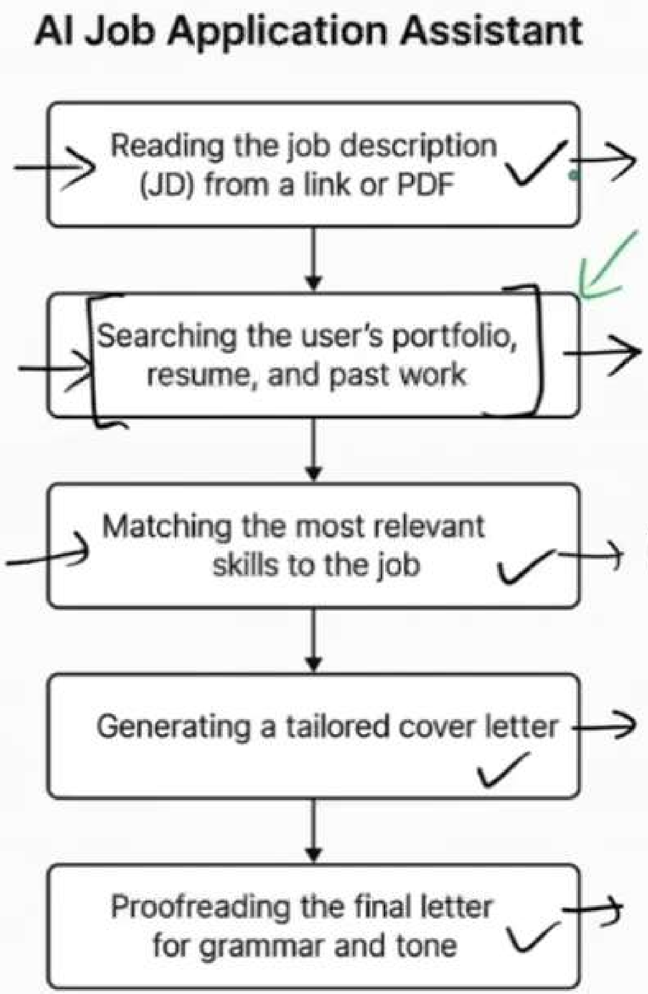
Why do we need LangSmith

Friday, 15 August 2025 8:05 AM

complex LLM workflow



→ fetch

→ matching

$\boxed{\text{Startup}}$ → problem J

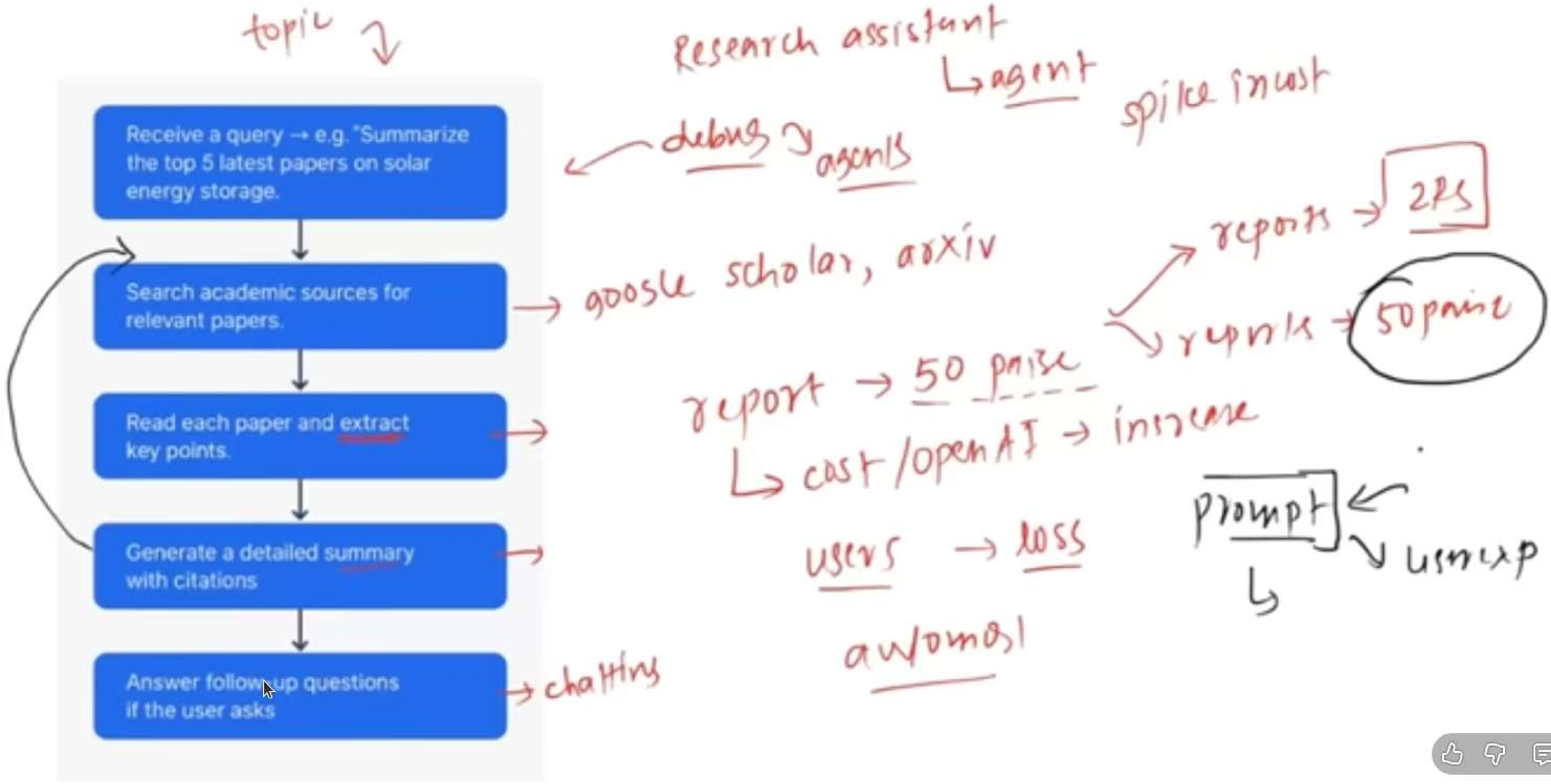
job → naukri ↓ filter ↓ jd → resume, cover letter → apply

LLM app

10 bar

latency → 2-min

slow → 7-10 mins


topic
Receive a query → e.g. "Summarize the top 5 latest papers on solar energy storage.
Search academic sources for relevant papers.
Read each paper and extract key points.
Generate a detailed summary with citations
Answer follow-up questions if the user asks
Research assistant
↳ agent
spike in cost
debug → agents
→ google scholar, arxiv
reports → 2Rs
reports → 50 paise
report → 50 paise
↳ cost/openAI → increase
users → loss
anomal
prompt
user exp
→ chatting

conversational response

•

Observability is the ability to understand a system's internal state by examining its external outputs, like logs, metrics, and traces.

It allows you to diagnose issues, understand performance, and improve reliability by analyzing data generated by the system.

Essentially, it's about being able to answer "why" something is happening within a system, even if you didn't anticipate the problem.

What is LangSmith

Friday, 15 August 2025 8:11 AM

LangSmith is a unified observability & evaluation platform where teams can debug, test, and monitor AI app performance.

What does LangSmith Trace?

Friday, 15 August 2025 8:11 AM

1. Input and Output
2. All the intermediate steps
3. Latency
4. Token usage
5. Cost
6. Error
7. Tags
8. Metadata
9. Feedback

Code Examples

Friday, 15 August 2025 8:11 AM

RAG apps have **two big failure modes**:

1. **Retriever errors** – wrong / irrelevant docs retrieved.
2. **Generator errors** – model hallucinates or misuses context.

In production, it's often unclear *where the failure happened*. Was the retriever bad, or did the LLM ignore the docs?

- LangSmith automatically records:
    - User query
    - Retrieved documents
    - LLM prompt (with inserted docs)
    - LLM response

- Every graph execution can be logged in LangSmith as a **trace**.
- Each **node** (e.g., retriever, LLM, tool call, subgraph) becomes a **run** inside the trace.
- You can visualize the **path taken**:
  - `START → Retriever → Reranker → LLM Answer → END`
- If a workflow branches (conditional/parallel/subgraph), LangSmith captures which path was executed.

Other Features of LangSmith

$\boxed{57}$

multiple traces

monitor

observ

↓

LLM → exe

↓

trace →

Monitoring and Alerting

What it does:

Monitoring in LangSmith looks across **many traces at once** to track the overall health of your LLM system. It aggregates key metrics like **latency** (**P50, P95, P99**), **token usage**, **cost**, **error rates**, and **success rates**. You can set up **alerts** to notify you when these metrics drift outside acceptable ranges (e.g., a spike in latency, higher error rates, or unexpected cost growth).

Why it matters:

In production, issues often appear first as **patterns across multiple runs** rather than in a single trace. Monitoring helps you catch these early signals before they impact users at scale. Instead of waiting for customer complaints, you're proactively alerted when performance degrades or costs spike, enabling faster response and more reliable applications.

Evaluation

What it does:

Evaluation in LangSmith helps you **systematically measure the quality of your LLM outputs**. You can run tests against **gold-standard datasets** or apply **custom evaluation metrics** such as faithfulness, relevance, or completeness. LangSmith supports multiple approaches: automated scoring with **LLM-as-a-judge**, **semantic similarity checks**, or even **custom Python evaluators**. Evaluations can be run both

Monitoring and Alerting

What it does:

Monitoring in LangSmith looks across **many traces at once** to track the overall health of your LLM system. It aggregates key metrics like **latency (P50, P95, P99), token usage, cost, error rates, and success rates**. You can set up **alerts** to notify you when these metrics drift outside acceptable ranges (e.g., a spike in latency, higher error rates, or unexpected cost growth).

Why it matters:

In production, issues often appear first as **patterns across multiple runs** rather than in a single trace. Monitoring helps you catch these early signals before they impact users at scale. Instead of waiting for customer complaints, you're proactively alerted when performance degrades or costs spike, enabling faster response and more reliable applications.

Evaluation

What it does:

Evaluation in LangSmith helps you **systematically measure the quality of your LLM outputs**. You can run tests against **gold-standard datasets** or apply **custom evaluation metrics** such as faithfulness, relevance, or completeness. LangSmith supports multiple approaches: automated scoring with **LLM-as-a-judge, semantic similarity checks**, or even **custom Python evaluators**. Evaluations can be run both **offline** (batch tests before deployment) and **online** (continuous checks on live traffic).

Why it matters:

LLM behavior can be unpredictable — a small change in prompts, models, or retrieval logic may improve some cases but break others. Evaluation provides an **objective, repeatable way to track performance over time**, ensuring that new versions are actually better and preventing regressions.

Example:

For a RAG chatbot, you might evaluate:

- **Faithfulness** → Are answers grounded in retrieved documents?
- **Relevance** → Did the response actually address the user's question?

By running the same dataset across GPT-4, Claude, and LLaMA, you can directly compare which model (or pipeline setup) performs best.

- **Relevance** → Did the response actually address the user's question?

By running the same dataset across GPT-4, Claude, and LLaMA, you can directly compare which model (or pipeline setup) performs best.

Prompt Experimentation

What it does:

Prompt Experimentation in LangSmith allows you to **systematically test and compare different prompt versions**. You can run **A/B tests** across prompts on the same dataset, track their performance against evaluation metrics, and record the outcomes. Results are stored over time, giving you a clear history of which prompt variations worked best and under what conditions.

Dataset Creation & Annotation

What it does:

- Provides tools to build datasets for evaluation and fine-tuning.
- Supports manual annotation (e.g., labeling whether an answer is correct).
- Stores datasets versioned for reuse across projects.

Why it matters:

High-quality datasets are critical for evaluation and feedback loops.

Example:

- Customer support: Build a dataset of **common questions + expected answers**.
- Use it to benchmark your RAG agent every time you change retrieval logic.

User Feedback Integration

What it does:

- Lets you capture **thumbs up/down, ratings, or structured feedback** from users in production.
- Feedback is logged alongside traces → tied to the exact prompt, model, and state.
- Supports bulk analysis of what users like/dislike.

Collaboration

What it does:

- Team members can view, share, and comment on traces, datasets, and evaluations.
- Provides a **web UI** where non-engineers (PMs, QA, annotators) can inspect and annotate runs.
- Enables **shared experiment dashboards**.