EVALUATING AI AGENT

AI AGENTS AND HOW TO PROPERLY EVALUATE THEM

Two Layers of Evaluating LLM Agents

Model Evaluation

(Testing the LLM Alone)

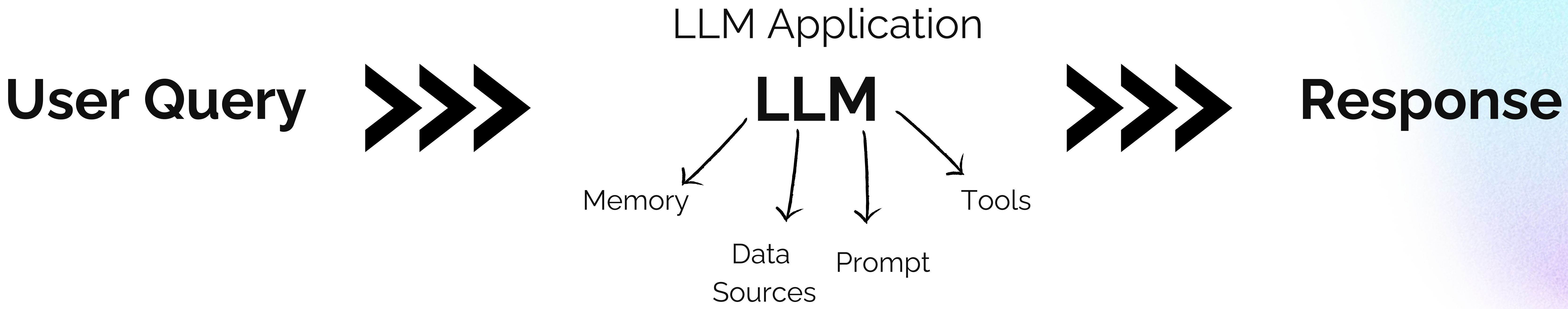
- Check if the LLM understands language well.
- Use benchmark datasets like MMLU (questions) and HumanEval (coding tasks).
- Focus is only on the LLM itself.

System Evaluation

(Testing the Full App)

- Check if the whole app (LLM + other parts) works well for real needs.
- Use custom or real-world data for testing.
- Focus is on the full user experience, not just the LLM.

From Query to Response



Difference Between

Traditional Software Testing

LLM Software Testing

Evolution of Testing

Traditional Software Testing:

- **Unit Testing:** Validate individual modules or components separately.
- **Integration Testing**: Ensure multiple components interact and function together as expected.

Testing with LLMs:

- **Handling Variability**: LLM outputs are non-deterministic — the same input can produce different results.
- **Task-Centric Evaluation:** Test how well the system handles real user tasks and requirements.
- **Output Quality Check**: Assess relevance, coherence, and usefulness of the model's responses.

Types of Evaluation For LLM System

- Hallucinations
- Retrieval relevance
- Q&A on retrieved data
- Toxicity
- Summarization performance
- Code writing correctness and readability

What Are Agents?

Agents are intelligent software systems that act on behalf of users by reasoning through tasks and making decisions. Unlike fixed scripts, agents dynamically analyze problems and determine the best action to take.

The agent typically does three things:

- **Reasoning:** Uses AI (like LLMs) to understand and think about the user's request.
- **Routing:** Figures out which tool or service to use to solve the user's request.
- **Action:** Executes the task, like calling an API, running a tool, or making another LLM call.

Agent Use Cases

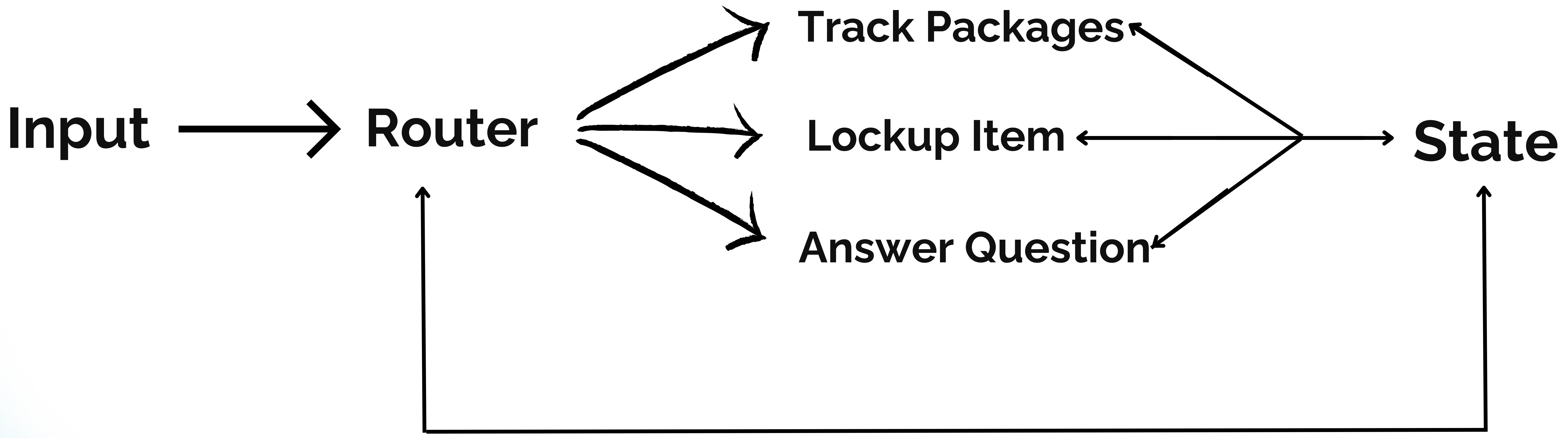
1) **Customer support automation**: Handle user queries, complaints, and FAQs instantly.

2) **Task automation agents - **Perform repetitive tasks like data updates, reporting, or notifications automatically.

3) **Data entry and processing bots** - Extract, clean, and organize information from documents, emails, or forms.

4) **Intelligent scheduling assistants** - Manage calendars, book meetings, and optimize schedules based on user preferences.

Main Components



TRACING AI AGENTS

Evaluating AI Agents Through Observability

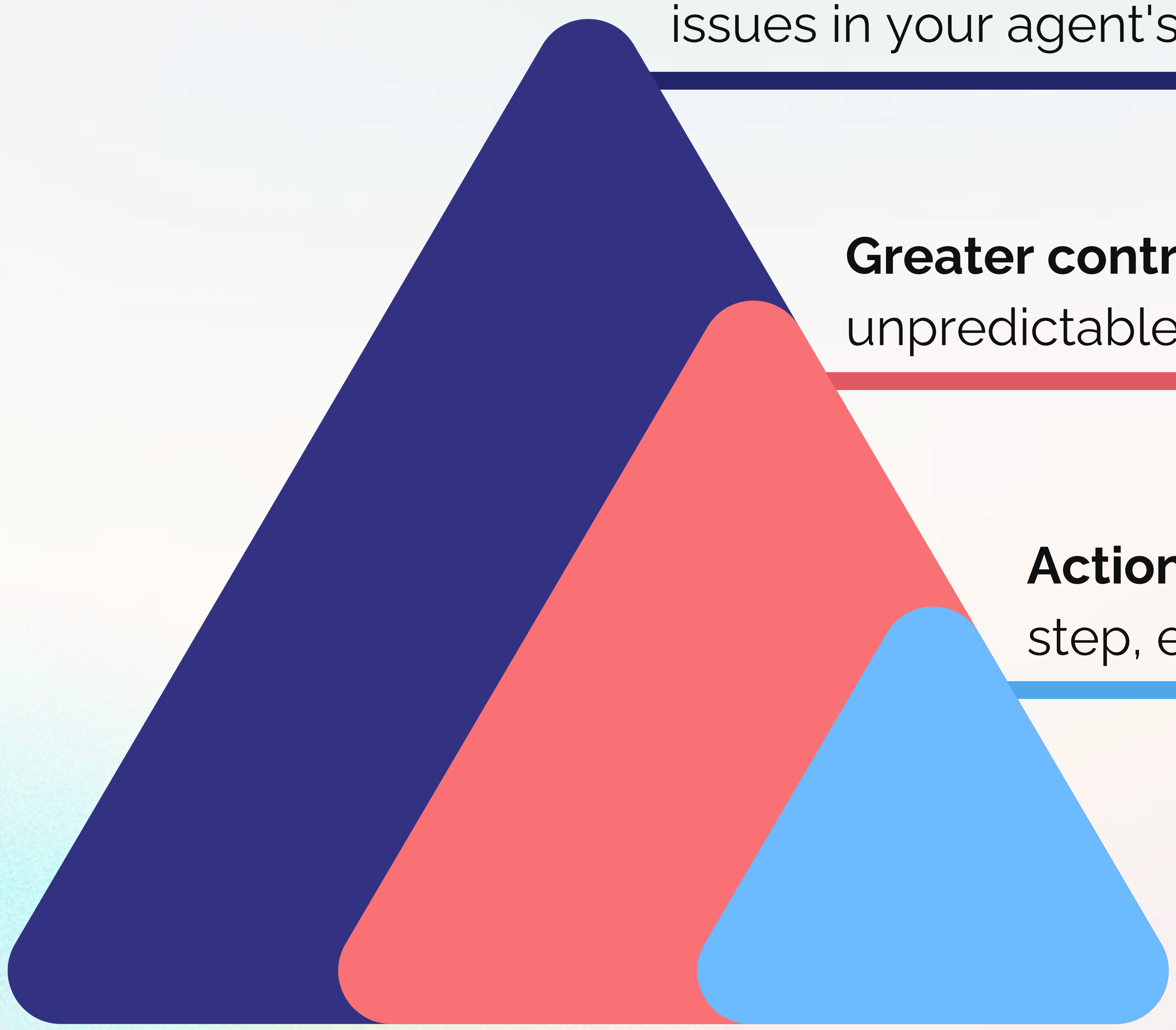
Observability enables deep insight into each component of an AI agent's behavior — from prompt construction to final response.

Why It Matters: To effectively evaluate AI agents, we must monitor how inputs flow through the system and how decisions are made at every stage.

Traces -
Track the complete path of a request across tools and reasoning steps. Useful for debugging and analysis.

Spans -
Capture individual operations (like retrieval or LLM calls) within a trace for detailed evaluation.

Benefits of Observability



Faster debugging – Quickly identify and fix issues in your agent's workflow.

Greater control – Understand and manage the unpredictable behavior of LLMs more effectively.

Actionable insights – Access detailed logs for each step, enabling precise performance evaluations.

CHOOSING THE RIGHT EVALUATOR

Types of Evaluation For LLM System

Code Based Evals

LLM -as-a-Judge- Evals

Human Evals

✅ Code-Based Evaluators

Automated scripts that compare outputs to expected results or compute performance metrics.

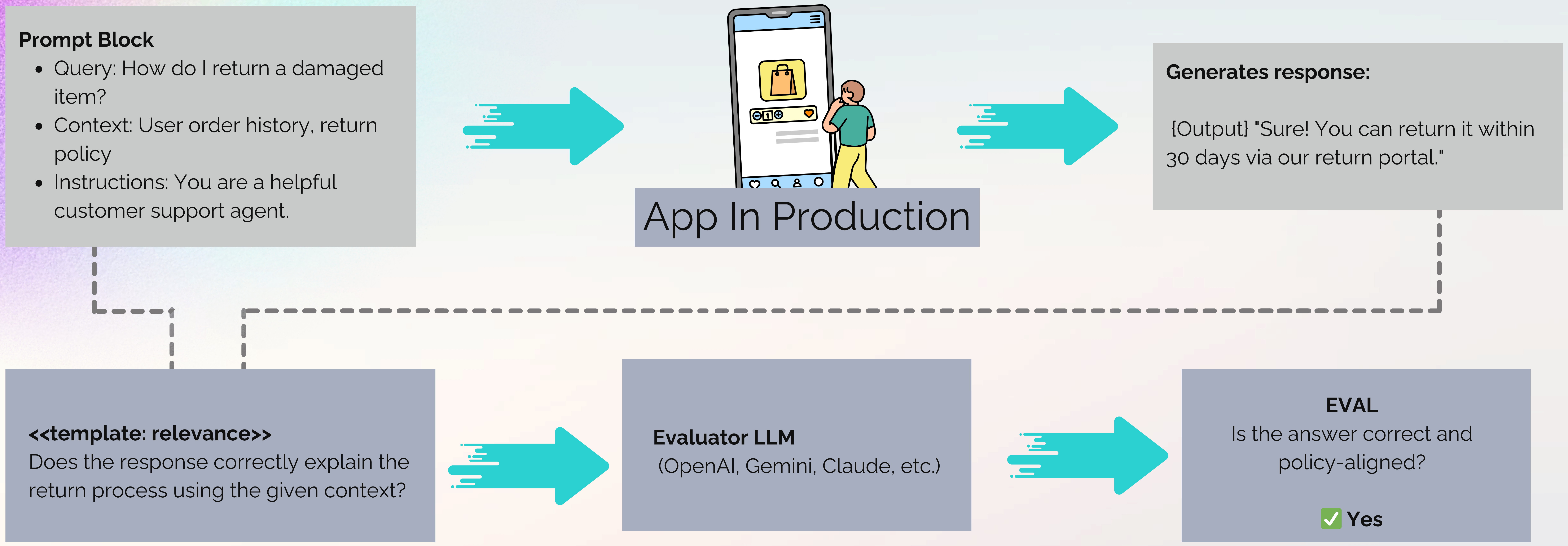
They typically include checks like:

- Regex Matches — Ensure outputs follow a pattern (e.g., only numbers).
- JSON Parsing — Verify if responses are valid and correctly structured as JSON.
- Keyword Checks — Confirm if outputs contain required terms (e.g., a competitor's name).

Compare outputs — against expected results using:

- Direct match
- Cosine similarity or distance.

🤖 LLM-as-a-Judge Evaluators



Prompt Block

- Query: How do I return a damaged item?
- Context: User order history, return policy
- Instructions: You are a helpful customer support agent.

App In Production

Generates response:

{Output} "Sure! You can return it within 30 days via our return portal."

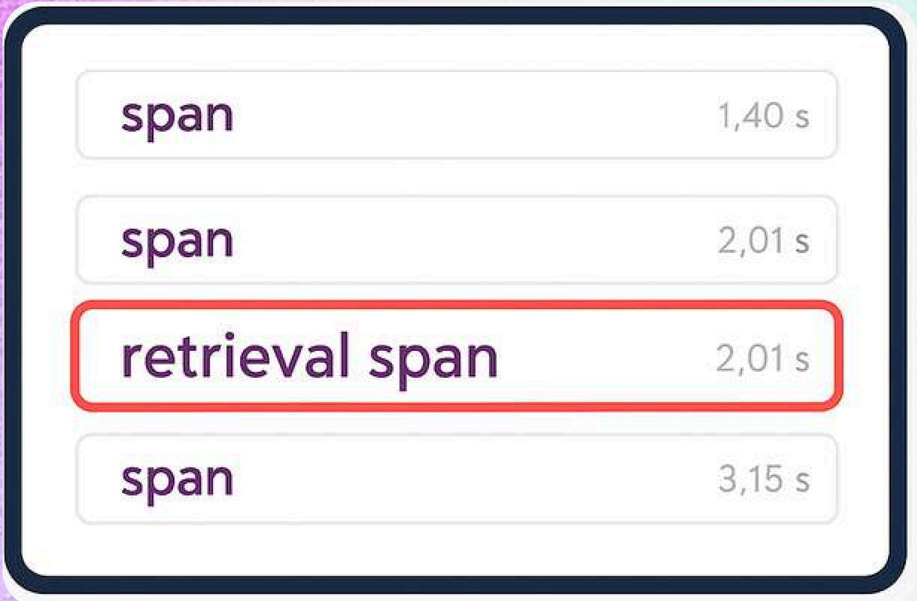
<<template: relevance>>
Does the response correctly explain the return process using the given context?

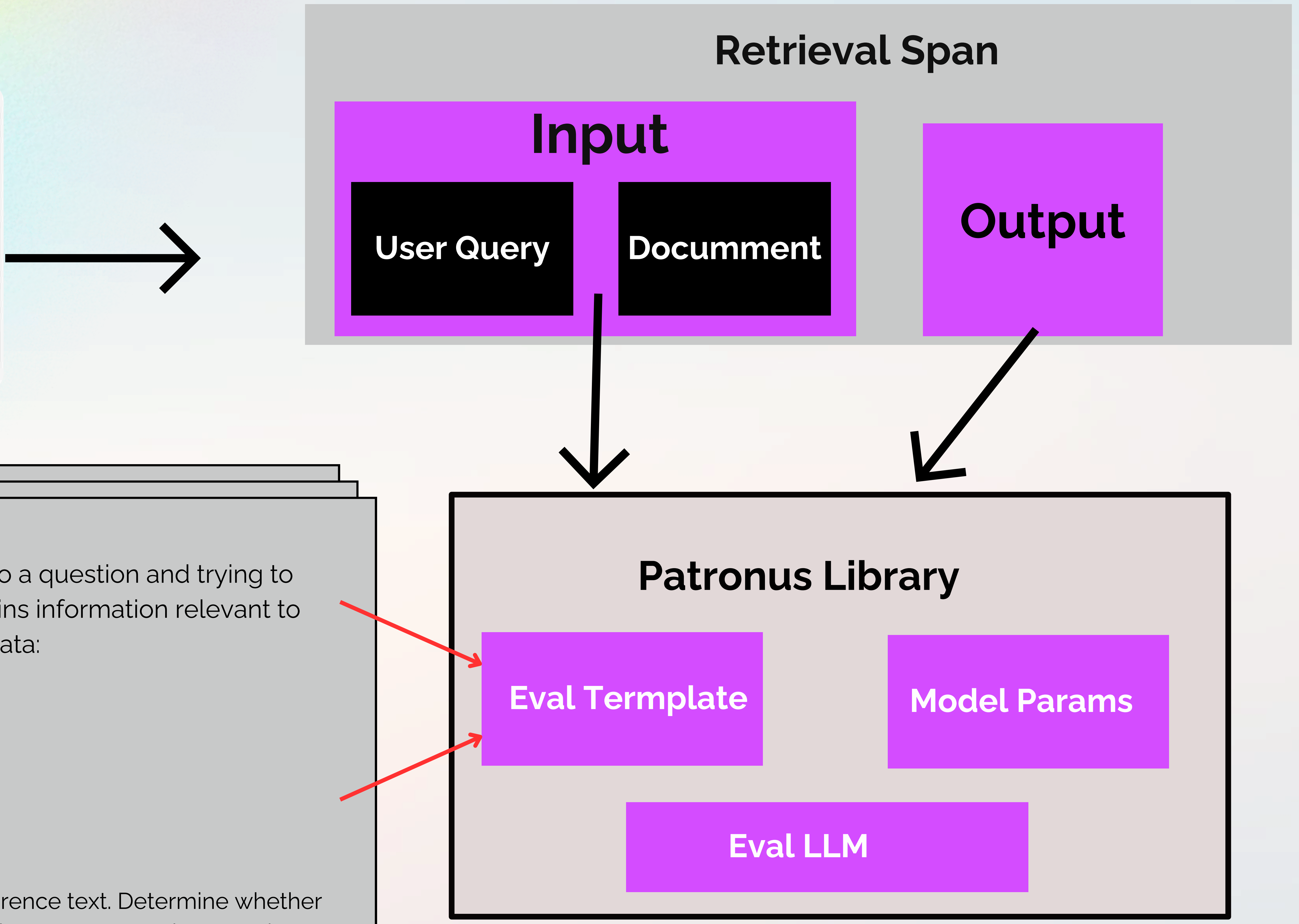
Evaluator LLM
(OpenAI, Gemini, Claude, etc.)

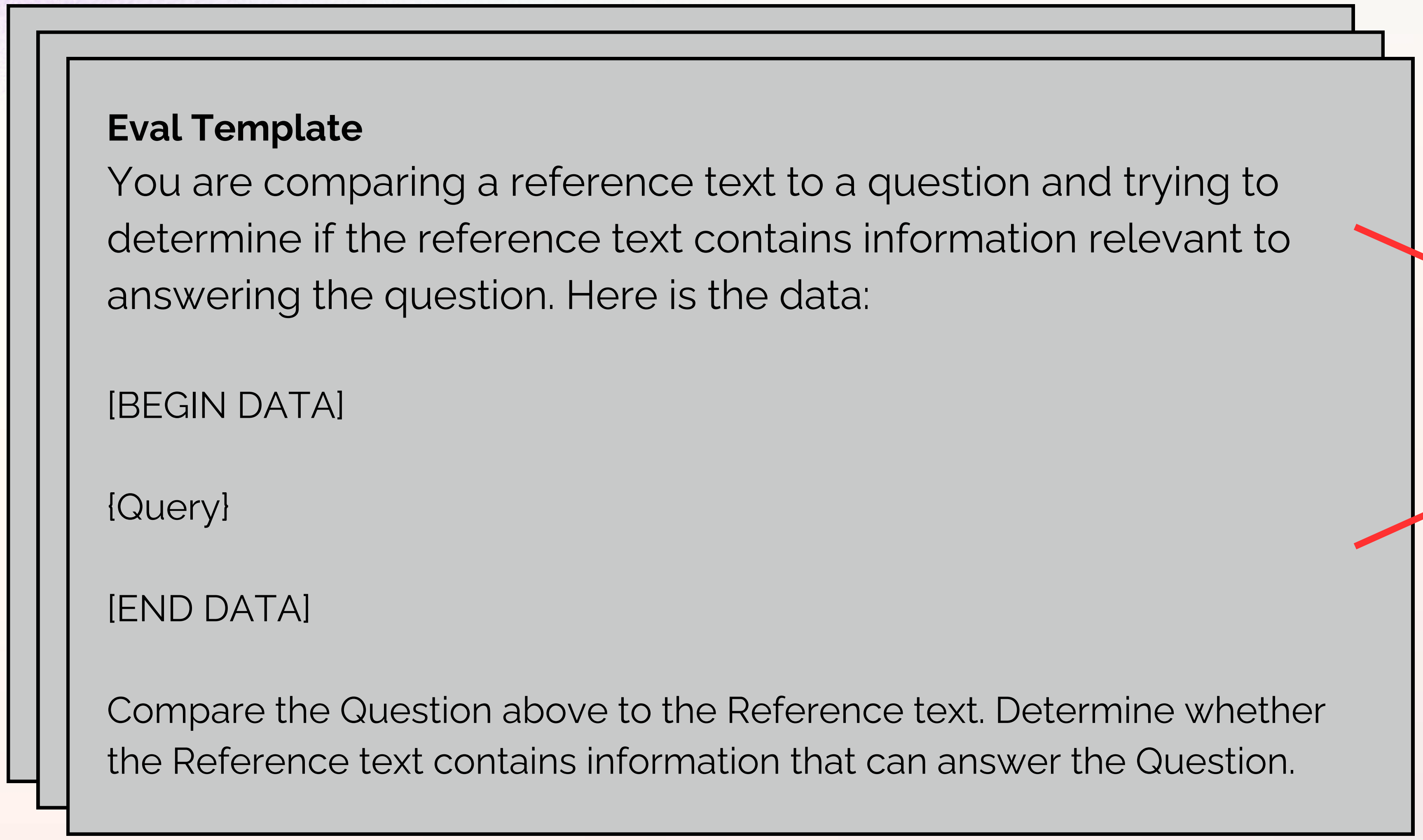
EVAL
Is the answer correct and policy-aligned?

✅ **Yes**

LLM-as-a-Judge Example







Eval Template
You are comparing a reference text to a question and trying to determine if the reference text contains information relevant to answering the question. Here is the data:

[BEGIN DATA]

{Query}

[END DATA]

Compare the Question above to the Reference text. Determine whether the Reference text contains information that can answer the Question.

Key Principles for Using LLM-as-a-Judge

Only top-tier models tend to closely match human evaluations.

No LLM judge is flawless—some level of inaccuracy is expected.

Well-crafted evaluation prompts can significantly improve reliability.

Use clear, binary labels (e.g., “correct” / “incorrect”) instead of vague or scaled scores.

👥Annotations

Add human evaluation labels to your traces to better understand and improve your system's responses.

Ways to collect annotations:

- ✅ Use annotation queues and human labelers to manually review outputs
- 👍👎 Collect thumbs-up/thumbs-down feedback directly from end users

Choosing the Right Evaluation Method

	Non-deterministic	Deterministic
Adaptable – suited for nuanced or subjective evaluations	LLM-as-a-judge	Human Labels
Strict – requires clear, measurable criteria		Code-based evals

Breaking Down Evaluation Targets

1. **Router Decisions -** Evaluating how well the system selects the right function and extracts relevant parameters

2. **Skills or Tool Usage -** These can typically be assessed using existing LLM evaluation techniques.

3. **End-to-End Path -** The most difficult aspect to evaluate reliably and at scale

Two Ways to Evaluate a Router

Parameter Identification – Did it extract the appropriate parameters needed to call that function?

Function Selection Accuracy – Did the router choose the correct function to invoke based on the query?

Evaluating a Router using LLM-as-a-Judge

```
TOOL_CALLING_PROMPT_TEMPLATE = """
You are an evaluation assistant evaluating questions and tool calls to determine whether the tool called would answer the question. The tool calls have been generated by a separate agent, and chosen from the list of tools provided below. It is your job to decide whether that agent chose the right tool to call.

[BEGIN DATA]
************
[Question]: {question}
************
[Tool Called]: {tool_call}
[END DATA]

Your response must be single word, either "correct" or "incorrect", and should not contain any text or characters aside from that word.
"incorrect" means that the chosen tool would not answer the question, the tool includes information that is not presented in the question, or that the tool signature includes parameter values that don't match the formats specified in the tool signatures below.
"correct" means the correct tool call was chosen, the correct parameters were extracted from the question, the tool call generated is runnable and correct, and that no outside information not present in the question was used in the generated question.
[Tool Definitions]: {tool_definitions}
"""
```

Evaluating a Router using LLM-as-a-Judge

User: Hi, can you help check on the status of my order? #1234

Agent: Definitely!

- {tool_call: "order_status_check(order_number=1234)"}
- {role: "tool", content: "status=shipped"}
- Your order has been shipped!

User: When will it arrive?

Agent: Let me check.

- {tool_call: "shipping_status_check(shipping_tracking_id=1234)"}

Evaluating Skills

Skills can be evaluated using standard LLM or code-based evals:

- 🔴 Relevance
- 🔴 Hallucination
- 🔴 Question and answer correctness
- 🔴 Generated code readability
- 🔴 Summarization
- 🔵 Regex
- 🔵 JSON parseable
- ⚫ ...

EVALUATING THE AGENT’S PATH, NOT JUST THE OUTPUT

Agent Trajectory

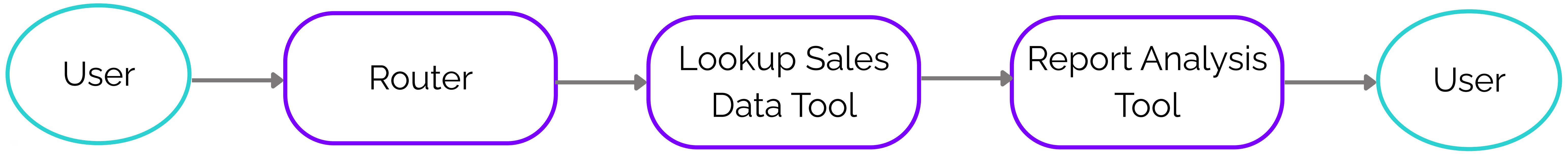
An agent trajectory is the step-by-step path an AI agent follows to answer a user query, including the tools it uses and the decisions it makes along the way.

Why it Matters
✅ Reveals how the agent thinks
✅ Helps optimize performance
✅ Useful for debugging and improvement

Agent Trajectory

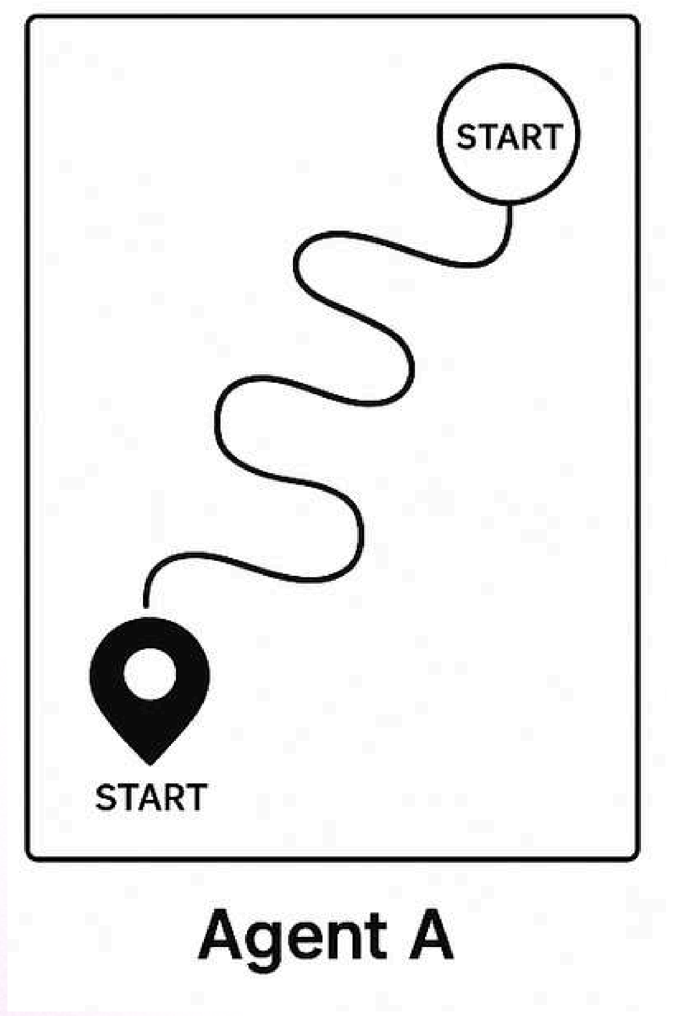
Example

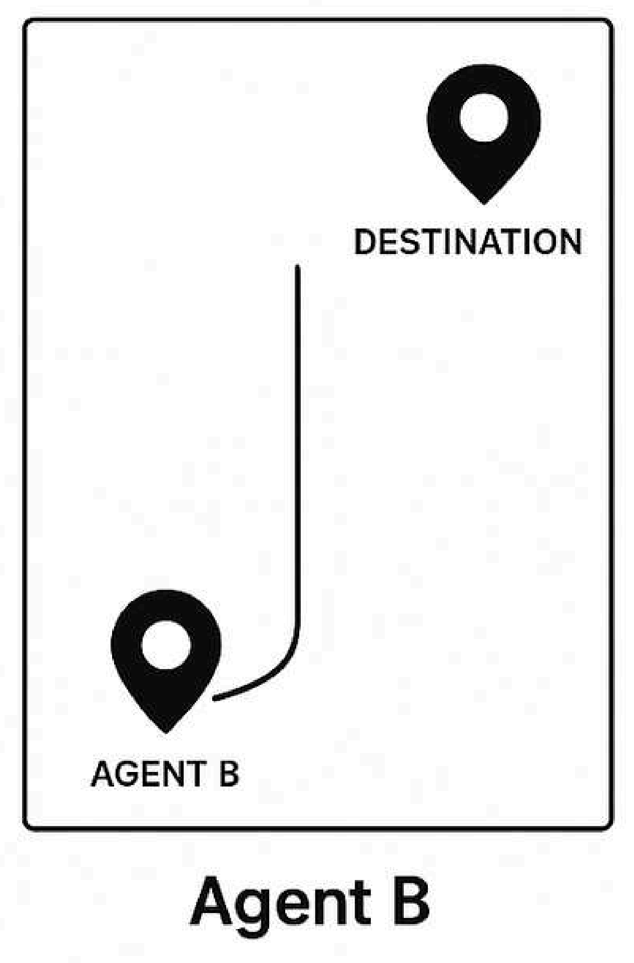
User Prompt:
"Summarize the latest earnings report."



Agent Trajectory

Why does Agent Trajectory matter if the output is correct?

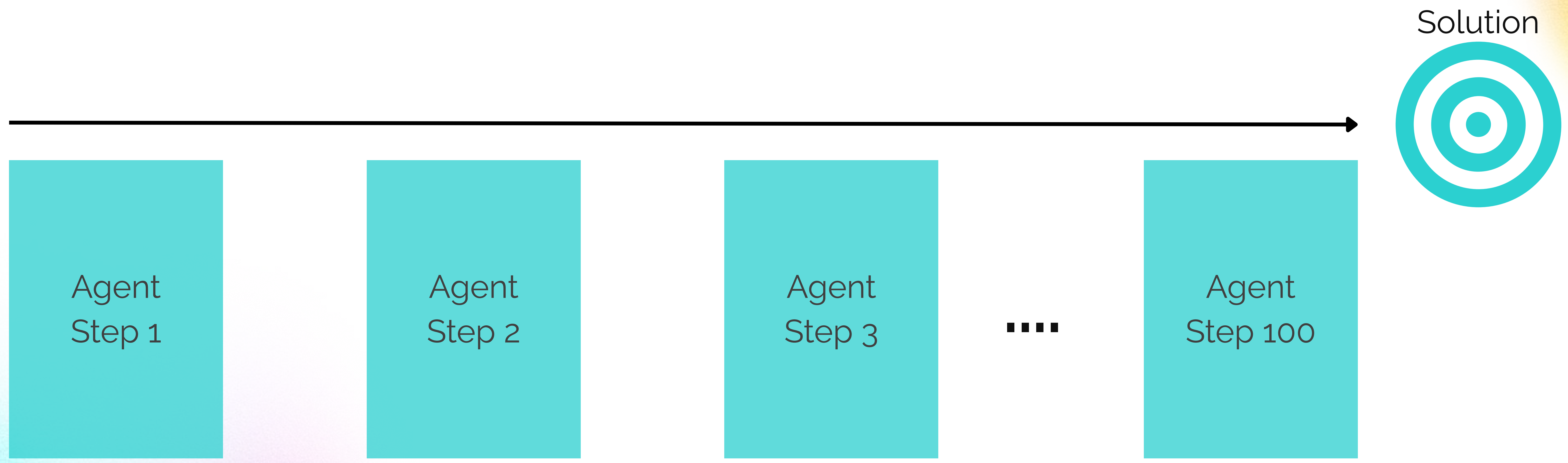




Even if the destination is the same, an inefficient trajectory wastes time, consumes more resources, and makes your agent harder to debug, scale, and trust.

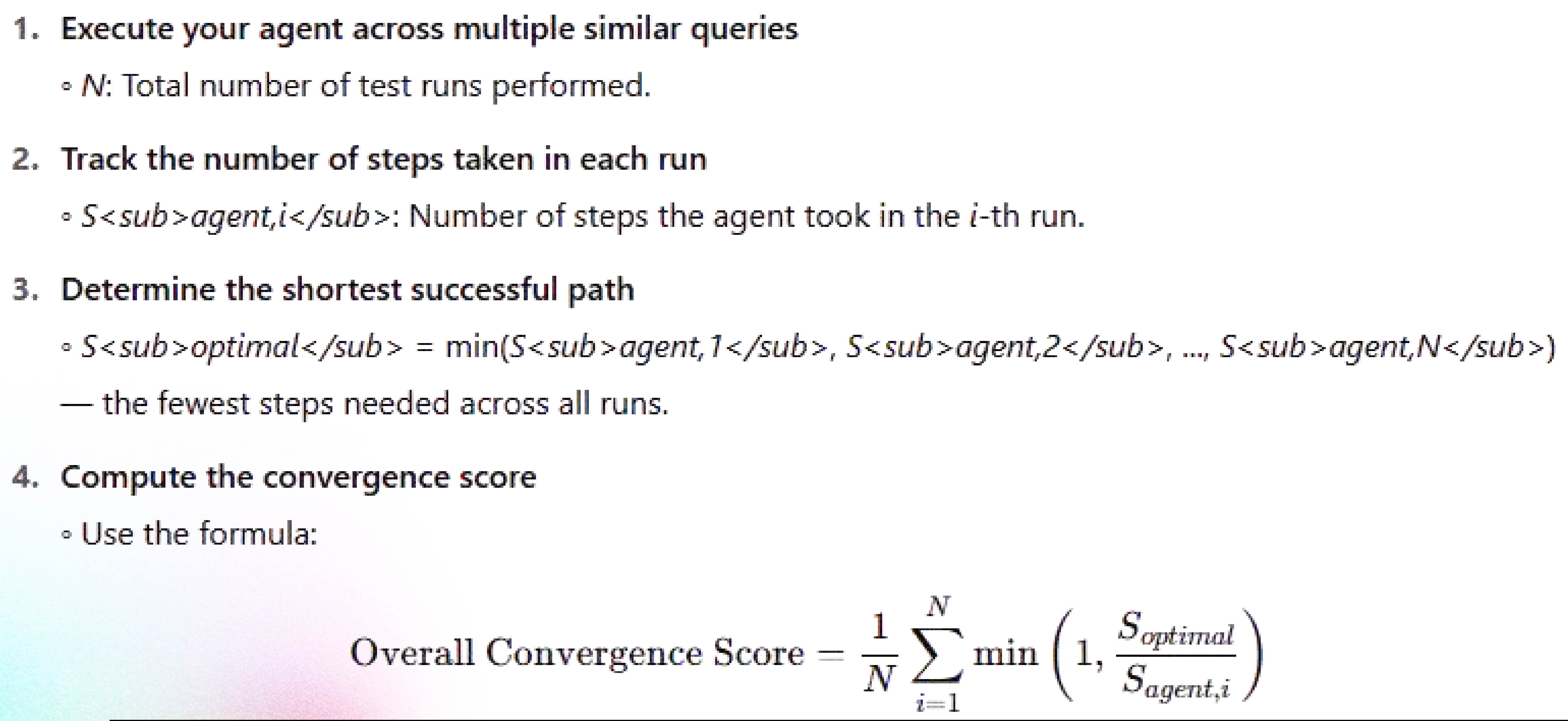
Convergence

Convergence measures how efficiently an agent reaches the correct answer by following the fewest and most relevant steps possible.
In simple terms, it measures how quickly your agent reaches a solution.



Convergence

How do you test for convergence?



1. **Execute your agent across multiple similar queries**
   - *N*: Total number of test runs performed.
2. **Track the number of steps taken in each run**
   - *S_{agent,i}*: Number of steps the agent took in the *i*-th run.
3. **Determine the shortest successful path**
   - *S_{optimal}* = min(*S_{agent,1}*, *S_{agent,2}*, ..., *S_{agent,N}*)

     — the fewest steps needed across all runs.
4. **Compute the convergence score**
   - Use the formula:

$$\text{Overall Convergence Score} = \frac{1}{N} \sum_{i=1}^{N} \min\left(1, \frac{S_{optimal}}{S_{agent,i}}\right)$$

Convergence

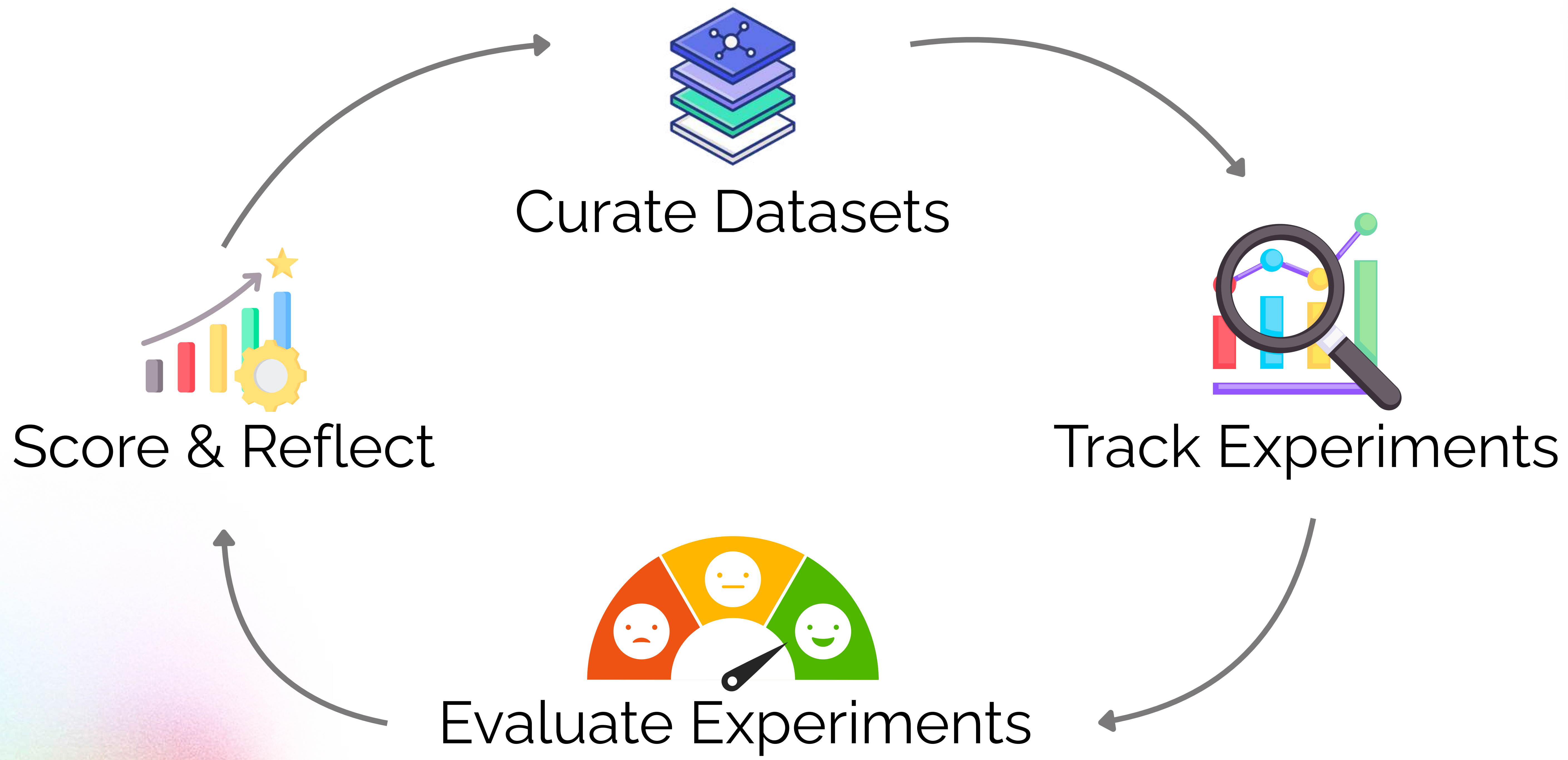
Another way for testing convergence

- What percentage of the time is your agent taking the optimal path, for a given set of inputs?
- Convergence score of 1 means that agent is taking the optimal path 100% of the time

$$\text{Overall Convergence Score} = \frac{1}{N}\sum_{i=1}^{N}\min\left(1, \frac{S_{optimal}}{S_{agent,i}}\right)$$

COMBINE EVALUATORS TO EVOLVE YOUR AGENT SMARTER

Evaluation-led Development



Evaluation-led Development

Curate Datasets

Build a Dataset of Test Cases

- Start with 25+ representative inputs for your agent
- Make it comprehensive — cover edge cases, typical flows, and variations
- Use data from live agent runs or pre-constructed scenarios
- Link each input with its expected output for evaluation

Note: LLM-as-a-Judge can work even without expected outputs

Evaluation-led Development

Track Experiments

Test and validate any proposed updates to your agent by running evaluations on:

- Model changes
- Prompt iterations
- Skill structure
- Router logic
- Tool definitions

Use experiments as a structured method to run your curated test cases through the agent and capture the resulting outputs for comparison.

Evaluation-led Development

Evaluate Experiments

To assess your experiment results, continue using the evaluators introduced in earlier lessons:

Code-based

- Compare against ground truth
- Generated code is runnable
- Convergence

LLM-as-a-Judge

- Function calling
- Analysis clarity
- Entity correctness

Evaluation-led Development

Evaluate Experiments

Component of your agent	Example test case	Experiment	Evaluations
Router	{'input': 'Which car model had the highest sales in 2024?', 'expected_output': 'database_lookup'}	Try out different ways of describing the tool's function	1. Compare output with the ground truth (code-based) 2. Assess function calling using LLM-as-a-Judge
Database lookup tool	{'input': 'List all electric car models sold in California in 2024','expected_output': 'database_lookup'}	Explore variations in SQL prompt generation	1. Code-based accuracy check against expected output
Database analysis tool	{ 'input': {'role':'user','content':'What was the overall market trend for hybrid cars in 2024?'}, 'role': 'tool content': 'role': 'tool', 'content': 'Model_Name, Market_Share, Growth_Rate, ...'}}	Compare performance across multiple LLM models	1. Evaluate clarity of analysis 2. Verify correctness of extracted entities

Evaluation-led Development

Score & Reflect

Screenshot of comparing the experiments score

IMPROVING YOUR LLM AS A JUDGE

Why validate with LLM-as-a-Judge, if Code-based works well?

✅ Catch what code misses

✅ Double-check evaluation logic

✅ Cover failures in edge cases

Improving your LLM-as-a-Judge

Using experiments ensures LLM judges make more reliable decisions.

LLM Judge	Example Test Case	Experiment	Evaluations
Function Execution Judge	{'input': 'Retrieve the customer purchase history for 2022', 'expected_output': 'database_lookup'} {'input': 'Find employee attendance records for March', 'expected_output': 'database_lookup'}	Testing variations of LLM-as-a-Judge prompts	1. Automated code-based validation against ground truth
Analysis Interpretatio n Judge	{'input': 'Summarize the key drivers behind revenue growth in Q4.', 'expected_output': 'database_lookup'} {'input': 'Identify the main reasons for customer churn last year.', 'expected_output': 'database_lookup'}	Comparing different models for evaluation prompts	1. Ground truth comparison 2. Clear evaluation criteria for outputs

YOUR AGENT IS IN PRODUCTION... NOW WHAT?

Managing Agent Performance in Production

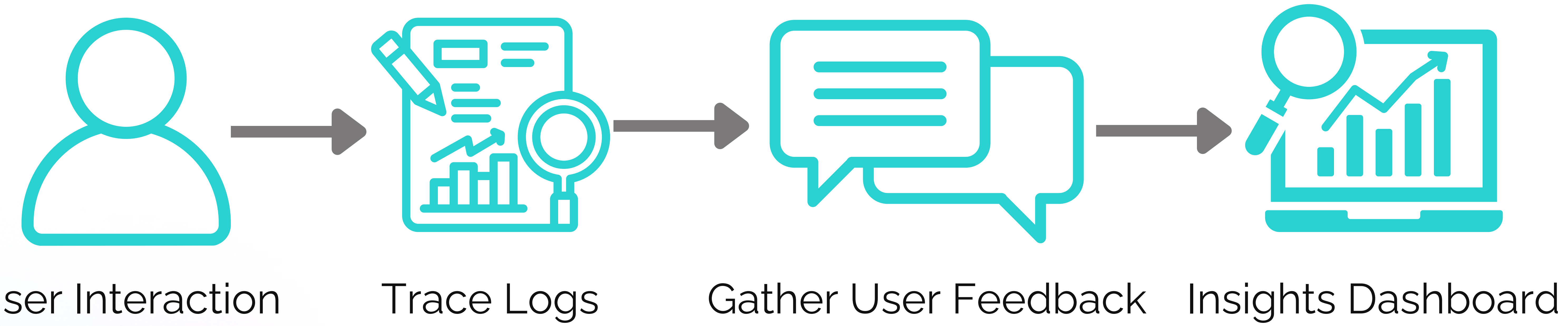
🔍 Observability & Feedback

📊 Monitor Metrics

🔄 Continuous Improvement

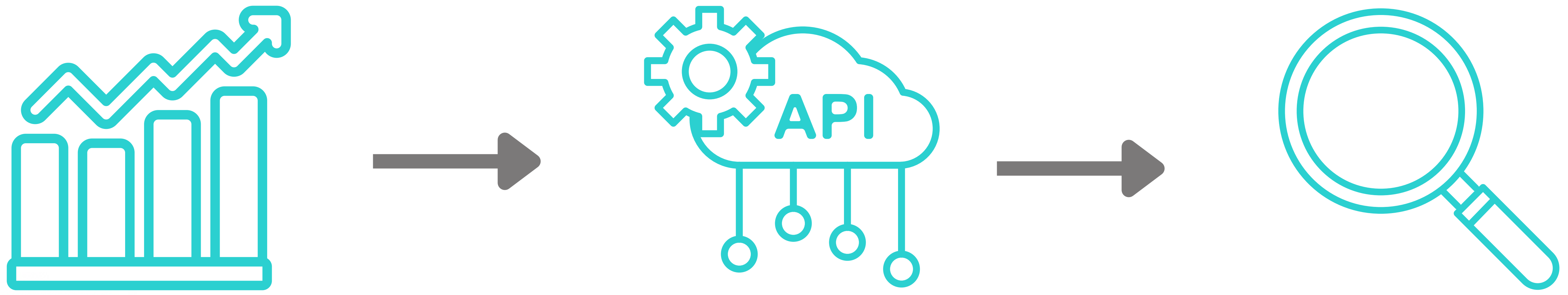
Managing Agent Performance in Production

🔍 Observability & Feedback



Managing Agent Performance in Production

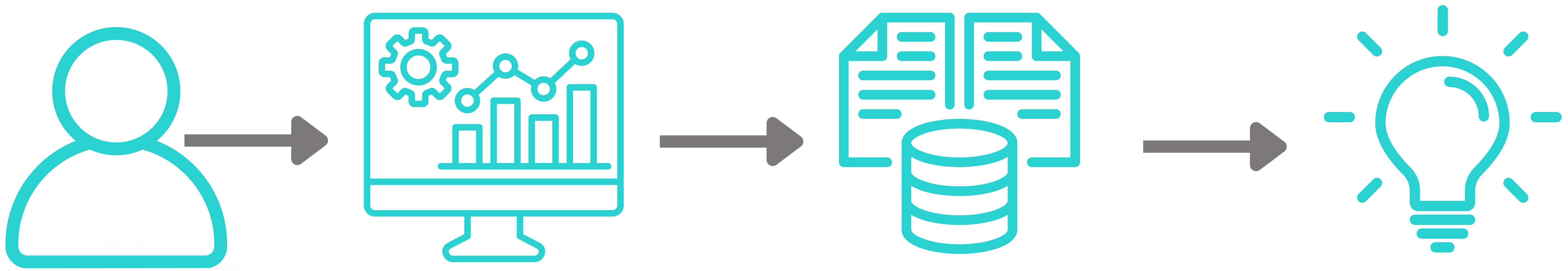
📊 Monitor Metrics



Track Efficiency → Watch API Latency and Cost → Identify Improvements

Managing Agent Performance in Production

🔄 Continuous Improvement



THANK YOU