Working with Frameworks

LangChain & LangGraph: Orchestrating Complex Workflows

AutoGen: Building Conversational Systems

Course Roadmap: From Foundations to Capstone

01

Understanding the Fundamental

LLM architecture, history, and core prompt engineering.

02

Making Agents Powerful

RAG & Tool usage

03

Working with Frameworks

LangGraph, and building complex chains.

AutoGen, MCP

04

Observability & Fine Tuning

Tracing (LangSmith), evaluation, and custom model tuning.

05

Capstone Project

End-to-end implementation and final presentation.

Day 3 Roadmap: Orchestration

Today's Goal

Learn to structure and connect multiple LLM calls, tools, and data sources into a robust, repeatable system.

Focus: Modular Design and State Management.

Popular Agentic AI Patterns & Implementation of Planner-Executor

Why Framework?

LangGraph: State & Cycles Building agents that can reason and loop

MCP &AutoGen Framework build conversational agents

Hands on & Task for Day4 (Reading on evaluation Metrics)

https://docs.langchain.com/langsmith/evaluate-rag-tutorial

Task 1

Implement tool calling without Langchain agent

Task 2

- Identify Prompts used for creating SQL agent in langchain. Identify possible issues with that
- Identify steps taken by SQL agent behind the scene.

How can we improve the performance of SQL Agent?

Let us revisit Prompt Engineering

Zero-Shot

Asking the model to perform a task without any examples. Relies entirely on its pre-trained knowledge base.

✓ Few-Shot

Providing 2-3 examples of inputs and desired outputs in the prompt to guide the model's pattern matching.

Chain-of-Thought

Instructing the model to "think step-by-step." dramatically improves performance on complex logical tasks.

When to use few-shot prompting with SQL agents

Few-shot examples are useful when:

- The model generates **incorrect SQL dialect** (Postgres vs MySQL vs SQLite)
- It struggles with **JOIN logic**
- You want **consistent query style** (CTEs, snake_case, limits)
- You want to enforce **business logic** (e.g., exclude soft-deleted rows

Task 3 - For today

Create multi tool calling agent

https://www.geeksforgeeks.org/blogs/free-apis-list/

Foundational Agentic Architectures

Single-step Tool Agent

Direct Action & One-Shot

The simplest agent: determines the single best tool to solve the request, executes it, and formats the result.

Description: LLM calls one tool without intermediate steps.
Example: SQL query agent, simple API fetch.

ReAct (Reason + Act)

Interleaved Reasoning Loop

The core agentic loop. It generates a thought , executes an (tool call), and receives an observation to iterate.

Description: Interleaves reasoning + action for multi-step tasks.
Example: Complex chat-based tool use (e.g., browsing, calculation).

Planner–Executor

Decoupled Strategy

Splits the cognitive load: one agent creates a list of steps, and a separate agent or workflow executes those steps sequentially.

Description: Separate planning (strategizing) and execution (doing).
Example: Incident response, multi-stage data processing.

Scaling Agency: Complex & Collaborative Architectures

Self reflection

This pattern critiques the output and generates a decision on whether to keep to accept the output or decline it.

Supervisor–Worker

Delegation and Management

A master agent directs the flow of multiple specialized worker agents, ensuring task completion and consistency.

Description: One agent manages many specialized executors.
Example: Multi-team automation (e.g., AutoGen's conversational approach).

Autonomous Loop Agents

Self-Directed Iteration

The ultimate agent, designed for indefinite execution until a clear goal is met, using continuous reflection and self-correction.

Description: Self-directed iteration with continuous reflection and refinement.
Example: Research agents, continuous monitoring/optimization.

Hands on

Why Frameworks? The Production Requirement

The Challenge with "Vanilla" API

A bare LLM API call is simple, but any real-world application requires sequencing:

- **Pre-Processing:** Formatting user input, running RAG retrieval/Calling tools.
- **Post-Processing:** Validating JSON output, retrying failed calls, logging.
- **Tool Orchestration:** Deciding *when* to call a tool and feeding the result back.

What is LangGraph? Orchestration for Agents

LangGraph: Graph-Based Execution

LangGraph is a framework built on top of LangChain, designed to model and execute stateful, multi-step, multi-agent workflows using graph-based execution.

Why Graphs?

Unlike linear chains, agents require complex flow:

- Conditional branching (if tool call fails).
- Loops (retry, reflection, correction).
- Parallel execution (running multiple agents simultaneously).

LangGraph makes this complexity **explicit, auditable, and maintainable**.

LangGraph doesn’t make models smarter — it makes agents controllable.

LangGraph: Why It's Needed in Production Systems

Production Systems are Not Linear

Real-world AI workflows involve loops, decisions, and external interaction that linear chains cannot cleanly express. LangGraph addresses this by enabling:

- **Conditional Branching:** "If validation fails, go to the Human-in-the-Loop node."
- **Loops & Retries:** Automatic self-correction loops for tool or LLM failures.
- **Multi-Agent Coordination:** Structuring communication between a Planner, a Coder, and an Executor agent.

LangGraph: State, Nodes, and Edges

Core Concepts

- **State:** The single, consistent source of truth (e.g., chat history, retrieved documents, current tool call).
- **Nodes:** The functions or components that execute logic and modify the State. (e.g., llm calls, executors).
- **Edges:** The connections that define the flow *between* nodes.
- **Conditional Edges:** The decision gates. Based on the State, the graph decides which node to execute next (e.g., "If LLM output is a Tool, go to Tool Executor node").

Activity

Goal: Work on a Lead generation system to identify potential clients from linkedin and rank order them based on relevance.

Requirements: No-coding only components of your workflow

https://docs.langchain.com/oss/on/langgraph/thinking-in-langgraph

Conditional Execution

```
graph.add_conditional_edges(
    "reflect",
    decision_router,
    {
        "persist": "persist",
        "execute": "execute",
        "interrupt": "interrupt"
    }
)
```

How This Works (Step-by-Step)

1. The `"reflect"` node finishes execution
2. `decision_router(state)` is called
3. The function returns a **string key**
4. LangGraph routes execution to the mapped node

Hands on

What is MCP

- MCP (Model Context Protocol) is an open-source standard for connecting AI applications to external systems.
- Using MCP, AI applications like Claude or ChatGPT can connect to data sources (e.g. local files, databases), tools (e.g. search engines, calculators) and workflows (e.g. specialized prompts)—enabling them to access key information and perform tasks.
- Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect electronic devices, MCP provides a standardized way to connect AI applications to external systems.

Why Do We Need MCP Tools?

Large Language Models:

- Cannot browse the web by default
- Cannot safely execute external actions

MCP tools solve this by:

- Isolating tools from the model
- Allowing controlled access to external systems
- Enabling real-world interaction (web, files, databases)

We will use “mcp-server-fetch”

Capabilities:

- Fetches web pages via HTTP
- Returns page content to the agent
- Handles redirects and errors
- Works in a sandboxed environment

Example Use Cases:

- Web research agents
- Wikipedia summarization
- News aggregation
- Live data ingestion

Hands on

Microsoft AutoGen

Conversational Programming &

Autonomous Agents.

The "Conversational" Paradigm

Standard Agents

Input -> Thought -> Action -> Output.

AutoGen

Agent A says something. Agent B responds.

Agent A corrects Agent B.

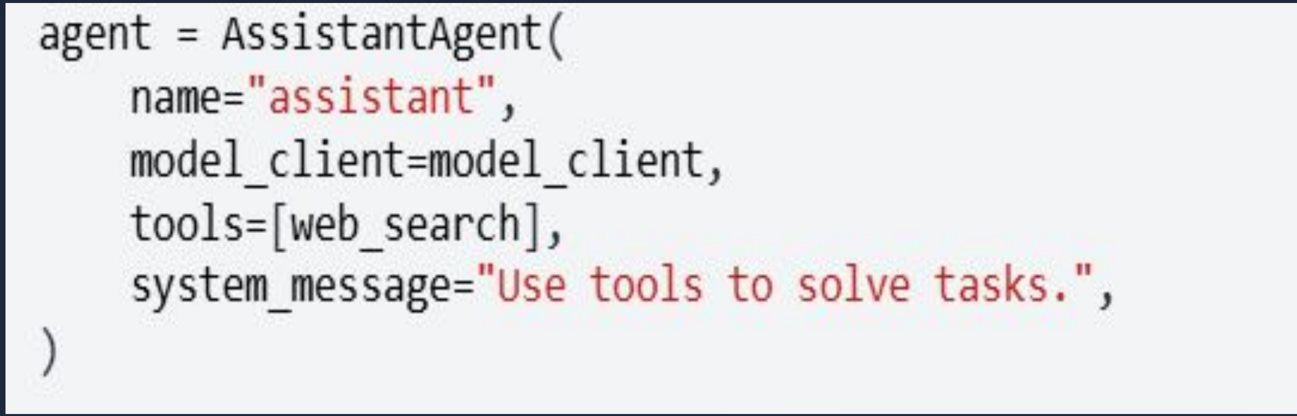
Dynamic and self-correcting loop.

Types of Agent in AutoGen

1. The AssistantAgent

The "Brain".

- Backed by an LLM (GPT-4).
- Writes code, plans tasks, solves problems.
- **Cannot execute code.** It only suggests it.



```
agent = AssistantAgent(
    name="assistant",
    model_client=model_client,
    tools=[web_search],
    system_message="Use tools to solve tasks.",
)
```

2. The UserProxyAgent (Human-in-the-Loop)

The "Hands" (and the Human Proxy).

- Can execute code (locally/Docker).
- Can ask the Human for input.
- **Default behavior:** If it sees a code block in the chat, it runs it and replies with the result.

Advanced Components
Group Chats

Scaling from 2 agents to a room full of them.

The GroupChat Manager

What if you have 3+ agents? (Coder, Designer, Product Manager).

The **GroupChatManager** acts as a moderator.

- It sees the conversation history.
- It decides **"Who speaks next?"** based on the LLM's decision.

Speaker Selection Methods

Auto

The LLM decides who is best suited to speak next.

Round Robin

A -> B -> C -> A.

Random

Chaotic, but sometimes useful for brainstorming.

Human Input Modes

human_input_mode	Meaning
"ALWAYS"	Always pauses for human input before continuing
"NEVER"	Fully autonomous, no human asked
"TERMINATE"	Stops when human input would normally be needed

Hands On

Q and A

Task for Day 4:

https://docs.langchain.com/langsmith/evaluate-rag-tutorial