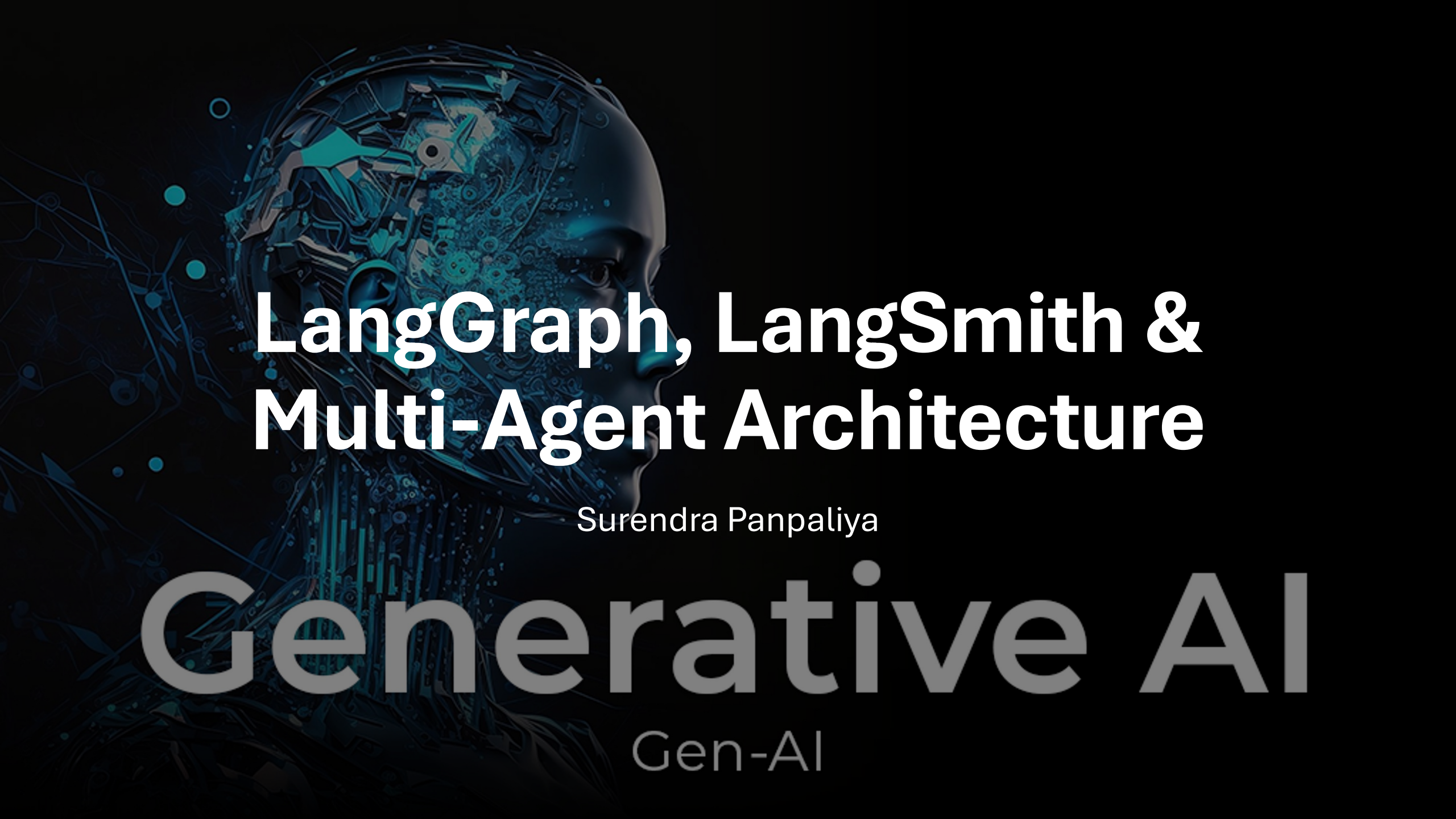


LangGraph, LangSmith & Multi-Agent Architecture

Surendra Panpaliya

Agenda

Why orchestration matters:

connecting multiple agents and workflows

LangGraph Architecture:

state graph, nodes, and conditional flows

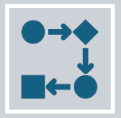
Agenda

LANGSMITH

TRACING, DEBUGGING, AND

OBSERVABILITY OF LLM PIPELINES

Demo



Visualizing a workflow:

“Data → Retriever → Validator → Summarizer → Notifier”

Implementing decision branches

using LangGraph visual editor

Demo

- **Multi-Agent Orchestration:**
- Planner, Reviewer, and Reporter agents (no coding)
- Real-time demonstration of AI pipeline monitoring using LangSmith UI
- Compliance audit example:
- tracking AI outputs and data lineage

Agentic AI & Task Chaining

What is Agentic AI?

Core components:

Agents

Tools

Planner

Executor

What is Agentic AI?

Agentic AI = Smart AI agents that

Understand goals

Plan steps

Execute tasks

Work together (like a team)

What is Agentic AI?

New way of using AI

to **plan, execute, and manage tasks**

independently

like an intelligent assistant

that thinks and acts.

What is Agentic AI?

ACT LIKE A RESPONSIBLE ASSISTANT,

DOING TASKS **INDEPENDENTLY**

BASED ON GOALS YOU GIVE IT.

Agentic AI uses Agents that can

Understand goals

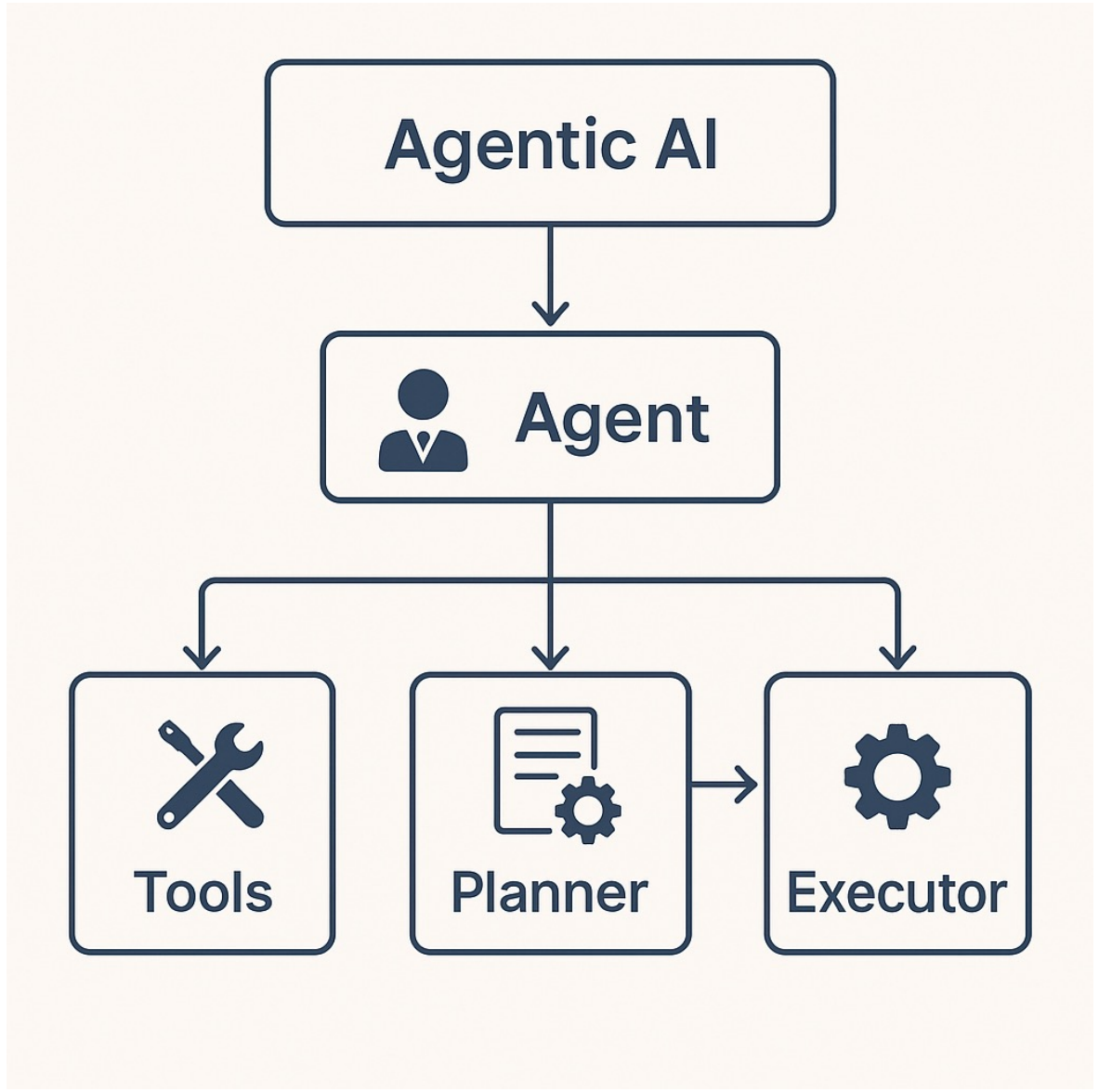
Break them into tasks

Use tools or APIs to complete tasks

Adjust actions based on results

Core Components of Agentic AI

1. Agents
2. Tools
3. Planner
4. Executor


Agentic AI
Agent
Tools
Planner
Executor

1. Agents

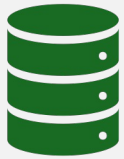
Individual units with specific roles or goals.

Can reason, learn, and make decisions

based on their goals.

2. Tools

Resources or capabilities available to agents.



For example: APIs, databases,

web search engines, or AI models.

3. Planner

Determines how to achieve the goal.

Decides which agents

should collaborate, and in what order.

4. Executor

Executes the planned tasks.

Coordinates interaction

between agents and tools

to complete tasks.

Core Components of Agentic AI

Component	Purpose
👨‍💼 **Agent**	An intelligent entity that receives instructions and decides what to do.
🧰 **Tool**	A function or API the agent can use (like a search engine, database query, code interpreter, etc.)
📋 **Planner**	Breaks down a big problem into smaller tasks.
⚙️ **Executor**	Executes tasks one by one, using tools, and adjusts based on feedback.

What Are Agent Design Patterns?

AGENT DESIGN PATTERNS DEFINE

HOW AN AI AGENT

THINKS, PLANS, AND

EXECUTES TASKS.

What Are Agent Design Patterns?

These patterns guide:

- How the agent reasons
- How tasks are broken down
- How tools are used
- How the agent adapts to feedback

Key Agent Design Patterns

ReAct (Reasoning and Acting)

Chain-of-Thought (CoT)

Plan-and-Execute (PnE)

ReAct (Reasoning and Acting)

The agent **reasons**

step-by-step and then

decides an action,

repeating until

the final answer is found.

Pattern

Thought → Action → Observation →

Thought → Action → … → Final Answer

Chain-of-Thought (CoT)

The agent is encouraged

to **explain its reasoning process**

before giving an answer.

It doesn’t act with tools

but reasons clearly.

What is Chain-of-Thought (CoT)?

Instead of jumping to the final answer,

the model is instructed to **think aloud**

by breaking the problem into

logical reasoning steps.

What is Chain-of-Thought (CoT)?

Improves accuracy, interpretability

often factual correctness,

especially for complex or

multi-step questions.

Plan-and-Execute (PnE)

The agent **first plans**

the full sequence of tasks,

then executes each step.

What is Plan-and-Execute (PnE)?

Powerful agent pattern that:

First plans a sequence of subtasks

based on the user request.

Then **executes each step**,

often using tools, APIs, or functions.

What is Plan-and-Execute (PnE)?

This improves

reliability,

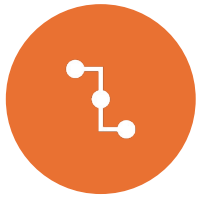
transparency, and

modularity in AI task solving.

Why Orchestration Matters?

Connecting Multiple Agents and Workflows

Single LLM Calls to Systems of Agents



Move beyond a single prompt–response interaction

Need **orchestration**

The coordination of multiple agents

(each with a role) and tools

like databases, APIs, or

human-in-the-loop steps).

Healthcare RAG setup

RETRIEVER AGENT PULLS POLICY DATA.

SUMMARIZER AGENT DISTILLS IT.

COMPLIANCE AGENT CHECKS REGULATIONS.

AUDITOR AGENT LOGS ACTIVITY.

Without orchestration

Agents act independently

causing duplication,

errors

untraceable logic flow.

Core Benefits of Orchestration

Aspect	What It Enables	Example
Control	Structured task sequencing	Query → Retrieve → Reason → Respond
Transparency	Track state and data lineage	Audit trail of retrieved sources
Error Handling	Retry/fallback logic	Retry embedding store if vector DB fails

Core Benefits of Orchestration

Aspect	What It Enables	Example
Parallelism	Concurrent agent execution	Policy summarization and fact extraction in parallel
Scalability	Add/modify agents without breaking flow	Plug in a new “translator” agent easily

Analogy

Think of orchestration as a

Conductor in an Orchestra

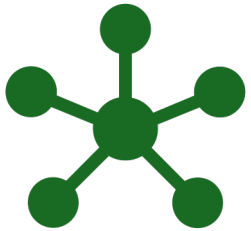
Agents (instruments) play different roles,

Conductor ensures

Timing, harmony, and handoffs.

LangGraph Architecture

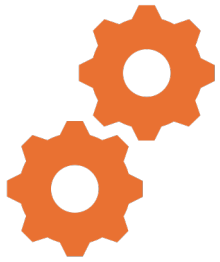
State Graph



Nodes

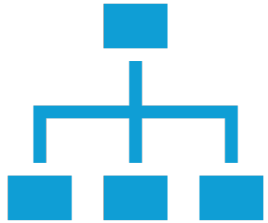
Conditional Flows

LangGraph Architecture



Open-source orchestration framework

for building Multi-agent systems



with dynamic control flow.

LangGraph Architecture

Graphs-as-programs

State transitions and

Node executions

Define how agents interact.

Core Components

Concept	Description	Analogy
Graph	The overall workflow connecting agents, tools, and logic paths	Blueprint of a system
Nodes	Executable units (functions, LLM calls, agents)	Steps or roles
Edges	Transitions between nodes (with conditions)	Arrows showing flow

Core Components

Concept	Description	Analogy
State	Shared memory or context that flows between nodes	Shared “workspace”
Conditional Flow	Decision logic that determines which path to take next	“If answer not found → fallback to external search”

Why LangGraph Is Powerful?

Combines **LangChain Runnables** with **graph-based orchestration**

Why LangGraph Is Powerful?

Allows **visualization**

Debugging

Multi-step LLM

Reasoning

Why LangGraph Is Powerful?

Supports

Persistent state

across nodes

Why LangGraph Is Powerful?

Works with **async,** **parallel,** **conditional** executions

Why LangGraph Is Powerful?

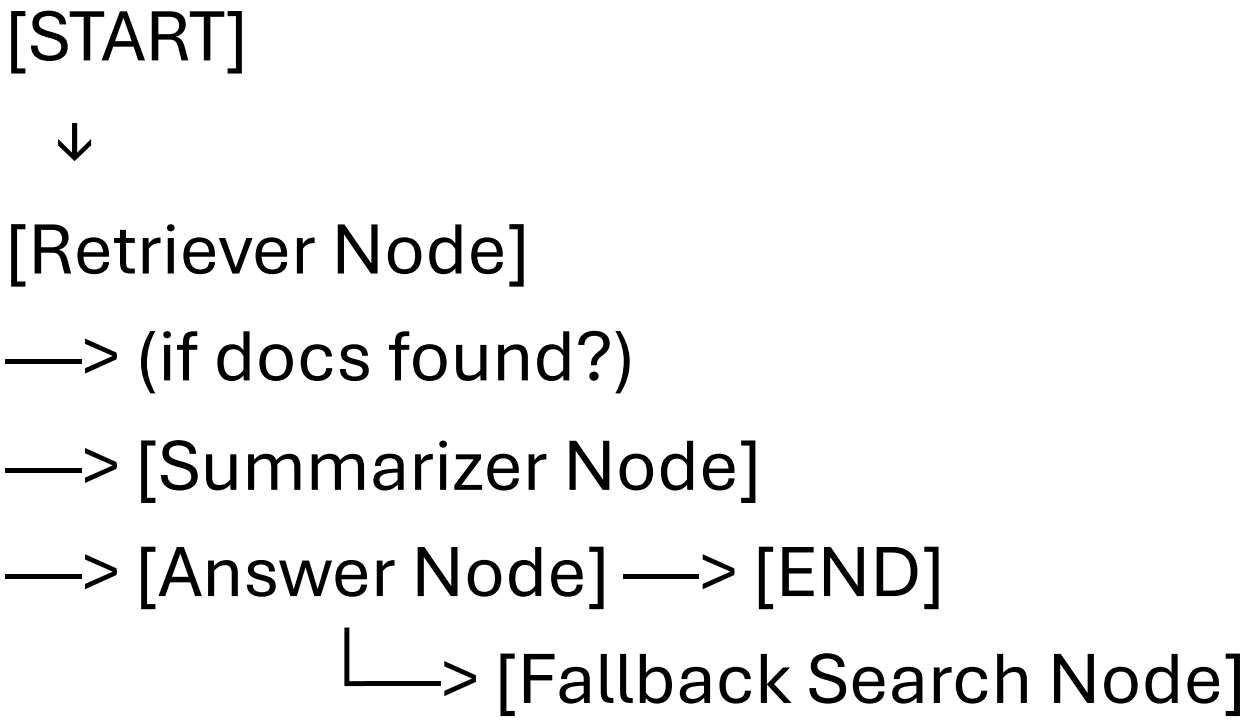
Ideal for

Multi-agent collaboration

Retrieval workflows

Governance tracking

Visualization of Flow



Orchestration + LangGraph = Cohesive Multi-Agent Systems

Orchestration provides

“why and when” (strategy).

LangGraph provides

“how” (implementation pattern).

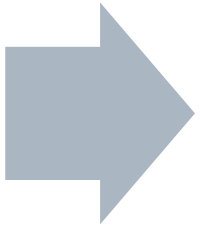
Concept Reinforcement

Concept	Implementation
Orchestration	Graph coordinates agents via state and edges
Multi-Agent Workflow	Retriever → Summarizer → Answer → Auditor
Conditional Flow	Branches if no docs retrieved
Governance & Audit	State tracks every step’s activity
Extensibility	Add “Reviewer” or “Compliance Agent” easily

Multi-agent orchestration

Instead of one

“do-everything” agent,



Use **specialized agents**

that collaborate

Multi-agent orchestration

Planner:

decomposes goals into steps

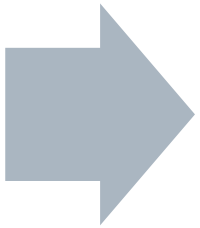
Researcher:

searches/reads knowledge base & tools

Multi-agent orchestration

Coder:

generates/fixes code & tests



Reviewer:

checks quality/safety

Multi-agent orchestration

Supervisor:

routes work among agents,

decides “done/hand back to user”

Multi-agent orchestration Benefit

Separation of concerns,

Easier testing/guardrails,

Better reuse.

Why LangGraph for orchestration?

State graphs with nodes

(functions/LLMs/tools)

conditional edges

Checkpoints/memory

per session (resume, replay)

Why LangGraph for orchestration?

Built-in **tool execution** nodes

Guardrails (insert moderation/validators as nodes)

Deterministic control flow you can test

LangSmith for Solution Architects

TRACE

EVERY LLM CALL

TOOL STEP

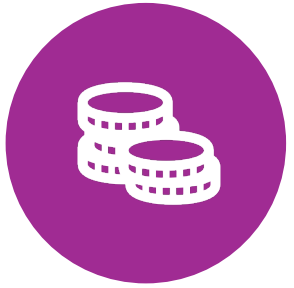
IN A PIPELINE

LangSmith for Solution Architects

Debug

Failures and regressions

with exact inputs/outputs and



token/cost telemetry

LangSmith for Solution Architects

OBSERVE

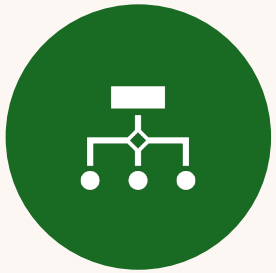
LATENCY, COST, AND

QUALITY OVER TIME WITH

DATASETS, EVALS, AND FEEDBACK

LangSmith for Solution Architects

GOVERN



RUNS WITH **PROJECT-LEVEL ISOLATION**,

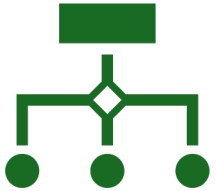
REDACTION, AND

AUDIT-READY ARTIFACTS

Why Observability Matters in Pharma/Healthcare

Audit & compliance

Preserve who ran what,



with which prompt/model/parameters

when (key for SOPs, CAPAs, validation).

Safety & privacy

Detect	Detect PHI leaks,
Apply	apply redaction,
Tag and quarantine	tag and quarantine risky runs.

Reliability

Reproduce incidents with

full traces to speed up MTTR.

Cost control

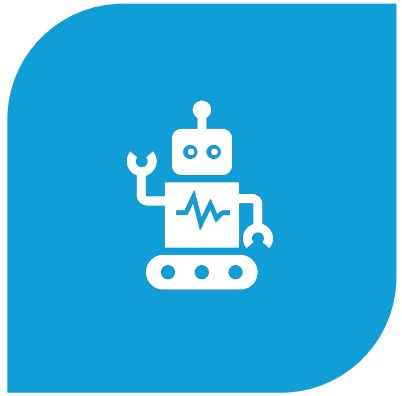
Track tokens + latency per component

(triage “where time/money goes”).

Quality

CLOSED-LOOP IMPROVEMENTS

VIA USER FEEDBACK +



AUTOMATED EVALS.

Core LangSmith Concepts (mental model)

Project

Logical workspace

(e.g., Viatris-Safety-QA).

Run / Trace

A single execution path

across LLM calls,

tools,

retrievers,

chains.

Dataset

Versioned test

set of

input → expected output

Evaluation

AUTOMATED GRADING

(LLM-AS-JUDGE OR RULE-BASED)

OVER A DATASET.

Core LangSmith Concepts (mental model)

Feedback:

Human or programmatic signals

Categories,

Numeric ratings,

Comments

Core LangSmith Concepts (mental model)

TAGS/METADATA:

KEY–VALUE CONTEXT TO FILTER,

SLICE, AND ANALYZE

(E.G., ENV=STAGING, PHI=FALSE).

What is LangSmith?

Platform designed for

observability, evaluation, and deployment

of large-language-model (LLM) applications and

agent workflows

Key aspects

Framework-agnostic.

LangSmith works

with any framework.

Tracing:

Capture each run of your LLM

Agent pipeline

inputs, outputs, metadata,

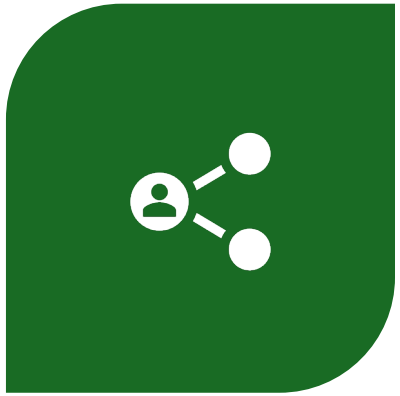
timing, cost

Debugging / root-cause

See	See where things go wrong
Tool	Tool calls, retries, errors
Enable	Enable alerts.

Evaluation + feedback

CREATE DATASETS OF INPUT → EXPECTED OUTCOME,



EVALUATE YOUR LLM/AGENT PERFORMANCE,

COLLECT FEEDBACK/HUMAN-IN-LOOP.

Governance / auditability / deployment

Support for enterprise-grade security,

Tag runs with metadata

Retention policies,

Redaction, export of traces for audits.

In short

If you build LLM-based workflows

Care about reliability, cost, compliance,

then LangSmith is the

observability + evaluation house around them.

LangChain vs LangGraph vs LangSmith

Tool	Primary Purpose	Best For	Key Strengths	Things to Watch/Limitations
LangChain	Framework for building LLM apps (chains, tools)	Linear or moderately complex LLM workflows (chatbots, RAG)	Easy to get started, rich integrations with models/data (Codebasics)	When flows get complex (branching, loops, state) it may become harder to manage
LangGraph	Graph-based orchestration of workflows/agents	Complex workflows with branching, loops, stateful agents	Explicit graph control, state management, agent support (LangChain)	Steeper learning curve; more infrastructure/complexity
LangSmith	Observability, tracing, evaluation, monitoring	Production deployments of LLM/agent workflows	Provides dashboards, tracing, feedback loops, audit-ready metrics (Medium)	It's not a workflow engine by itself; you still need to build the pipeline

Summary — Key Takeaways

USE LANGCHAIN FOR BUILDING YOUR

INITIAL LLM WORKFLOWS

DOCUMENT Q&A,

SUMMARISATION,

RETRIEVAL

Summary — Key Takeaways

When workflows become multi-step,

stateful, or agentic

(for example PV triage agent with human review),

consider LangGraph for orchestration.

Summary — Key Takeaways

Regardless of workflow complexity,

embed LangSmith from early on

to enable tracing, cost/latency monitoring,

feedback loops, audits.

Summary — Key Takeaways

In the pharma/healthcare domain

you have additional demands

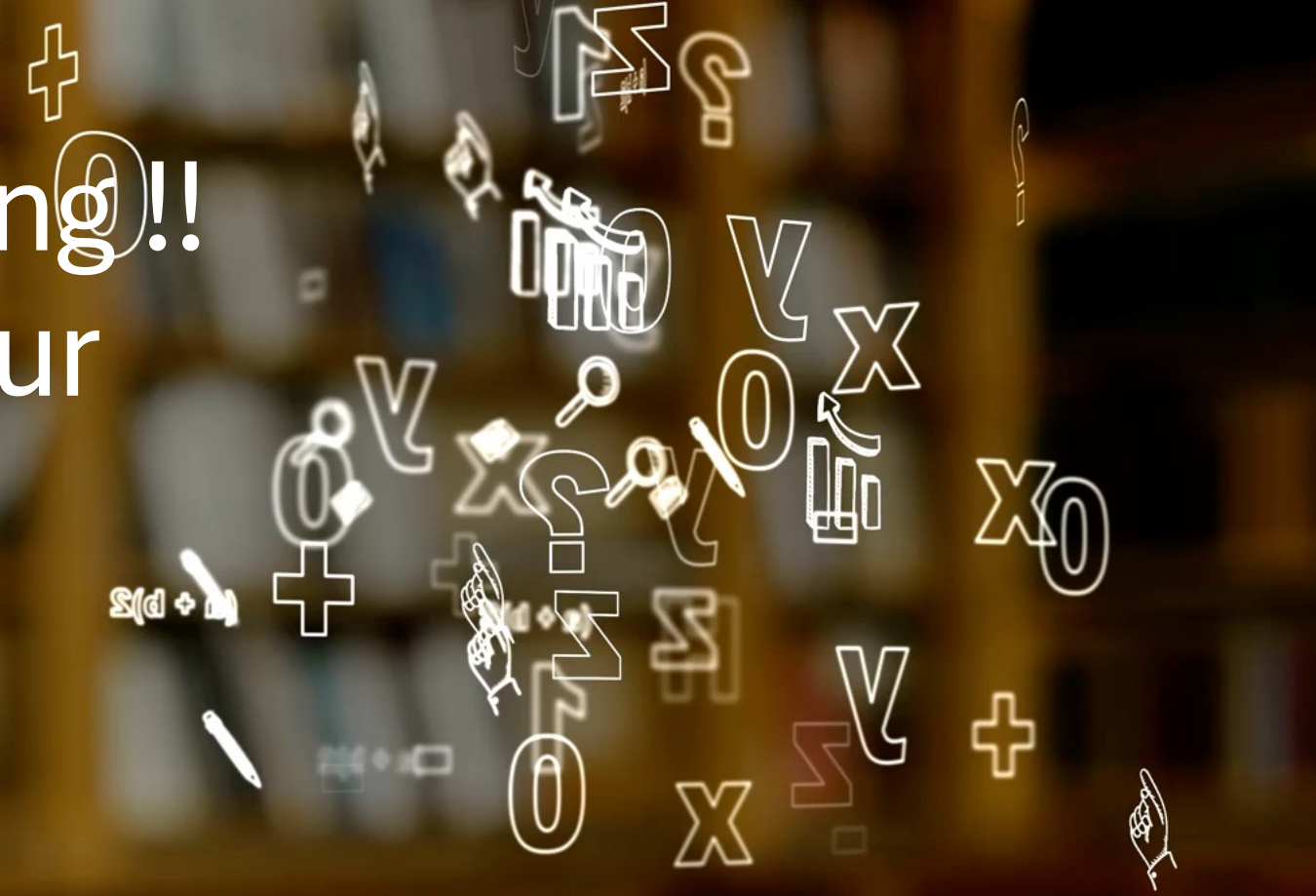
(compliance, PHI, audit trails)

which make observability, tagging,

review workflows essential

LangSmith is a strong fit here.

Happy Learning!! Thanks for Your Patience ☺



Surendra Panpaliya

GKTCS Innovations