Agentic - AI
Langgraph

21 - 08 - 2025

→ AGENTIC - AI.
- → foundations
- → Langgraph fundamentals
- → Advance Langgraph
- → AI - Agent.
- → Agentic RAG
- → Productanization

Lecture - 01. →

Generative AI VS Agentic AI

→ refers to a class of AI models that can create new content - such as text, images, audio, code, video etc. that resembles human - created data.

Traditional AI → It is about finding patterns in data and giving predictors →

← Generative AI is about learning the distribution of data so that it can generate a new sample from it.

The best part of GenAI is that it mimic human. (their functions.)

Applications Areas →
- Creative and Business writing
- Software Development
- Customer Support *
- Education
- Designing

Problems →
- → Reactive
- → No memory.
- → Generic Advice
- → can't take Actions.

Normal GenAI Chatbots

So, the solution is :

→ RAG Chatbot. (Retrieval Argemented Generation)
By giving data from us, it store that and make at parametric knowledged.

So it solving a -
- Specific Advice

So, there is solution of rest problem:

↳ Tool Augement Chatbot (RAG + Tool)

It solve problems. →

- Specific Advice
- Can take action

[Natural active
No memory
Can't Adapt

So Now, most advance chatbot for these

↳ Agentic AI Chatbot (AI agents)

→ It is proactive that can plan and execute.
→ Obviously it has memory so, it can plan.
→ Specific Advice
→ It can take actions.
→ It's adaptable.

Gen AI is about creating content
Agentic AI is about solving a goal

Gen AI is reactive, Agentic AI is proactive (autonomous)

Gen AI is building block of Agentic AI.
(subset) (superset)

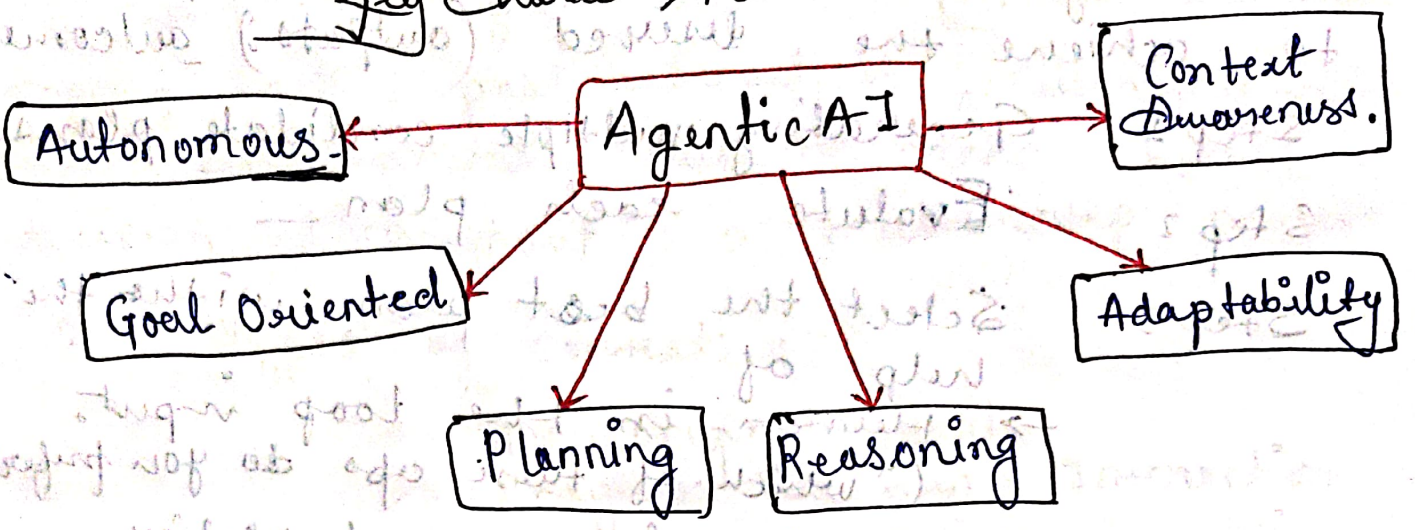
Capability Behavior *

Lecture - 02 What is Agentic AI.

→ It is a type of AI that can take up a task or goal from a user and then work toward completing it on its own, with minimal human guidance *.

→ It takes plan, take actions, adapts to changes, and seeks help only when necessary.

Key Characteristics →



Autonomy → Autonomy refers to the AI systems ability to make decision and take actions on its own to achieve a given goal, without needing step by step human instructions.

But it can controlled → otherwise it can be dangerous so the autonomy.

a. Permission Scope
b. Human in the Loop (HTL)
c. Override Controls
d. Guardails / Policies

Goal Oriented → It means that the AI system operates with a persistent objective in mind and continously directs its actions to achieve that objective, rather than just responding to isolated prompts.

1. Goals acts a compass for autonomy
2. Gals can work with constraints
3. Goal are stored in core memory.
4. Goals can be altered.

Planning → Planning is the agents ability to break down high level goal into a structured sequence of actions or subgoals and decide the best path to achieve the desired (outputs) outcome.

Step 1 · Generating multiple candidate plans

Step 2 · Evaluate each plan

Step 3 · Select the best plan with the help of
→ Human in the loop input.
("Which of these ops do you prefer")
→ A pre programmed policy.

Reasoning → It is the cognitive process through which an agentic AI system interprets information draws conclusions and make decisions — both while planning ahead and while executing actions in real time.

Reasoning during planning	during executing
1. Goal decomposition	1. Decision-making
2. Tool selection	2. HTL Handling
3. Resource estimation	3. Error Handling.

Adaptability → It's agents ability to modify its plan, strategies or actions in response to unexpected conditions — all while staying aligned with the goal.

1. Failures (Calender API)
2. External feedback (less no. of application)
3. Changing goals (Hiring a freelancer)

Context Awareness →

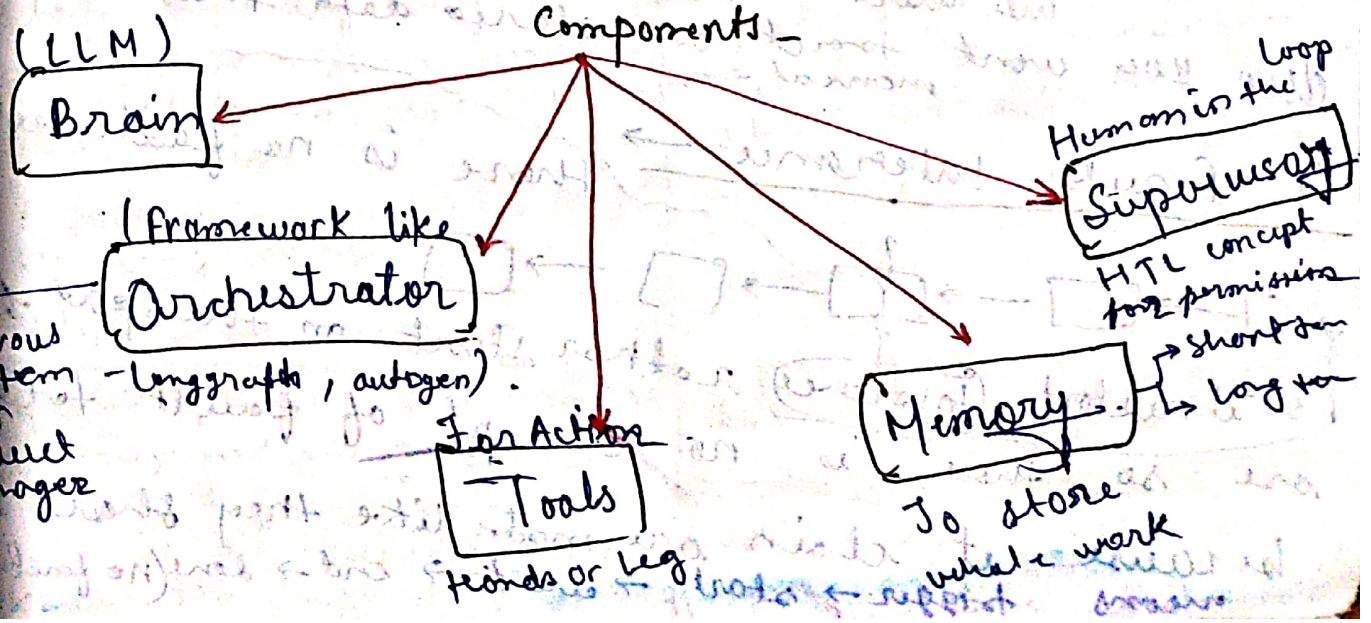
It's the agent's ability to understand, retain and utilize relevant info. from the ongoing task, past interactions, user preferences and environmental cues to make better decisions throughout a multi-step process.

[Environment ↔ Agent work in
Eg – Agent → play chess
so, Env → Chessboard]

1. Types of context →
   a. The original goal
   b. Progress till now + Interaction history
   c. Environment state.
   d. Tool response
   e. User specific preferences
   f. Policy or guardrails

2. Context implement through memory
3. Short term memory
4. Long term memory

Components of Agentic AI. [top 5]



lecture → 03

LangChain & LangGraph VS

alternates of LangGraph → Autogen, Crewai etc.

Why LangGraph → We create graphs of nodes and connect by its edges

Workflow → Static, predefined path

Agent → Dynamic, that creates their own path to execute process effici planning.

LangChains Challenges →

1. Contral Flow Complexity :
[No logic to write in langchain need to use python glue code] that effect maintenability
It only efficint for linear workflow not conditional or parallel
1) Conditional branch
2) loops
3) Jump
4) glue code

2. Handling State :
[State mean's set of datapoint and group of set of state is workflow]
It has only conversational memory. there is no storing or Tracking of data so, it is stateless. Think when you need to change in step then we were have to do manually - also you want track or stored data that is manual -

3. Fault tolerance → there is no fault tolerance

so it support tolerance rather stop on the spot fault are so there is no concept of fault tolerance because it chain are made like they short lived linear → start → work → end → done/no fault tolerance

Why LangGraph →

1. No (zero) glue code

2. Stateful → graph

State object
- Pydantic
- TypedDict [mutable is it]

3. check_pointer → to save/track (State)

there is possible Event_driven Execution in the workflow.

4. Fault Tolerance

is there →

retry logic (for short problem & faults)

recovery (when server is down or system issue & errors)

It is possible because of concept of check_pointers

also there [resume func^n] to resume the work where it stop previous by defining a state - then automatically identify state and also their problem

Fault tolerance is very high in LangGraph.

5. Human in the loop

LangGraph allows you to pause a execution indefinitely - for min, hours, days even - until human input is recieved [because it saves the graph state after each step]

Challenge 3 →

Event driven Execution

workflow will pause in conds and wait for external trigger to resume workflow

IR is not build for this LangChain is build for sequential.

Challenge 4 →

Fault Tolerance

Challenge 5 →

Human in the loop

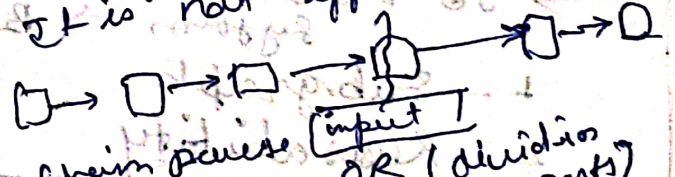
pauses in workflow for human permission and their aproval after it start work

But in LangChain ~~it is~~ not interaction in workflow

Yah you can take input

But the thing is it applicable for short running workflow not for long running

It is not efficient and feasible



chain pauses [input ? OR (dividing two parts)]

But not maintainable and glue code is there a lot.

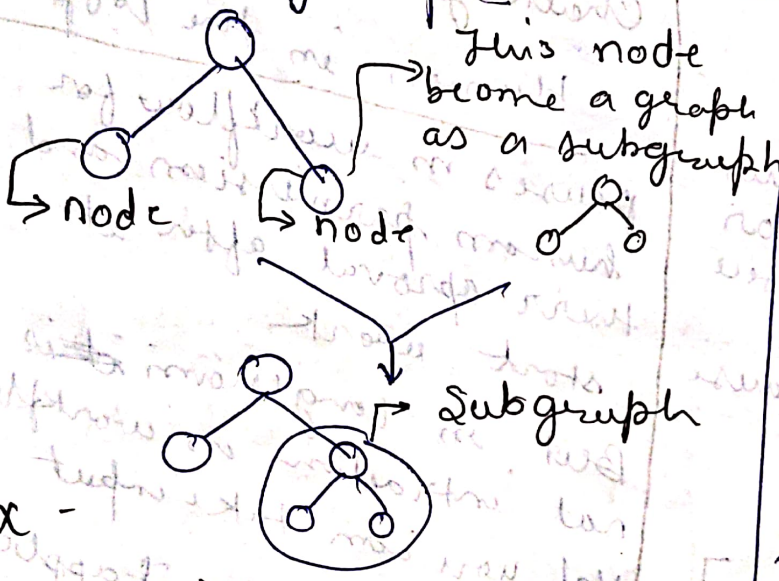
~~and~~ and later resume the work flow without any time constraints.

Nested workflow

LangGraph's feature

→ It use in like a subgraph →

A subgraph is graph that use as a node in another graph — this is concept of encapsulation. applied to LangGraph



Ex -

Multiagent System

→ like Self driving cars

→ To build Multiagent system → By

1. Subgraph
2. Reusability

LangChain's Challenge

It have no capabilities to do this by no concept of nested workflow

Challenge 1 Observability →

refers to how easily you can monitor, debug, and understand what your workflow is doing at runtime

But → It is in LangChain but via LangSmith. But it can only monitor LangChain code not glue code the bad things is in LangChain a glue code is there a lot to build good workflow

So, it is partial observability

1. Observability

It store whole timeline of workflow via LangSmith in LangGraph

when you deploy workflow you can monitor very efficiently with LangSmith they are very tightly coupled to this feature integrations

LANGGRAPH → It is an orchestration framework that enables you to build stateful multistep, and event driven workflows using LLM's. It's ideal for designing both single agent and multi agent agentic AI Applications.

Think of LangGraph is flowchart engine for LLM's —

You define the steps (nodes), how they're connected (edges) and the logic that governs the transition. LangGraph take care of state management — conditional, branding, looping, pausing/resuming and fault memory — feature essential for building robust, production-grade AI system —

WHEN

LangGraph	LangChain
◦ Conditional path	◦ Simple
◦ Loops	◦ Linear workflow
◦ Human in the loops	◦ For single task
◦ Multi Agent coordination	
◦ Asynchronous or event-driven execution	

→ LangChain Advance → is LangGraph.

⟹ Not replacing the LangChain.

Lecture → 04

Long Graph Core Concept

21-08-

It enables advance features like parallelism, loops, branching, memory and reusability making it ideal for agentic and production grade AI applications.

Long graph is an orchestration framework for building intelligent, stateful, and multistep LLM workflows.

LLM Workflows → Series of task to a achieve a goal.

They are a step process using which we can build complex LLM applications.

Each step in workflow perform a distint task - such as prompting, reasoning, tool calling, memory access or decision making.

Common Workflows →

1. Prompt Chaining
2. Routing
3. Parallelization
4. Orchestrator Workers
5. Evaluator Optimizer

Graph → Nodes (what to do) + Edges (when to do) →

State → State is the shared memory that flows through your workflow. It holds all the data being passed between nodes as your graph runs.

→ Every node have access of states or shared or input

→ It is mutable and special type dictionary called TypedDict (alternate Pydantic) use via library.

Reducers → It define how updates from nodes are applied to the shared state.

→ Each key in the state can have its own reducer, which determine new data replaces merges, or adds to the existing value.

→ LangGraph Execution Mode [GoogleProge]

1. Graph Defination
2. Compilation
3. Invocation
4. Super Step begin → Node Activation
5. Message Passing
6. Halting Condition
   - Execution stop when
     - No nodes are active
     - No message are in transit

Lecture → 05 **Sequential Workflow** → Practical

→ BMI ~~Graph~~ basic workflow
Now adding some functionalities and features.
→ LLM workflow → In this pass a ~~partial~~ state
→ Prompt Chaining *

Lecture → 06 **Parallel Workflow** → Practical

→ Simple Parallel Workflow → batsmen-workflow → In this passing a partial_state
→ Parallelization Workflow →
↳ UPSC Essay Evaluation
↳ workflow
↳ Prompt Chaining → 22-08

Lecture → 07 **Conditional Workflow** using manual way not command.

→ Quadratic Equation Workflow
$ax^2 + bx + c = 0$
↳ roots → 2
→ Discriminant $= b^2 - 4ac = d$
$d > 0$ (2 real root) $d = 0$ (2 repeated root) $d < 0$ (dont have any root) Real

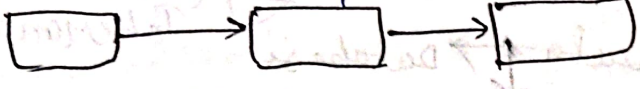
$x_1 = \frac{-b + \sqrt{d}}{2a}$, $x_2 = \frac{-b}{2a}$
$x = \frac{-b - \sqrt{d}}{2a}$

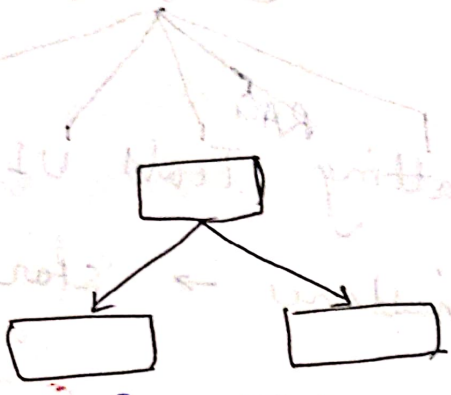
→ LLM Based Conditional Workflow →
↳ Customer Support Assistant Workflow.

Lecture → 08

Iterative Workflow

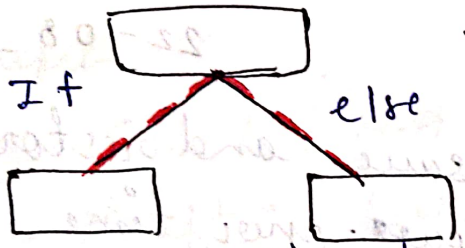
Sequential





Parallel

Workflows.

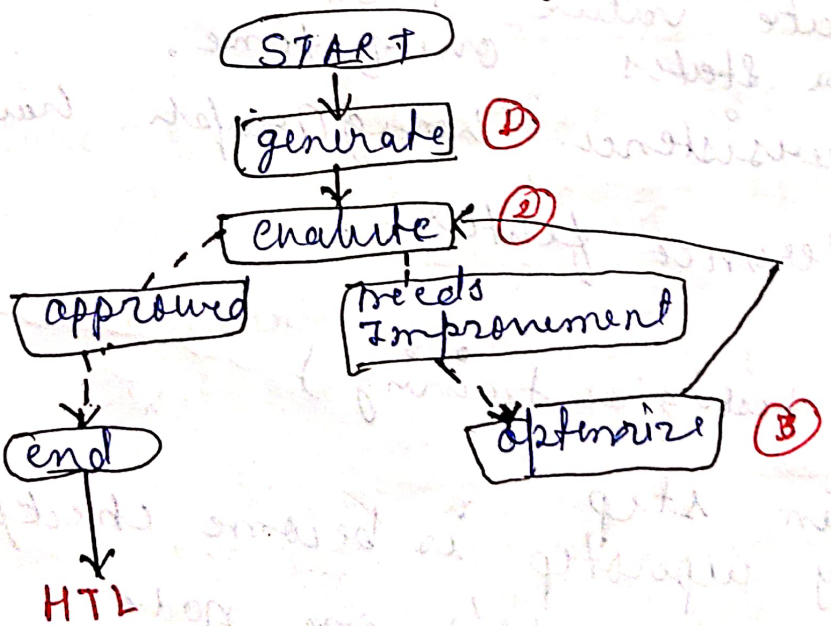


Conditional

Iterative or looping workflow →

Automated

eg.



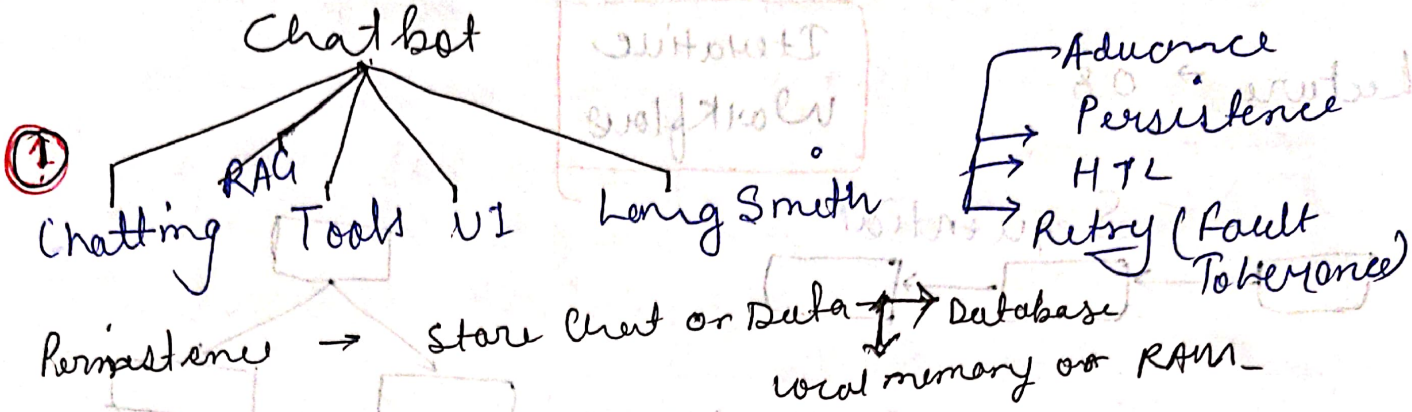
→ Now build this!

Lecture - 09

Chatbot in LangChain

22-08

1.



Persistence → Store Chat or Data ↔ Database
local memory or RAM.

Lecture - 10

Persistence

22-08.

In LangGraph it refers to save and restore the state of a workflow over time.

It not only store final value it also store intermediate value means all value of or states over time.

Because of persistence LangGraph have Fault Tolerance feature.

- Store
  - Database
  - RAM (just for testing or training)

Checkpointer in step

In graph every superstep is become checkpointer

Superstep → when in graph's one node going multiple node or another node this process or step is called one superstep.

Each time on superstep is check by checkpointer.

Threads In persistence →
While you store data in database of multiple state or multiple execution, so to don't occure the issue and conflicts we provide an id to each execution and that is known is called Thread.ID

By define the execution by assigning thread ID user can resume their history to check and see it.

Benefits →
- Short term memory.
- Fault Tolerance
- HTL (Human in the loop)
- Time Travel. [for replay the execution of code it helps a lot in debugging]

Lecture - 11 — Chatbot with UI in LangGraph

LangGraph Backend + Streamlit Frontend — Done (2)

Lecture 12 — Streaming In LangGraph — Done (3)

Streaming →
In LLMs, streaming means the model starts sending token (words) as soon as they're generated instead of waiting for the entire response to be ready before returning it.

Why →
1. Faster response time - low drop off rates
2. Mimics human like conversation (Builds trust, feels alive and keep user engaged)

3. Important for multi modal LLMs
4. Better UX for longcast such as code, articles etc -
5. You can cancel midway & saving tokens.
6. You can interleave UI updates -
eg, show, "thinking..." , show tool result.

Generator → In Python, a generator is a special type of iterator that allows you to generate values on the fly, one at a time, using the yield keyword instead of return

Lecture-13

Resume Chat feature

23-08

Done (4)

[chatbot history] like chatGPT or any other models

Lecture-14

SQLITE In LangGraph

Done (5)

Lecture -15

LangSmith

23-08-

Observability → is the ability to understand a system's internal state by examining its external output. 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 to able ask why something is happening within a system even if you don't anticipate the problem.

Langsmith → It is a unified & evaluation platform where teams can debug, test and monitor AI app performance.

What does LangSmith Trace?

1. Input/Output
2. All the intermediate trace
3. Latency
4. Token usage
6. Error
5. cost
7. Tags
8. Metadata
9. Feedback

Core Concept → Project, Trace, Run

Execution of each component in project

LLM Application

Execution of every one of LLM

WHILE In Production -

Rag have two big failure modes:

1. Retriever errors - wrong/irrelevent docs retrieved
2. Generator errors - model hallucinates or misuse the context

In production it's unclear where it occur

So, LangSmith help in: automatically records

→ User Query
→ Retrieved
→ LLM prompt (with inserted docs)
→ LLM response

for trace every component of the RAG you have define library & import traceable

o LangGraph + LangSmith

→ Every graph execution can be logged in LangSmith as a trace.

→ Each node becomes a run inside a run inside the trace (eg - retriever, LLM, tool call, subgraph)

→ You can visualise the path taken:

o START → Retriever → Reranker → LLM → END.

→ If a workflow branches (conditional/parallel/subgraph), LangSmith was captured ~~[illegible]~~ which path was executed.

Other features of LangSmith → (under LLMOP's)

Monitoring & Alerting:

Monitoring in this look across many traces at once to track the overall health of your LLM system. It aggregates key metrics 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 (eg., spike in latency, higher error rates or unexpected cost growth).

In production, issues often appear first as pattern across multiple runs, rather than in single trace.

Observability + Single trace

Monitoring ↓ having multiple trace across and studied also

Evaluation:

In langsmith helps you systematically measures the quality of you LLM's outputs.

You can test against gold-standard datasets or apply custom evaluation such as faithfulness relevance or completeness. It supports multiple approaches: LLM as a judge
semantic similarity checks
custom Python Evaluators.

Try out LLM on some datasets this called evaluation.

Prompt Experimentation:

It allows you to systematically test and compare different prompt versions.

You can run A/B tests across prompts on the same datasets, track their performance against evaluation, metrices and record the outcomes.

Results are stored over time, giving you a clear history of which prompt variations worked best and under what conditions.

Also provide prompt versioning.

Dataset Creation & Annotation:

- Provides tools to build datasets for evaluation and fine tuning
- Supports manual annotations (eg. labeling whether an ans is correct)
- Stores datasets versioned for reuse across project.

Why → High quality datasets are critical for evaluation and feedback loop.

User feedback Integrations :

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

Collaboration :

- Team members can view, share and comment on traces datasets and evaluation
→ provides a web UI where non engg (PMs, QA, annotators) can inspect and annotate runs.
→ Enables shared experiment dashboard.

Lecture → 16 — Observability in LangGraph — Done (6)

Adding this feature in Chatbot and connect it to Lang Smith.

Lecture → 17 — Tools in LangGraph — 24-08- Done (7)

Tool node → It is a prebuilt node type that acts as a bridge between your graph and external tools (functions, APIs, utilities etc)

tools_condition → It is prebuilt conditional edge function that helps your graph decide. Should the flow go to the Tool Node next, or back to the LLM?