ Try long prompts and documents.
- ✓ Place important info at beginning, middle, end.
- ✓ Check for truncation effects.
- ✓ Look for conflicting context.
- ✓ Test long multi-turn sessions.

9. COMMON MISTAKES

- ✗ Treating context as unlimited.
- ✗ Confusing context with training data.
- ✗ Assuming all included text is equally influential.
- ✗ Ignoring output budget.

10. MEMORY

Context window = the model's limited working space for this request.

Hallucination



Definition:

Hallucination is when a generative AI system produces content that appears plausible but is incorrect, fabricated, unsupported, or inconsistent with available evidence.



Why it matters:

- Fluent language can hide wrong facts.
- Hallucinations can create business, security, or compliance risk.



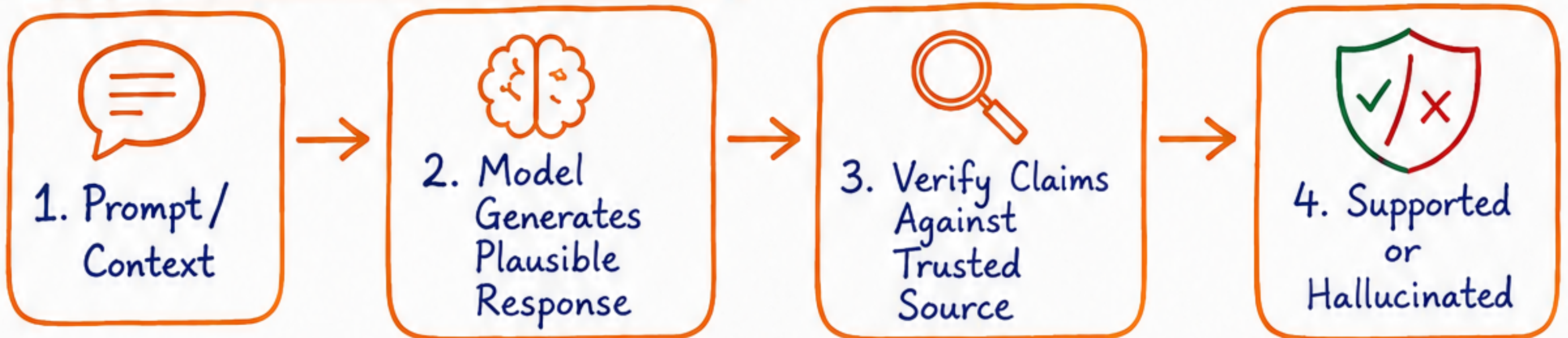
Mental model:

Confident answer without evidence.

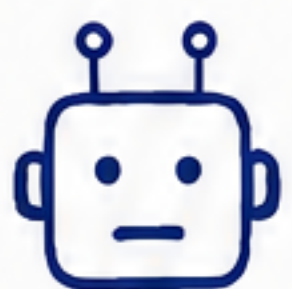
Core ideas

- ✓ claim – a statement about the world
- ✓ evidence – information from a trusted source
- ✓ factuality – matches reality and sources
- ✓ support – evidence that backs the claim
- ✓ contradiction – conflicts with trusted evidence

Flow (How to evaluate a response)



QA/Test Automation Example:



The refund policy allows returns within 90 days with no restocking fee.



→ This policy does not exist in the company's official documents. The response is a hallucination.

Distinction:

Hallucination is not simply a grammar error; a response can be fluent and still hallucinate.



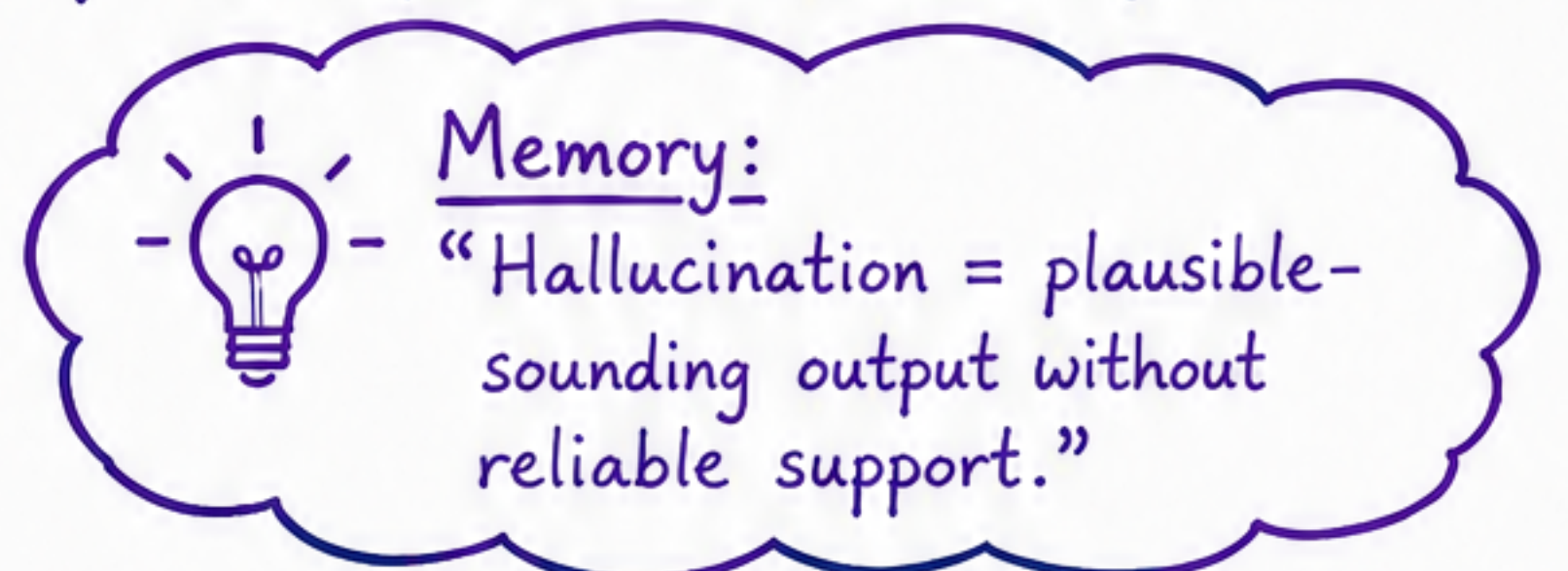
Tester Checks:

1. Factual claims
2. Citations / sources
3. Unsupported details
4. Missing-evidence behavior
5. Repeated runs (consistency)



Common Mistakes:

- ✗ Trusting confident tone
- ✗ Testing only obvious facts
- ✗ Equating relevance with truth
- ✗ Not checking sources



Retrieval-Augmented Generation (RAG)

1. DEFINITION



RAG is an AI application pattern that first retrieves relevant information from an external knowledge source and then supplies that context to a generative model to help produce a grounded answer.

2. WHY IT MATTERS



- It lets systems use current/private domain knowledge without retraining the model.
- Quality depends on both retrieval and generation.

3. MENTAL MODEL



Open-book exam: find the right pages, then answer from them.

4. FLOW (HIGH LEVEL)



① User Query



② Retrieve Relevant Chunks



③ Add Context to Prompt



④ LLM Generates Answer



⑤ Evaluate

5. CORE IDEAS (KEY TERMS)

- Query: the user's question or need.
- Retriever: component that finds relevant information.
- Chunk: a small piece of a document used for retrieval.
- Context: retrieved chunks inserted into the prompt.
- Generator: the model that produces the final answer using the context.

6. QA EXAMPLE



An HR policy assistant retrieves the relevant sections from the company's leave-policy document (e.g., types of leave, eligibility, carry-forward rules) before answering an employee question about leave balance.

7. DISTINCTION: RAG ≠ FINE-TUNING



Retrieval adds context at runtime from external sources. It does not retrain the base model. Fine-tuning updates model weights during training.

8. TESTER CHECKLIST



- ✓ Correct documents/chunks retrieved?
- ✓ Irrelevant or missing context?
- ✓ Is the answer supported by the retrieved context?
- ✓ Are the source documents current (stale docs)?
- ✓ Any retrieval failures (no/low results, bad ranking)?
- ✓ Any generation failures (hallucination, contradiction, poor grounding)?

9. COMMON MISTAKES



- ✗ Blaming the model for bad retrieval.
- ✗ Assuming retrieved context is always correct.
- ✗ Testing only the answer, not the retrieval.
- ✗ Using huge chunks with irrelevant information.



10. MEMORY HOOK

RAG = retrieve evidence first, generate answer second.

11. WHAT'S NEXT?



Next: Embedding.



ACCURACY =

RETRIEVAL QUALITY

Get the right context.

+

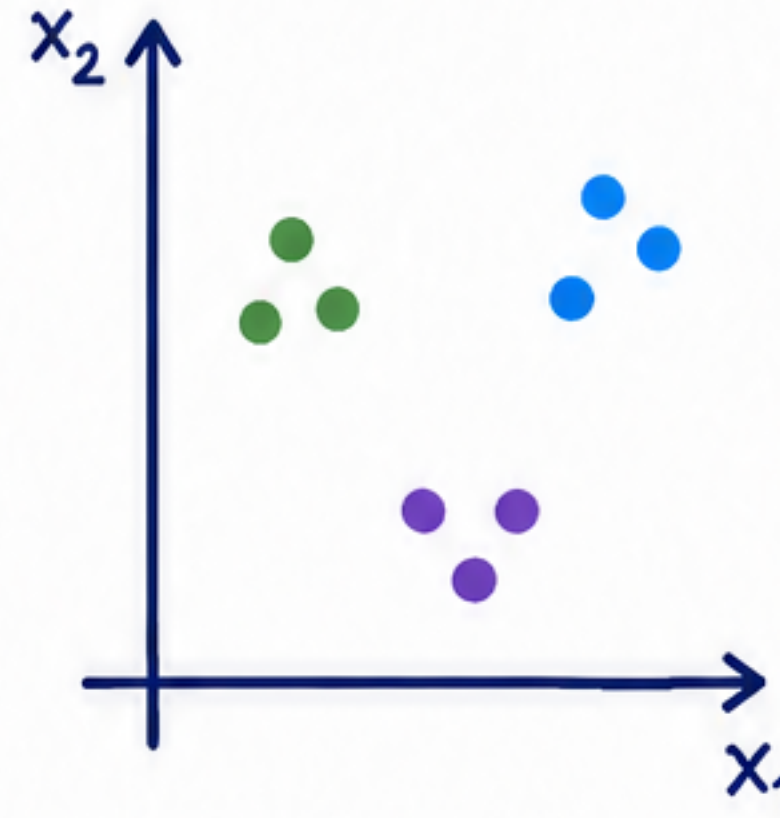
GENERATION QUALITY

Use it correctly to answer.

Embedding

1. DEFINITION

An embedding is a numeric vector representation of data, such as text, designed so items with similar meaning or characteristics are located closer together in vector space.



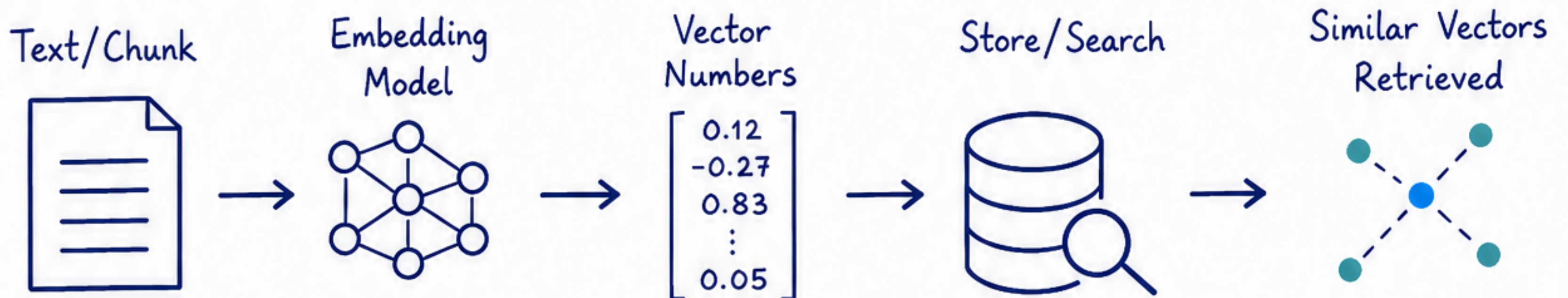
2. WHY IT MATTERS

Embeddings enable semantic search and retrieval even when wording differs.

3. MENTAL MODEL

Place meanings as points on a map; similar meanings live nearby.

4. FLOW (END-TO-END)



5. CORE IDEAS

- Text chunk
- Embedding model
- Vector
- Similarity
- Semantic search



6. QA EXAMPLE

Query: "reset forgotten password"
↓ (semantic search)

Retrieved Document:
"Account credential recovery"

Match occurs even without exact keyword match.

7. DISTINCTION



Embedding $\neq$ Original Text



Embedding $\neq$ Database

$$\begin{bmatrix} 0.12 \\ -0.27 \\ \vdots \\ 0.05 \end{bmatrix}$$

It is the numeric representation stored and queried by vector search.

8. TESTER CHECKS

- ✓ Semantically similar queries (with different wording)
- ✓ Domain-specific terms and jargon
- ✓ Ambiguous words and multiple meanings
- ✓ Multilingual data and translations
- ✓ Irrelevant nearest neighbors (false positives)

9. COMMON MISTAKES

- ✗ Treating embeddings as human-readable meaning
- ✗ Assuming similarity means truth
- ✗ Mixing incompatible embedding models
- ✗ Ignoring chunk quality (too big, too small, or poorly split)

10. MEMORY HOOK

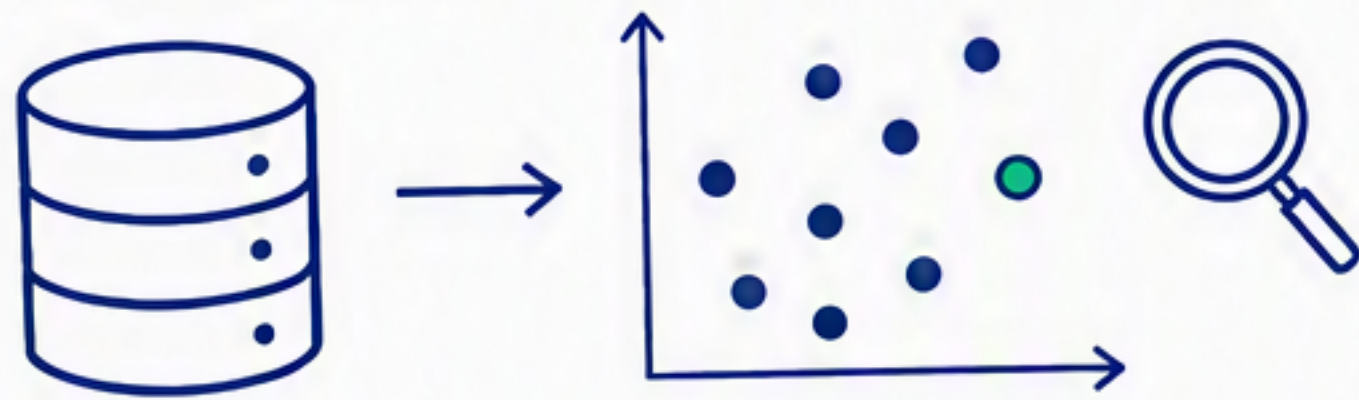


"Embedding = meaning represented as numbers for similarity search."

Vector Database

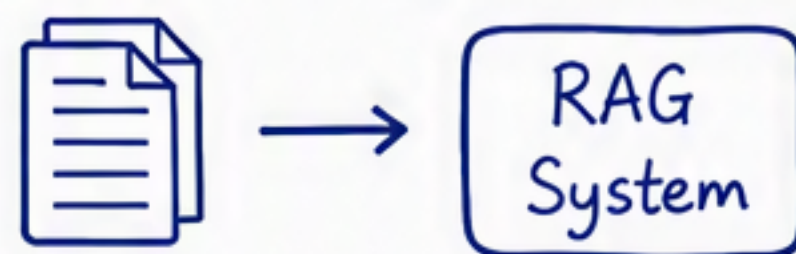
1. DEFINITION

A vector database stores and indexes vector embeddings so applications can search for items that are similar in meaning or characteristics.



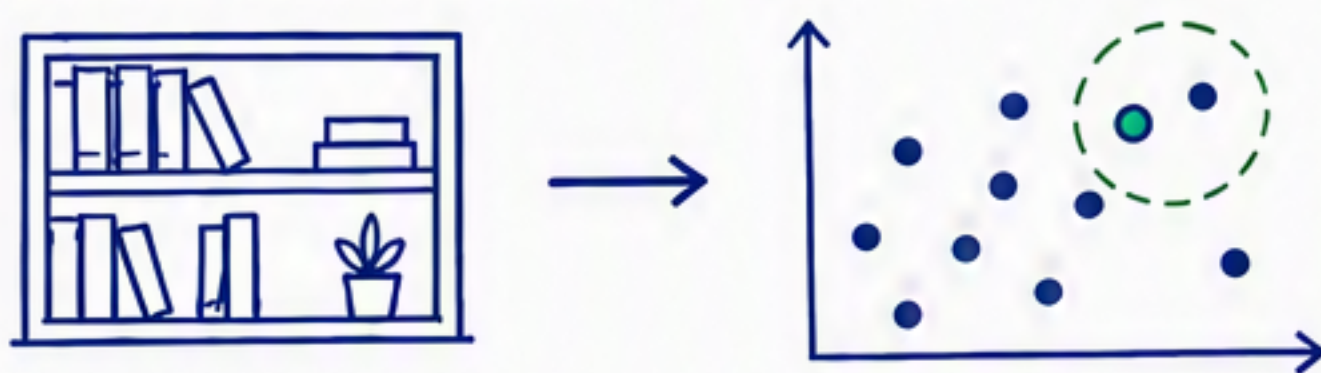
2. WHY IT MATTERS

It is a common retrieval component in RAG systems and supports fast similarity search over many embeddings.



3. MENTAL MODEL

A library organized by meaning rather than exact keywords.



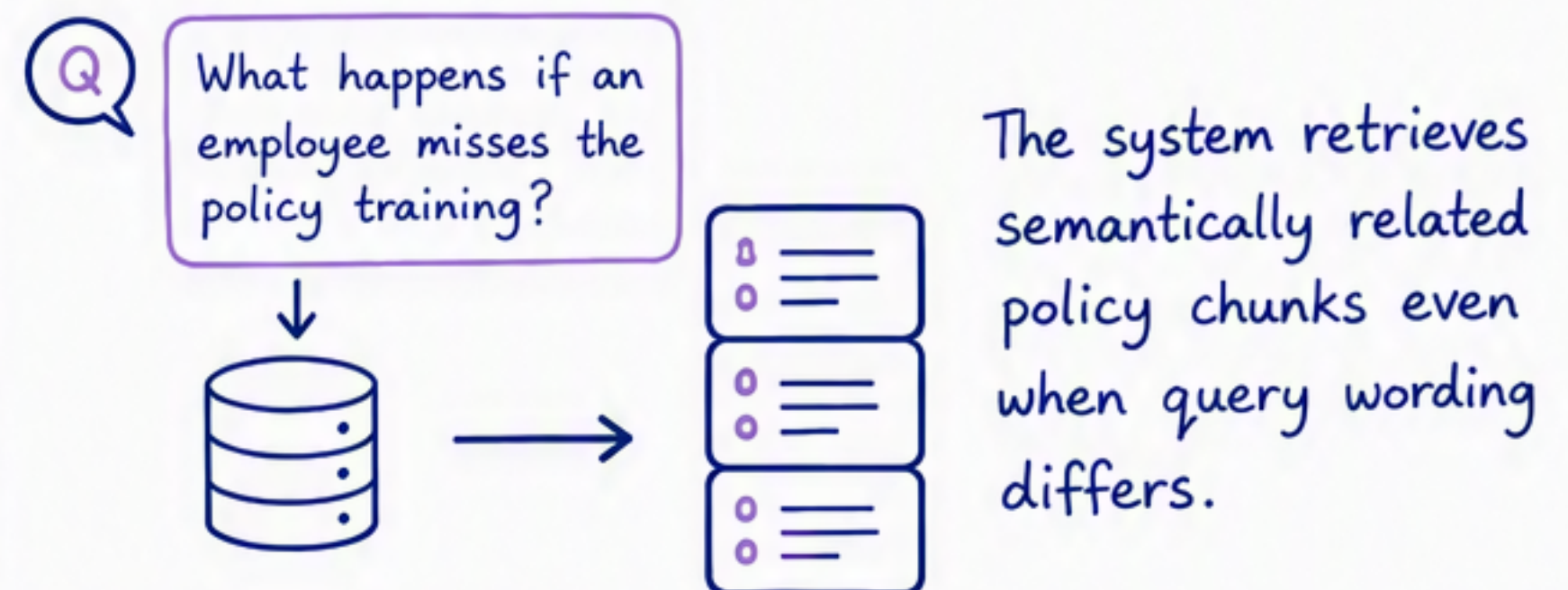
4. FLOW (AT A GLANCE)



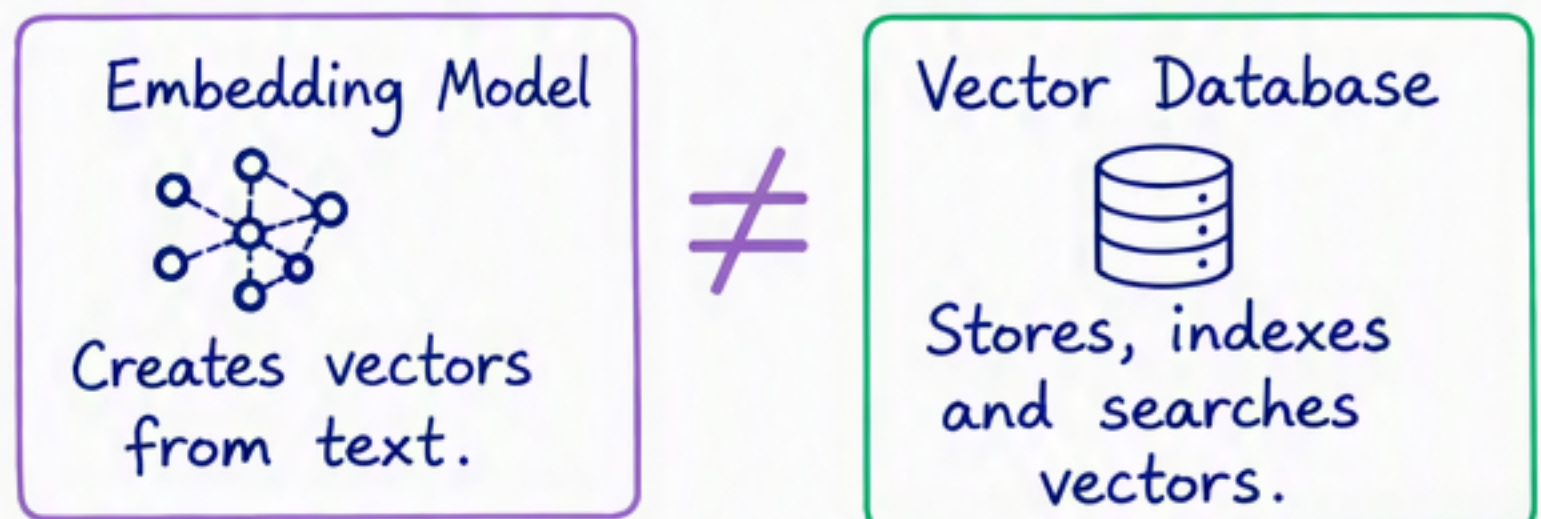
5. CORE IDEAS

- $[0.1 \ -0.3 \ \dots \ 0.7]$ Vector: numerical representation of meaning.
- Index: organizes vectors for efficient search.
- Metadata: extra information (e.g., source, date, doc type, owner, policy id).
- Similarity Search: finds vectors close to the query in meaning.
- Top Results: returns the most similar items.

6. QA EXAMPLE



7. DISTINCTION



The embedding model creates vectors; the vector database stores/indexes/searches them.

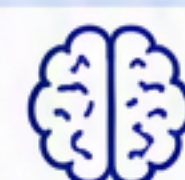
8. TESTER CHECKS (GOOD ✓)

- ✓ Correct metadata filters applied.
- ✓ Top results are relevant and useful.
- ✓ Missing or duplicate chunks identified.
- ✓ Indexes are up to date; no stale data.
- ✓ Semantic edge cases tested (synonyms, paraphrases, abbreviations, typos).

9. COMMON MISTAKES (RISK !)

- ✗ Assuming nearest means correct.
- ✗ Ignoring metadata and filters.
- ✗ Mixing embeddings from incompatible models.
- ✗ Testing only exact wording queries.

10. MEMORY HOOK



Vector database = store and search embeddings by similarity.

AI Agent



DEFINITION

An AI agent is a system that uses a model together with instructions, tools, context, and an execution loop to work toward a goal and take actions.

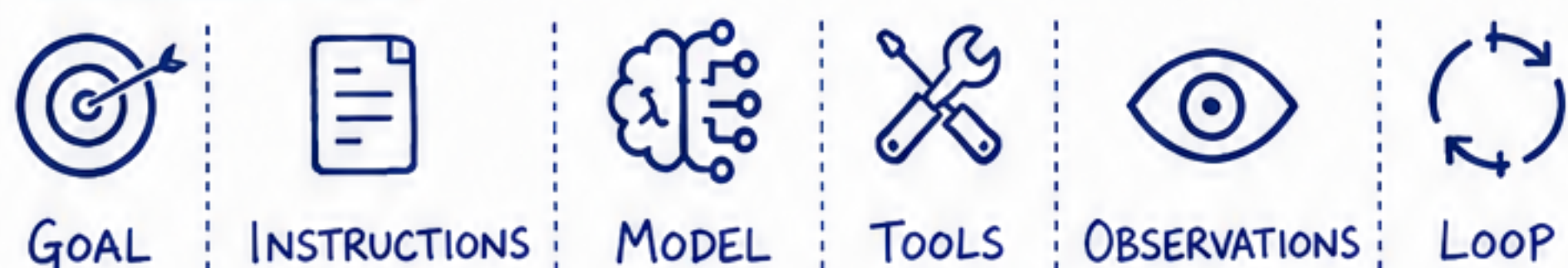
WHY IT MATTERS

Agents do more than generate text; they can choose tools, inspect results, and continue through multiple steps.

MENTAL MODEL



CORE IDEAS

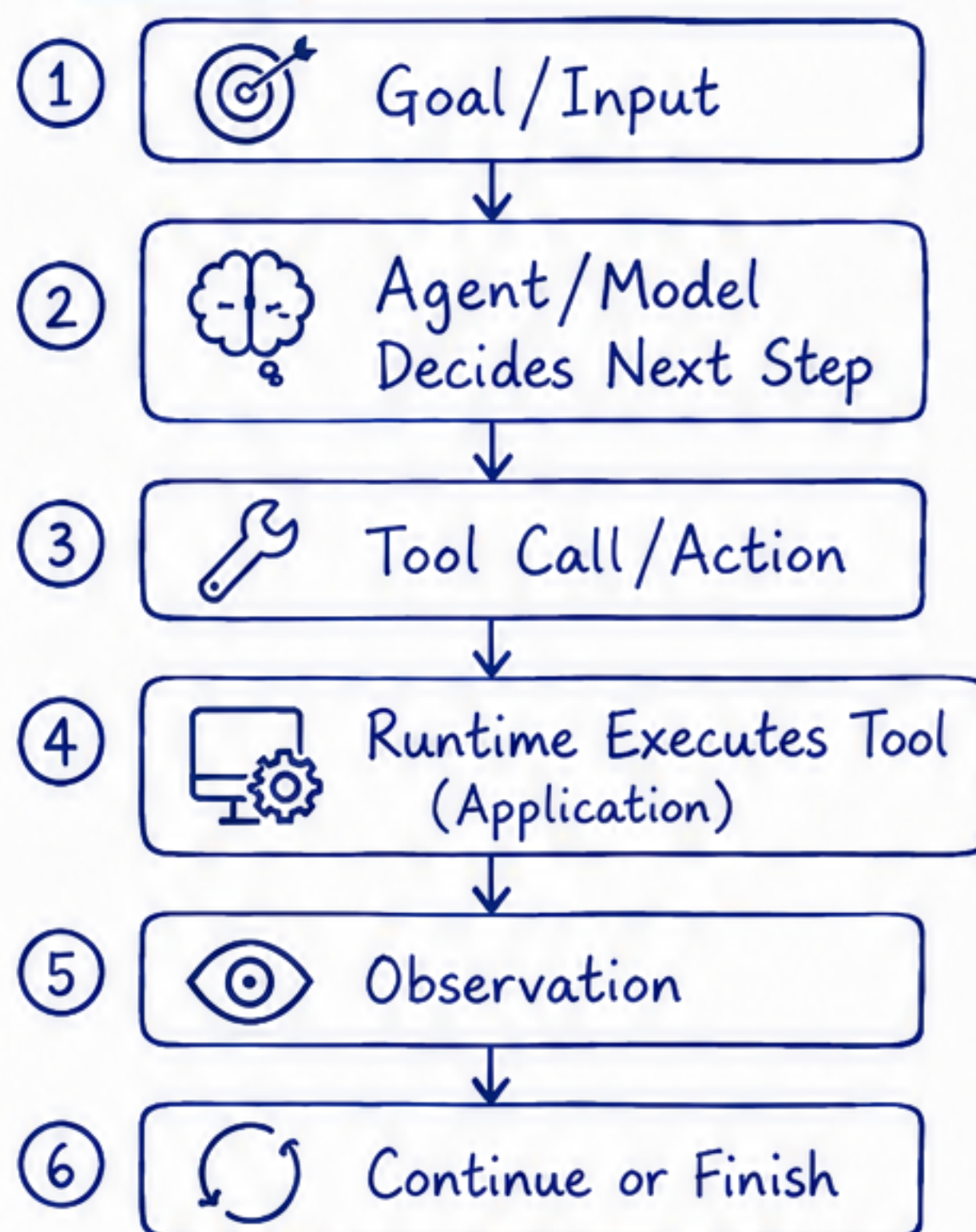


QA EXAMPLE



A defect triage agent reads a bug report, looks up logs, checks ownership, proposes severity, and creates a structured recommendation.

FLOW



DISTINCTION

Agent $\neq$ LLM alone.
Agent is the application loop around the model plus tools/state.



The runtime/application executes tools – not the model.

TESTER CHECKS

- ✓ Goal completion
- ✓ Correct tool selection
- ✓ Valid arguments/inputs
- ✓ Handling failures and timeouts
- ✓ Stopping behavior (no unnecessary loops)

COMMON MISTAKES

- ⚠ Testing only final text
- ⚠ Assuming model itself executes tools
- ⚠ Ignoring intermediate actions/observations
- ⚠ Allowing excessive permissions or broad tools

MEMORY



Agent = model + tools + loop
working toward a goal.

NEXT



Agentic AI.

Agentic AI

1. Definition



Agentic AI refers to AI systems designed to pursue goals through multi-step planning, decisions, tool use, observations, and adaptation rather than producing only a single response.

2. Why it matters

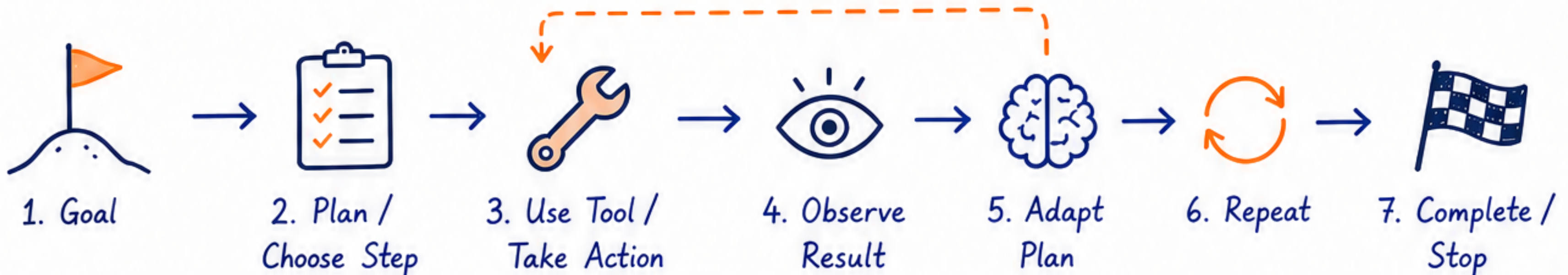
- More autonomy creates more possible failure paths.
- Requires testing of sequences, permissions, recovery, and task completion.



3. Mental Model

Not one answer, but a journey of decisions and actions.

4. How it Works (Typical Flow)



5. Core Ideas

- Autonomy** – Acts within given goals and boundaries.
- Plan** – Breaks goal into steps and chooses the next best step.
- Action** – Uses tools / APIs / systems to make changes or gather data.
- Observation** – Sees results and captures feedback.
- State** – Remembers context, results, decisions, and constraints.
- Completion** – Knows when the goal is satisfied or should stop.

6. QA Example



A release-readiness agent:

- ① Gathers test results from the test management system.
- ② Checks open defects and their severity.
- ③ Queries deployment status from the CD pipeline.
- ④ Analyzes the information and recommends “Go” or “No-Go” with rationale.

7. Key Distinction

AI Agent

A concrete system / application that performs tasks.



Agentic AI

The broader capability / approach of goal-directed, multi-step behavior (planning, acting, observing, adapting).

All AI Agents use Agentic AI principles, but not every workflow with rules is Agentic.

8. What Testers Should Check

- ✓ Correct trajectory to goal.
- ✓ Loops: exit conditions and iteration limits.
- ✓ Retries and backoff behavior.
- ✓ Avoids unsafe or destructive actions.
- ✓ Respects permission boundaries.
- ✓ Clear completion criteria.
- ✓ Graceful recovery from failures.

9. Common Mistakes

- ✗ Validating only the final answer, not the journey.
- ✗ Allowing unlimited retries or loops.
- ✗ Unclear or missing stop condition.
- ✗ Granting excessive autonomy without guardrails.

10. Memory Hook

“Agentic AI = goal-driven AI that plans, acts, observes, and adapts.”

Tool Calling

1) DEFINITION



Tool calling is a pattern where a model requests that an application invoke a defined external function or tool with structured arguments; the application/runtime validates and executes it, then returns the result to the model or workflow.

2) WHY IT MATTERS



It connects language models to APIs, databases, search, files, and actions.

3) MENTAL MODEL



Model fills out a request form; the application performs the actual action.

4) FLOW (END-TO-END)



1. User Request



2. Model Selects Tool + Arguments



3. Application Validates



4. Runtime Executes Tool



5. Tool Result



6. Model/Agent Continues

5) CORE IDEAS



Tool Schema: The contract (name, description, input/output schema).



Tool Name: Unique identifier of the tool.



Arguments: Structured data the tool needs.



Runtime: The application/system that validates and executes.



Tool Result: The output returned to the model or workflow.

6) QA EXAMPLE



User: "Get order 123"



Model: Chooses tool 'getOrder' with arguments {orderId: 123}



Application: Calls Order API GET /orders/123



Result returned to model: {orderId: 123, status: "Shipped", ...}



Model/Agent: Continues with the result (answers user, etc.)

7) DISTINCTION (EXECUTION BOUNDARY)

MODEL (CHOOSES/REQUESTS)



- Understands request
- Selects the best tool
- Fills arguments
- Requests execution
- Does NOT execute external actions

APPLICATION / RUNTIME (EXECUTES)



- Validates tool + args
- Checks permissions
- Executes the tool
- Handles errors/retries
- Returns result

8) TESTER CHECKLIST



- ① Correct tool selected
- ② Valid arguments (types, required, formats, ranges)
- ③ Authorization & permissions
- ④ Malformed / unexpected outputs
- ⑤ Retries & error handling
- ⑥ Tool-result handling in model/agent

9) COMMON MISTAKES



- ✗ Saying LLM directly executes the API
- ✗ Trusting arguments blindly
- ✗ Missing authorization checks
- ✗ Ignoring tool failures

10) MEMORY



Tool calling = model requests; application executes.

11) NEXT



Next: Model Context Protocol (MCP).

Model Context Protocol (MCP)

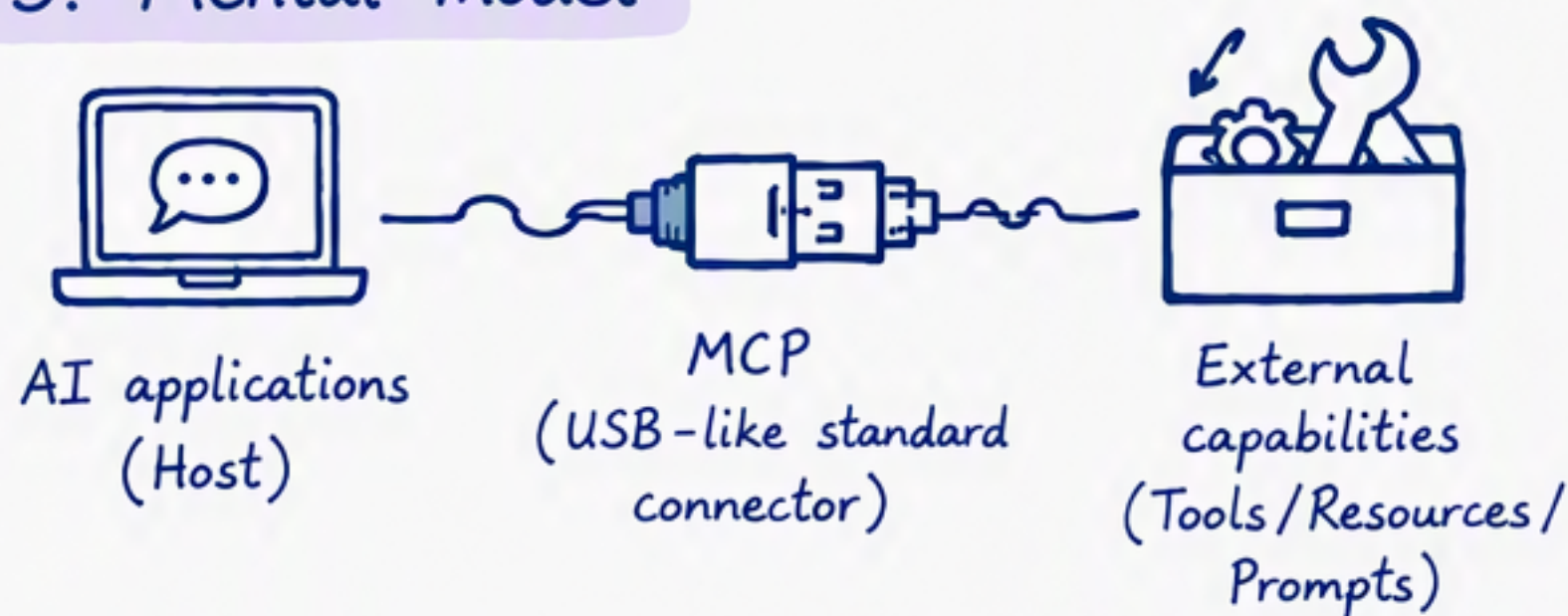
1. Definition

Model Context Protocol (MCP) is an open protocol for connecting AI applications to external tools, resources, and prompts through a standardized client-server architecture.

2. Why it matters

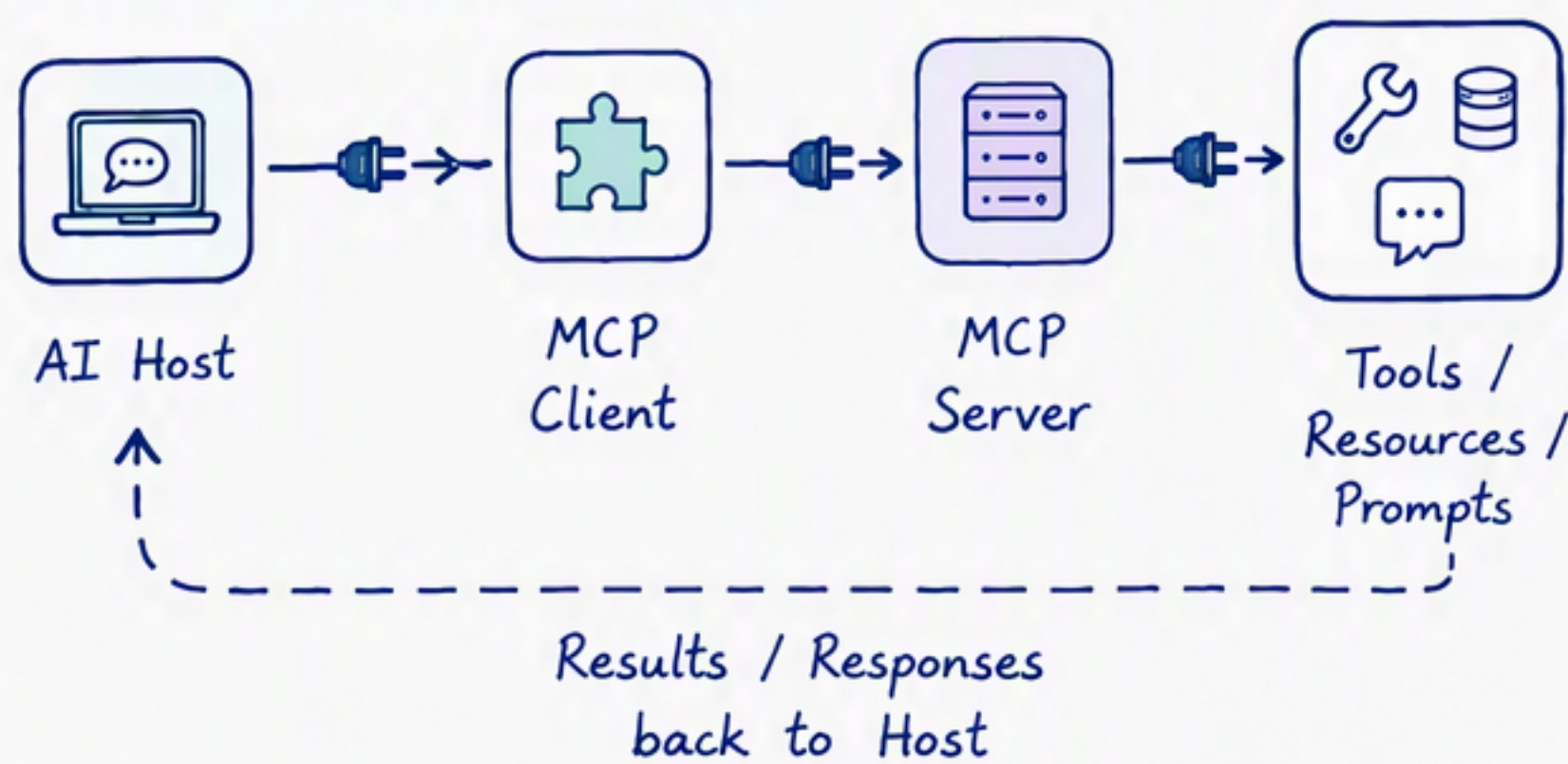
- It reduces custom integration work.
- It creates a consistent way for AI hosts to discover and use capabilities exposed by MCP servers.

3. Mental model



MCP is the standard connector that makes everything plug-and-play.

4. Flow (High Level)



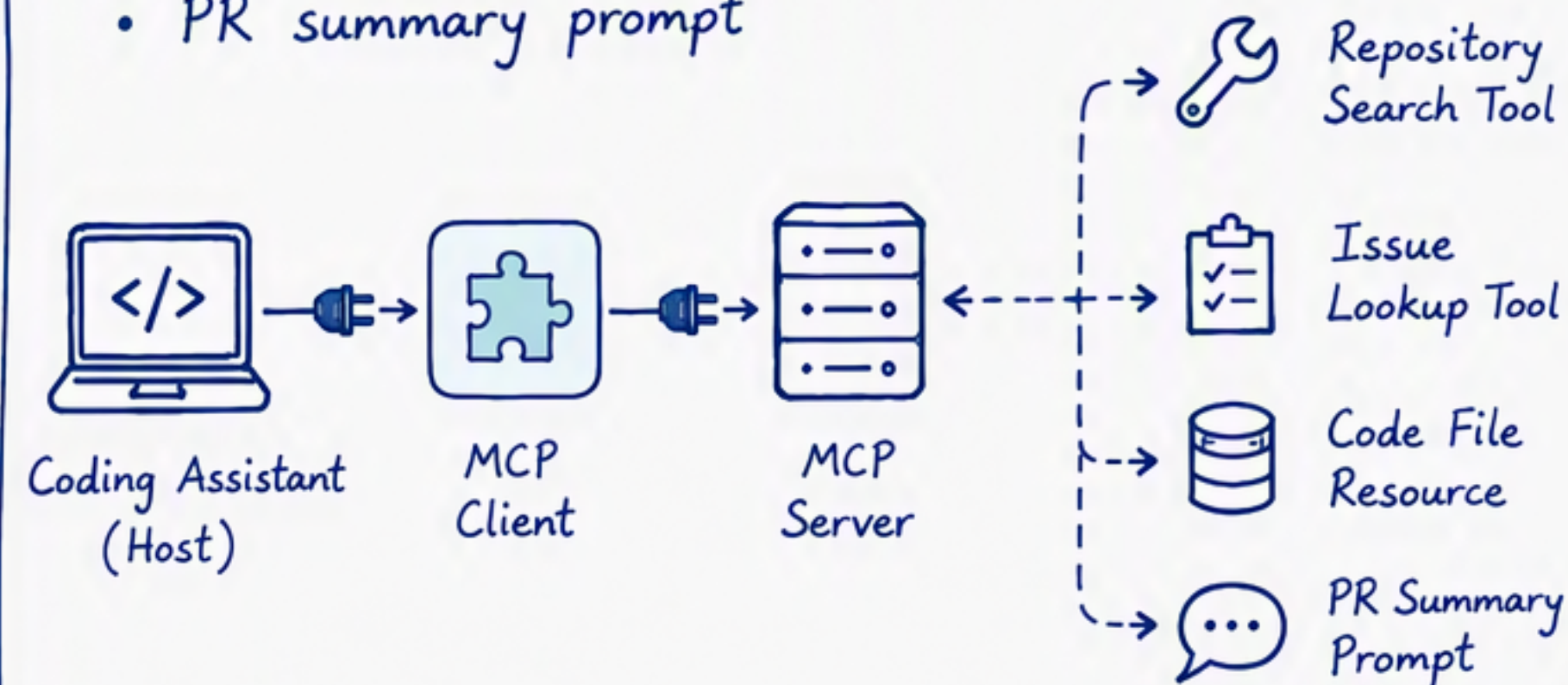
5. Core ideas

- Host - The AI application that initiates and consumes capabilities.
- Client - The component in the host that speaks MCP and manages connections.
- Server - Exposes capabilities (tools, resources, prompts) via MCP.
- Tool - An action the server can execute (e.g., search, create issue).
- Resource - Data or content the server can provide (e.g., files, docs, logs).
- Prompt - Reusable instructions or templates the server can provide.

6. QA Example

A coding assistant (Host) connects via MCP Client to an MCP Server that exposes:

- Repository search tool
- Issue lookup tool
- Code file resource
- PR summary prompt



The assistant discovers these capabilities, uses them, and gets results back.

7. Distinction

- ≠ MCP ≠ AI model (LLM)
- ≠ MCP ≠ AI agent
- ≠ MCP ≠ Database
- ≠ MCP is a protocol - a set of rules for communication and capability exchange.

8. What a Tester Should Check

- ① Capability discovery - Are capabilities correctly listed and described?
- ② Schema correctness - Do tools/resources/prompts conform to the expected schema?
- ③ Authorization - Are permissions and access controls enforced?
- ④ Server errors - How does the server handle invalid requests and failures?
- ⑤ Unsafe tool exposure - Are dangerous or destructive tools properly restricted?
- ⑥ Transport & session behavior - Connection setup, timeouts, reconnection, concurrency.

9. Common Mistakes

- ✗ Calling MCP on AI model.
- ✗ Confusing MCP server with an LLM server.
- ✗ Assuming all MCP servers are trusted.
- ✗ Ignoring permissions and access boundaries.

10. Memory Hook



"MCP = standard protocol connecting AI apps to external capabilities."

11. Next

→ AI Testing

AI Testing

1. DEFINITION



AI Testing is the systematic evaluation of AI-enabled systems to determine whether their behavior, outputs, risks, and quality meet defined requirements across representative inputs and operating conditions.

2. WHY IT MATTERS



AI behavior can depend on data, prompts, model configuration, context, retrieval, tools, and nondeterminism; quality is often multidimensional.

3. MENTAL MODEL

Traditional software testing



Data / Model behavior



Evaluation metrics



Safety / Risk checks



5. CORE IDEAS (BUILDING BLOCKS)

Test Input



What we ask or provide to the system.

Expected Criteria



What good looks like.

Dataset



Curated, diverse, representative examples.

Metric



How we measure quality.

Threshold



Pass / Fail boundary.

Risk



Potential harm or unacceptable outcome.

4. AI TESTING FLOW

1 Define Quality Goal



2 Build Representative Test Cases



3 Run AI System



4 Measure Outputs / Behavior



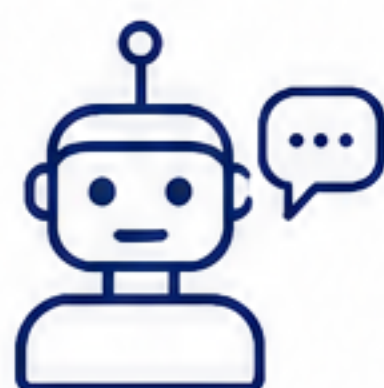
5 Analyze Failures



6 Regression Gate



6. QA EXAMPLE



A support assistant is tested for relevance, correctness, groundedness, safety, latency, and failure handling across a curated dataset.

7. DISTINCTION



AI Testing $\neq$ only model testing; test the end-to-end system.

8. TESTER CHECKS

- ✓ Functional behavior
- ✓ Data quality
- ✓ Robustness
- ✓ Safety
- ✓ Retrieval / Tools
- ✓ Repeatability (nondeterminism)



9. COMMON MISTAKES

- ✗ Exact-match-only testing
- ✗ One-run testing
- ✗ Small, happy-path dataset
- ✗ One metric for everything



10. MEMORY HOOK



"AI Testing = test behavior, quality, data, risks, and the whole system."

NEXT:

LLM Evaluation



LLM Evaluation

1. DEFINITION

LLM Evaluation is the structured process of assessing language-model or LLM-application behavior against defined quality criteria using test cases, metrics, graders, human review, or a combination.



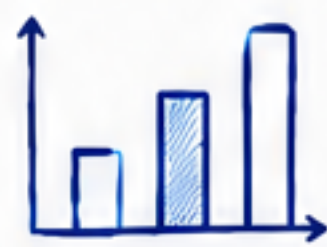
2. WHY IT MATTERS

- Open-ended outputs often cannot be tested with exact expected strings.
- Evaluation must measure dimensions such as relevance, correctness, groundedness, safety, and task completion.



3. MENTAL MODEL

Test suite + Rubric + Measurements



= Understanding language behavior

4. EVALUATION FLOW

- ① Evaluation Dataset
- ↓
- ② Run LLM/App
- ↓
- ③ Capture Output
- ↓
- ④ Apply Metrics/Graders
- ↓
- ⑤ Scores / Labels
- ↓
- ⑥ Analyze Failures
- ↓
- ⑦ Regression Gate

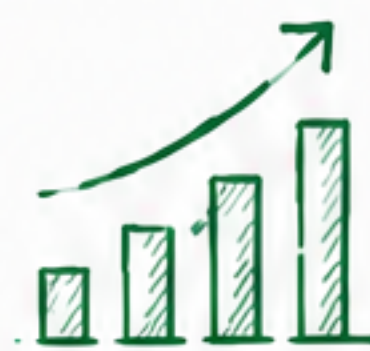


5. CORE IDEAS

- Test case - input + context + intent
- Expected criteria - what good looks like
- Metric - how we measure a dimension
- Grader - rule, model, or human scoring
- Threshold - pass/fail or quality bar

6. QA EXAMPLE

Run 100 support questions through the assistant and score:



- ✓ Answer relevance
- ✓ Groundedness (uses provided info)
- ✓ Safety (no harmful content)
- ✓ Latency (response time)

Each dimension is measured; together they give a quality picture.

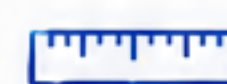
7. DISTINCTION

- Evaluation is broader than a single metric.
- No one score proves overall quality.
- Multiple methods (automatic metrics, model graders, and human review) may be needed.



8. TESTER CHECKLIST

- ✓ Representative cases (coverage of intents, domains, and difficulty)
- ✓ Reliable scoring (clear rubric or grader)
- ✓ Thresholds defined (quality bars per metric)
- ✓ Reproducibility (same setup, same results)
- ✓ Failure analysis (find root causes)
- ✓ Regression trends (track over time)



9. COMMON MISTAKES

- ✗ One metric only
- ✗ Tiny dataset
- ✗ Trusting averages (hide bad pockets)
- ✗ No human calibration



10. MEMORY HOOK



"LLM Evaluation = repeatable measurement of language-system quality."

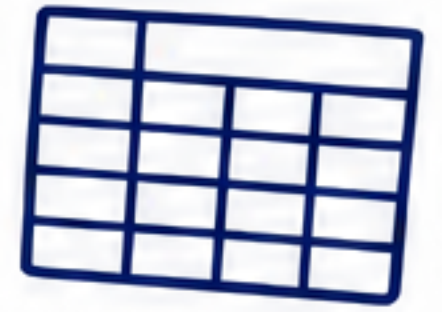
11. NEXT



Evaluation Dataset



Evaluation Dataset



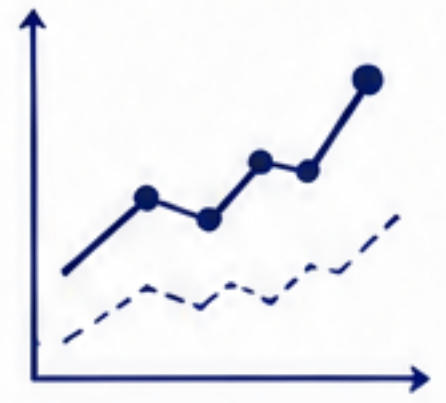
1. Definition



An evaluation dataset is a curated collection of representative test inputs and, when needed, expected outputs, references, labels, contexts, or criteria used to evaluate an AI system consistently.

2. Why it matters

- ✓ Evaluation quality is limited by the coverage and quality of test cases.
- ✓ Datasets support regression comparisons over time.

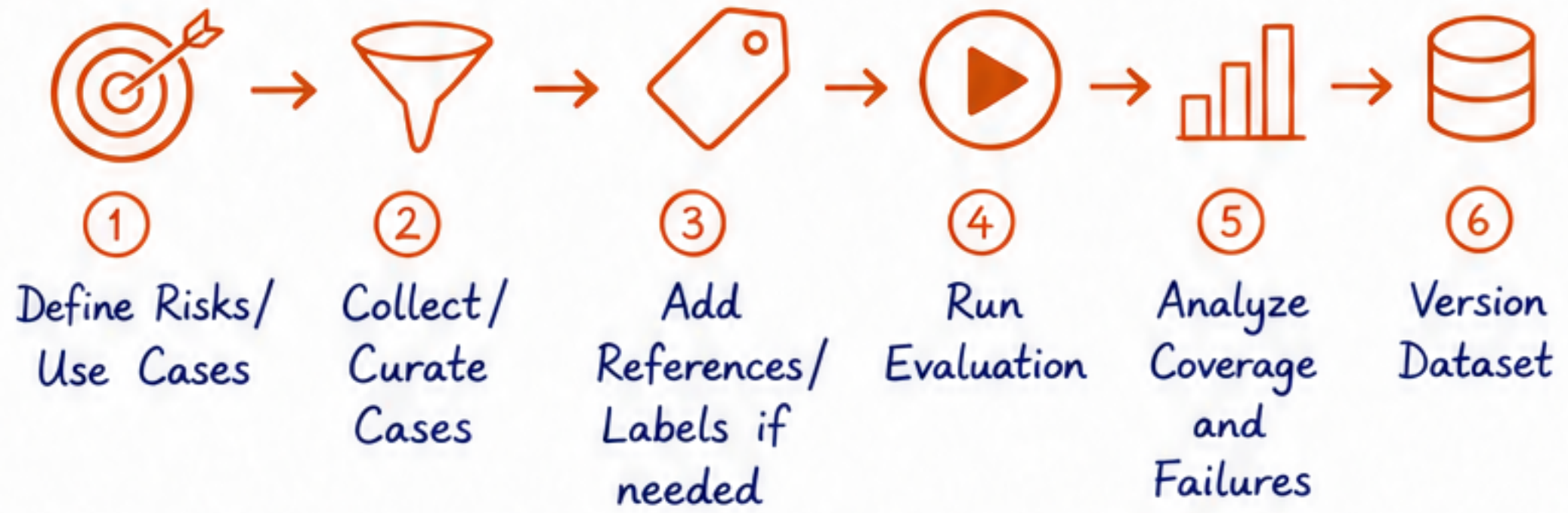


3. Mental Model

Think of the evaluation dataset as an AI test-suite data covering:



4. Typical Flow



5. Core Components

Component	What it contains	Example
Input	The test prompt, query, or request	"How do I reset my password?"
Expected/Reference	Ground truth answer or correct behavior	Step-by-step reset guide with link
Context	Additional info: user state, history, channel, locale, environment	User is logged in, on web, EN-US
Label	Category or class for analysis	Use Case: Password Reset
Metadata	Source, author, date, version, difficulty, risk level, tags	v1.2, 2025-05-20, High Risk, Billing
Coverage	Mapping to risks, features, domains, languages, etc.	Billing ✓, Security ✓, Edge ✓, French ✓

6. QA Example

A support chatbot evaluation dataset may include:

- Billing questions and refunds
- Password reset flow
- Subscription cancellation
- Ambiguous questions
- Unsupported requests
- Unsafe or harmful prompts
- Multilingual cases (e.g., EN, ES, FR, HI)



7. Distinction



8. Tester Checks

- ✓ Representativeness of real-world use
- ✓ Duplicates removed or minimized
- ✓ No leakage from training data
- ✓ Class / use-case balance
- ✓ Edge, negative, and adversarial cases
- ✓ Clear references or labels
- ✓ Versioned and change-tracked

9. Common Mistakes

- ✗ Only happy paths
- ✗ Too few cases to be meaningful
- ✗ Training/evaluation leakage
- ✗ Stale or outdated cases

10. Memory Hook



"Evaluation dataset = the AI test suite."

11. Next Step



Next: Evaluation Metric.

12. Accuracy: Key Takeaways



- High-quality evaluation depends on coverage, not just quantity.
- Include normal, negative, edge, adversarial, and domain-specific cases.



- Evaluation dataset must be independent of training data to measure true behavior.
- Good datasets enable fair comparisons across models and over time.



Evaluation Metric

1. Definition

An evaluation metric is a defined measure or scoring criterion used to quantify a particular aspect of AI quality.



Metrics may be deterministic, statistical, similarity-based, model-based, binary, categorical, or numeric depending on the task.

2. Why it matters

- No single metric captures all quality.
- Metrics turn vague goals into measurable dimensions.
- Thresholds can define pass/fail quality gates when calibrated carefully.



3. Mental Model

Use the right measuring instrument for relevance, correctness, safety, groundedness, latency, or task completion.



4. Evaluation Flow



5. Core Ideas



Dimension – the quality aspect being measured (e.g., relevance).



Score – the quantitative or scaled value produced.



Label – categorical outcome (e.g., pass/fail, safe/unsafe).



Threshold – the minimum (or maximum) acceptable level.

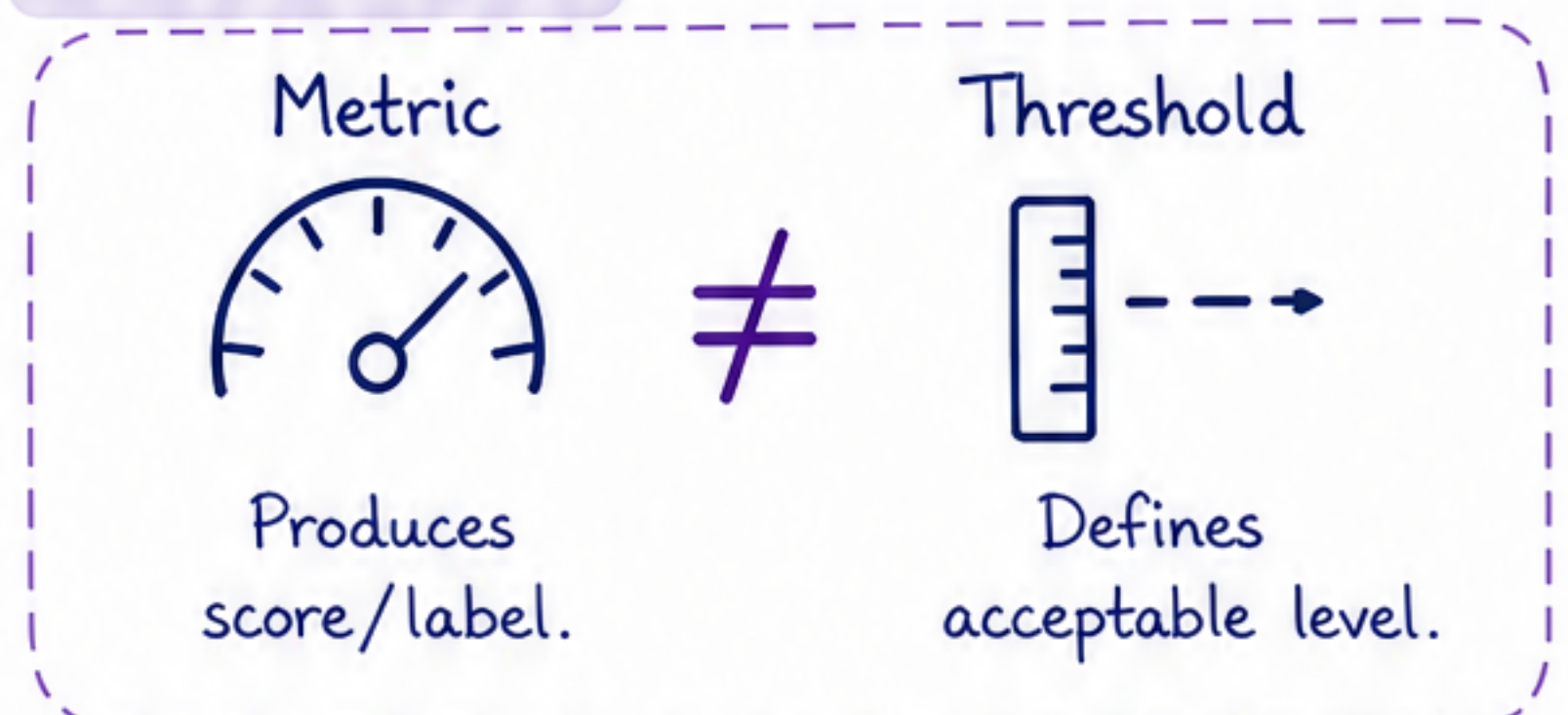


Evidence – the data, output, or rationale used for the score.

6. QA Examples

- ✓ Exact match for status code (e.g., 200 vs expected 200).
- ✓ Answer relevancy for open response (Does the answer address the question?).
- ✓ Groundedness for source-supported claims (Is the answer supported by provided sources?).
- ✓ Latency for performance (e.g., response under 800 ms).

7. Distinction



8. Tester Checks

- ✓ Intended behavior is measured.
- ✓ Scoring direction is correct (higher or lower is better?).
- ✓ Threshold is calibrated with context.
- ✓ Human correlation is verified.
- ✓ Hidden critical failures are not missed.



9. Common Mistakes

- ✗ Using accuracy for every problem.
- ✗ Treating score as objective truth without context.
- ✗ Setting arbitrary thresholds without calibration.
- ✗ Collapsing unrelated metrics into a single score.



10. Memory

"Metric = how quality is measured; threshold = how much quality is enough."

11. Next Step



LLM-as-a-Judge.

12. Accuracy



Multiple metrics often required.

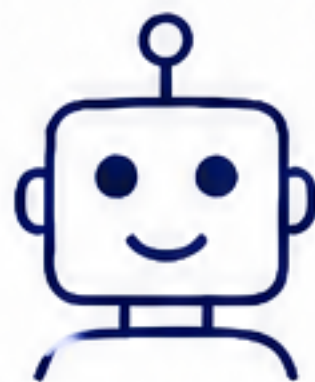


LLM-as-a-Judge



1) DEFINITION

LLM-as-a-Judge is an evaluation approach where a language model grades, compares, or critiques another model/application output using explicit criteria or a rubric.



2) WHY IT MATTERS

- Useful for open-ended qualities such as relevance, clarity, or groundedness where exact assertions are difficult.
- But judge models can be biased or inconsistent.

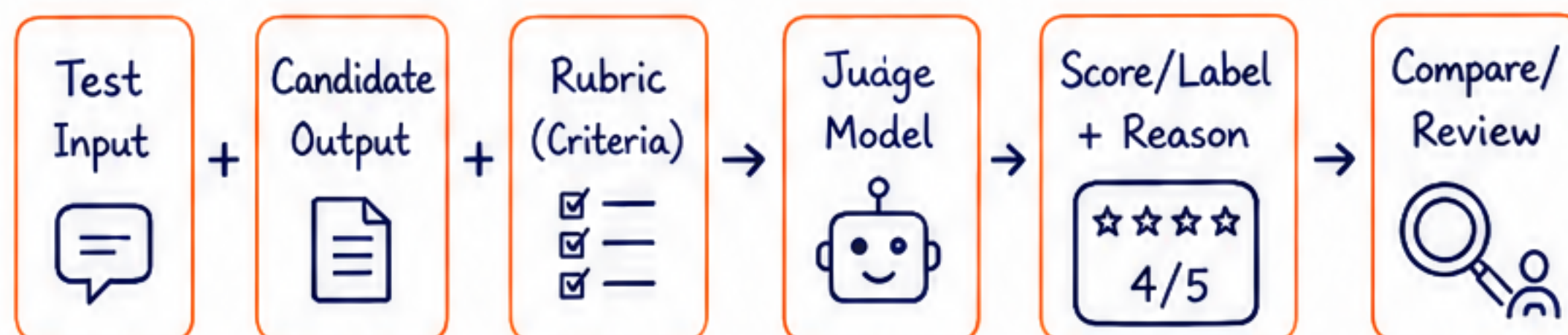


3) MENTAL MODEL



Think of the judge model as an automated reviewer following a scoring rubric. It helps, but it is not an infallible oracle.

4) FLOW (HIGH LEVEL)



5) CORE IDEAS

- Judge Prompt: Provide clear instructions, context, and constraints for the judge.
- Rubric: Define criteria with explicit levels and definitions.
- Score/Label: Produce an explicit rating or label (e.g., 1-5 or Pass/Fail).
- Rationale: Require concise reasons tied to the rubric.
- Calibration: Align scores with human judgments; measure agreement.

7) DISTINCTION



- LLM-as-a-Judge $\neq$ ground truth.
- Use deterministic checks (rules, schemas, unit checks, I/O assertions) when exact verification is possible.
- Calibrate and report uncertainty; align with human labels periodically.

8) TESTER CHECKLIST



- ☒ Is the rubric specific with clear definitions?
- ☒ Have you checked position/order bias (swap order, multiple runs)?
- ☒ Do human reviewers agree at a reasonable level (inter-rater agreement)?
- ☒ Do rationales look grounded and evidence-based?
- ☒ Are you aware of self-judging risks (model overconfidence, sycophancy)?

9) COMMON MISTAKES



- ☒ Prompting vaguely: "rate this."
- ☒ Trusting a single score as absolute truth.
- ☒ No calibration or human alignment.
- ☒ Ignoring variance across runs/models.
- ☒ Replacing reliable deterministic assertions with judge-only checks.
- ☒ Not inspecting or logging rationales.

10) MEMORY HOOK



LLM-as-a-Judge =
model grades model output
using a rubric.



6) QA EXAMPLE

A judge model rates generated test scenarios on completeness and correctness using a 1-5 rubric with explicit definitions.

Score	Completeness (Definition)	Correctness (Definition)
5	Covers all critical aspects and edge cases.	All information is accurate, consistent, and feasible.
4	Covers most aspects; minor gaps.	Mostly accurate with only minor issues.
3	Covers some key aspects; noticeable gaps.	Partially accurate; some notable issues.
2	Covers few aspects; major gaps.	Mostly inaccurate or misleading.
1	Missing almost everything important.	Mostly wrong or contradictory.

Judge output $\rightarrow$ Score (1-5) + Brief rationale referencing the rubric.

Answer Relevancy

1. DEFINITION



Answer Relevancy measures whether the generated answer stays focused on and addresses the user's input.

In referenceless implementations such as DeepEval's metric, the actual output is evaluated against the input without requiring an expected answer.

2. WHY IT MATTERS



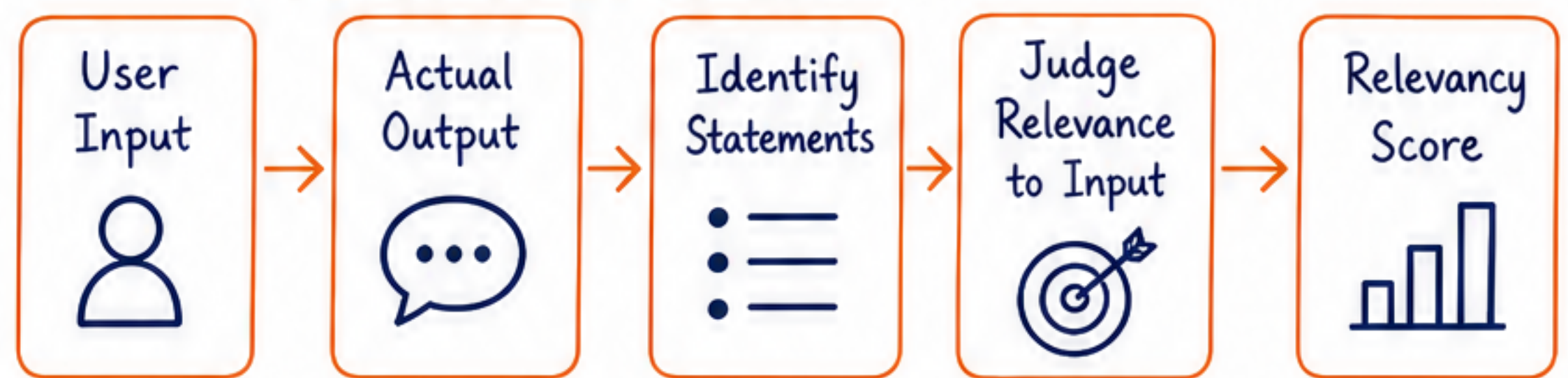
- A grammatically strong answer can still ramble or answer a different question.
- High relevancy does not prove factual correctness.

3. MENTAL MODEL



"How much of this answer actually helps answer what was asked?"

4. EVALUATION FLOW



5. CORE IDEAS



Input: The question or request from the user.



Output: The generated answer.



Relevant Statement: A statement that helps address the input.



Irrelevant Statement: A statement that does not help address the input.



Score: How much of the answer is relevant to the input.

6. QA EXAMPLE



User Input:
"How do I reset my password?"



Relevant (high relevancy)
Provides clear steps to reset the password.



Irrelevant (low relevancy)
Talks about company history or unrelated pricing plans.

7. DISTINCTION



Answer Relevancy checks answer vs input.

It does not prove truth or support from retrieved context.

Relevancy $\neq$ Factual Correctness

Relevancy $\neq$ Faithfulness

8. WHAT A TESTER CHECKS



- ① Directness: Does it directly address the input?
- ② Tangents: Does it go off on irrelevant topics?
- ③ Verbosity: Is it overly long with little relevant value?
- ④ Misunderstood Intent: Did it answer a different question or assume the wrong intent?
- ⑤ Relevant-but-Wrong: It may answer the right question but with incorrect facts – still relevant.

9. COMMON MISTAKES



- ① Equating relevance with correctness.
- ② Requiring an unnecessary reference answer.
- ③ Ignoring verbose but irrelevant statements that dilute focus.
- ④ Evaluating only on obvious questions and ignoring tricky or edge cases.

10. MEMORY HOOK

"Answer Relevancy = did the answer stay on the question?"



11. NEXT STEP

Next: Faithfulness

Answer Relevancy

Focus on the input.
Does the answer address what was asked?



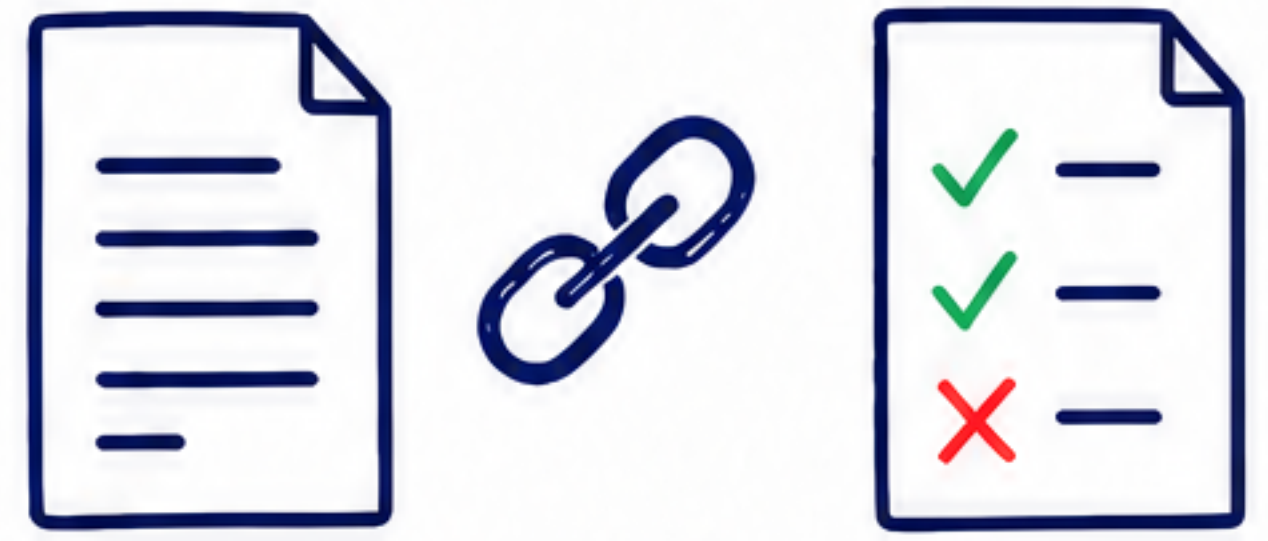
Faithfulness

Focus on the sources.
Is the answer supported by the retrieved context?

Faithfulness

1) Definition

Faithfulness measures whether claims in a generated answer are supported by and do not contradict the provided retrieval context. In RAG evaluation, the actual output is checked against retrieved evidence.



2) Why it matters

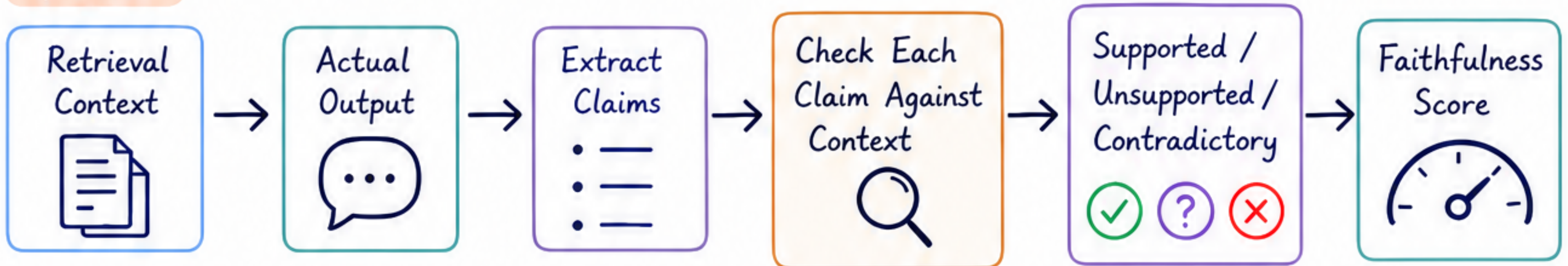
- A relevant answer can still add unsupported details.
- Low faithfulness usually points to generation/prompting behavior after retrieval, not necessarily retrieval quality.

3) Mental model



Open-book rule:
Answer must stay faithful to the retrieved pages.

4) Flow



5) Core ideas

- Claim:** an atomic assertion in the answer that can be verified.
- Retrieval Context:** the set of passages/pages provided to the model.
- Support:** the claim is explicitly or clearly implied by the context.
- Contradiction:** the claim conflicts with what the context states.
- Score:** proportion of claims that are supported (or a weighted variant).

6) QA Example

Context says: "Refunds within 30 days."
Answer adds: "We also offer free return shipping."
"Free return shipping" is not in the context, so that claim is unfaithful.

7) Distinctions

- Faithfulness:** checks answer vs. retrieved context (are claims supported / contradicted?).
- Answer Relevancy:** checks answer vs. user input (did it address the user's question?).
- Retrieval Metrics:** assess the retrieved/selected context (coverage, ranking quality, etc.).

8) Tester checks

- Support for each claim.
- Contradictions with context.
- Plausible details not present in context (unsupported).
- Prompt grounding (is the model instructed to use only context?).
- Traceability: can you point to the exact evidence (page/line/section)?

9) Common mistakes

- Calling every retrieval failure a faithfulness failure.
- Rewarding external facts that are absent from the required context.
- Checking tone or style instead of factual support.
- Assuming citations alone prove support.

10) Memory



Faithfulness =
answer claims must
match retrieved
evidence.

Groundedness

1) Definition



Groundedness measures how well a generated response is supported by the provided context or trusted source material.

A claim can be true in the real world yet still be ungrounded for a context-bound task if the supplied evidence does not support it.

2) Why it matters

- Central to RAG and enterprise assistants required to answer from approved sources.
- Helps detect fabricated or unverifiable claims.

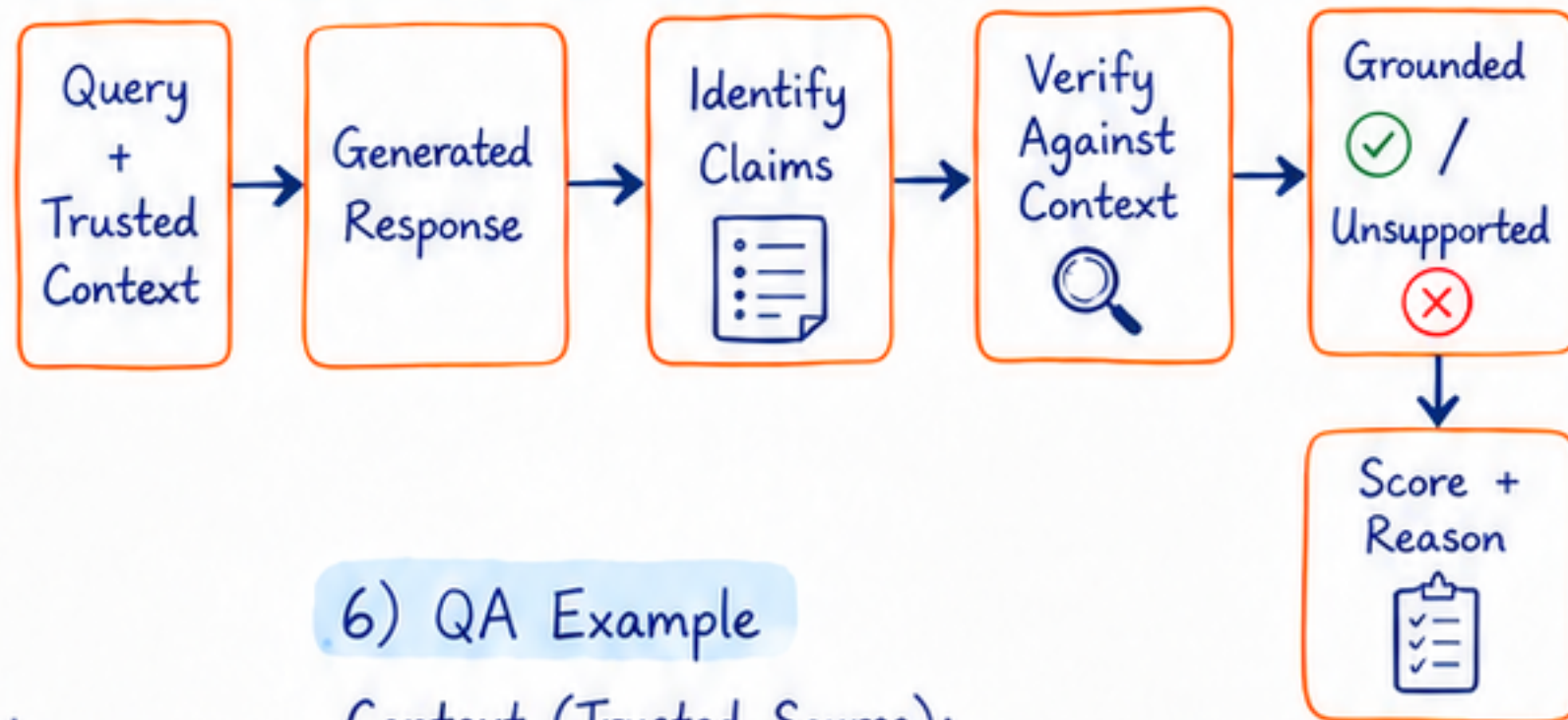


3) Mental model



An anchor keeps a boat tied to the seabed; grounded answers stay anchored to the supplied evidence.

4) Flow (Evaluation Process)



5) Core ideas

- Context - the provided source material.
- Claim - an atomic statement in the response.
- Evidence - specific text that supports a claim.
- Support - claim is entailed by the context.
- Score - degree of support (e.g., 0 to 1 per claim).

6) QA Example

Context (Trusted Source):

"Premium support is available Monday - Friday."

Model Response:

"Premium support is available 24/7."

Not Grounded.

Claim "24/7" is not supported by the context.

7) Distinctions (Do not confues)



Answer Relevancy asks: "Did it answer the question?"



Groundedness asks: "Is it supported by the given source?"



Faithfulness is closely related in RAG frameworks, though exact scoring implementations differ.

8) Tester Checks

- Traceability of claims to specific source text.
- External facts used without permission or instruction.
- Behavior when evidence is missing or insufficient.
- Citation presence and quality (specific and accurate).
- Regression after retrieval or prompt changes.

9) Common Mistakes

- Treating groundedness and relevancy as the same.
- Assuming true = grounded.
- Checking citation presence only, not support.
- Using a single quality metric for release decisions.

10) Memory



Groundedness = stay anchored to the provided evidence.

12) Keep in Mind



Accuracy: distinguish groundedness, faithfulness, and relevance carefully.



Never treat an evaluator as an infallible truth.



Use evaluators as signals; combine with judgment, tests, and user feedback.