Developing AI Agents / Multi-Agent Systems

Generative AI Learning Series

Alone, we go faster. Together, we go further.

Today’s Agenda

- AI Agent Overview
- Multi AI Agents Overview
- Multi AI Agents Architecture and Components
- Frameworks to Build Multi AI Agents
- LangGraph Framework Overview
- Demo with examples

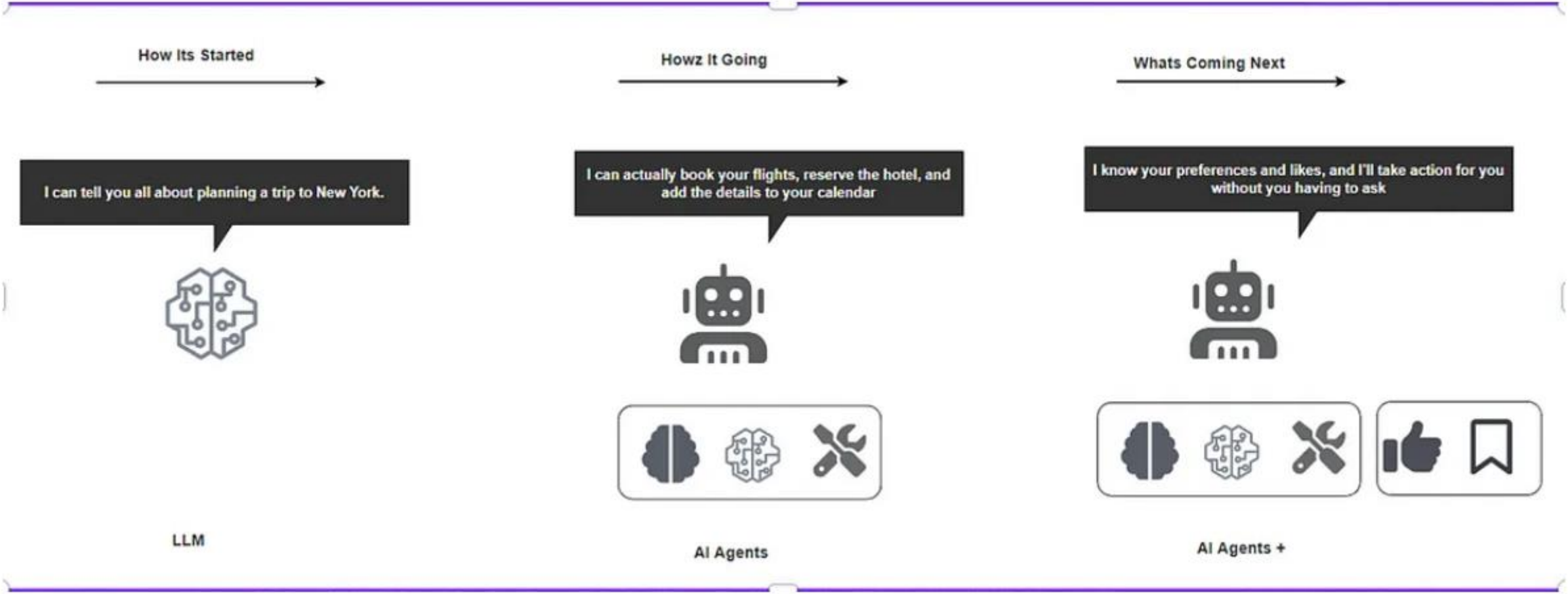
AI Agents Overview

- AI Agents are entities powered by LLMs and designed to perform tasks autonomously.
- AI Agents can plan, reason and take action to achieve specific goals.
- AI Agents are complex systems composed of several key components that work together to enable autonomous task completion.
  - LLM – The Brain
  - Planning – Dividing Objectives into Tasks
  - Action – Use of Tools
  - Profile – Describing Agent Behavior
  - Memory – Storing and Recalling Past Interactions

Why we all should learn and serious about AI Agents?

"Dear friends, I think AI agent workflows will drive massive AI progress this year — perhaps even more than the next generation of foundation models. This is an important trend, and I urge everyone who works in AI to pay attention to it." — Andrew NG

Why we all should learn and serious about AI Agents?



Microsoft is developing sophisticated AI agents that can perform complex tasks with minimal user intervention. These agents are designed to automate series of actions, such as creating and tracking client invoices based on order history, and even rewriting application code in different languages

https://www.theinformation.com/articles/to-unlock-ai-spending-microsoft-openai-and-google-prep-agents

In early 2024, Klarna, a prominent financial technology firm, reported remarkable results from its newly implemented AI Agent customer service assistant. Developed in collaboration with OpenAI, this virtual agent demonstrated impressive capabilities within its first 30 days of operation. The AI system successfully managed approximately 67% of all customer interactions, effectively performing tasks that would typically require a workforce of 700 full-time human representatives

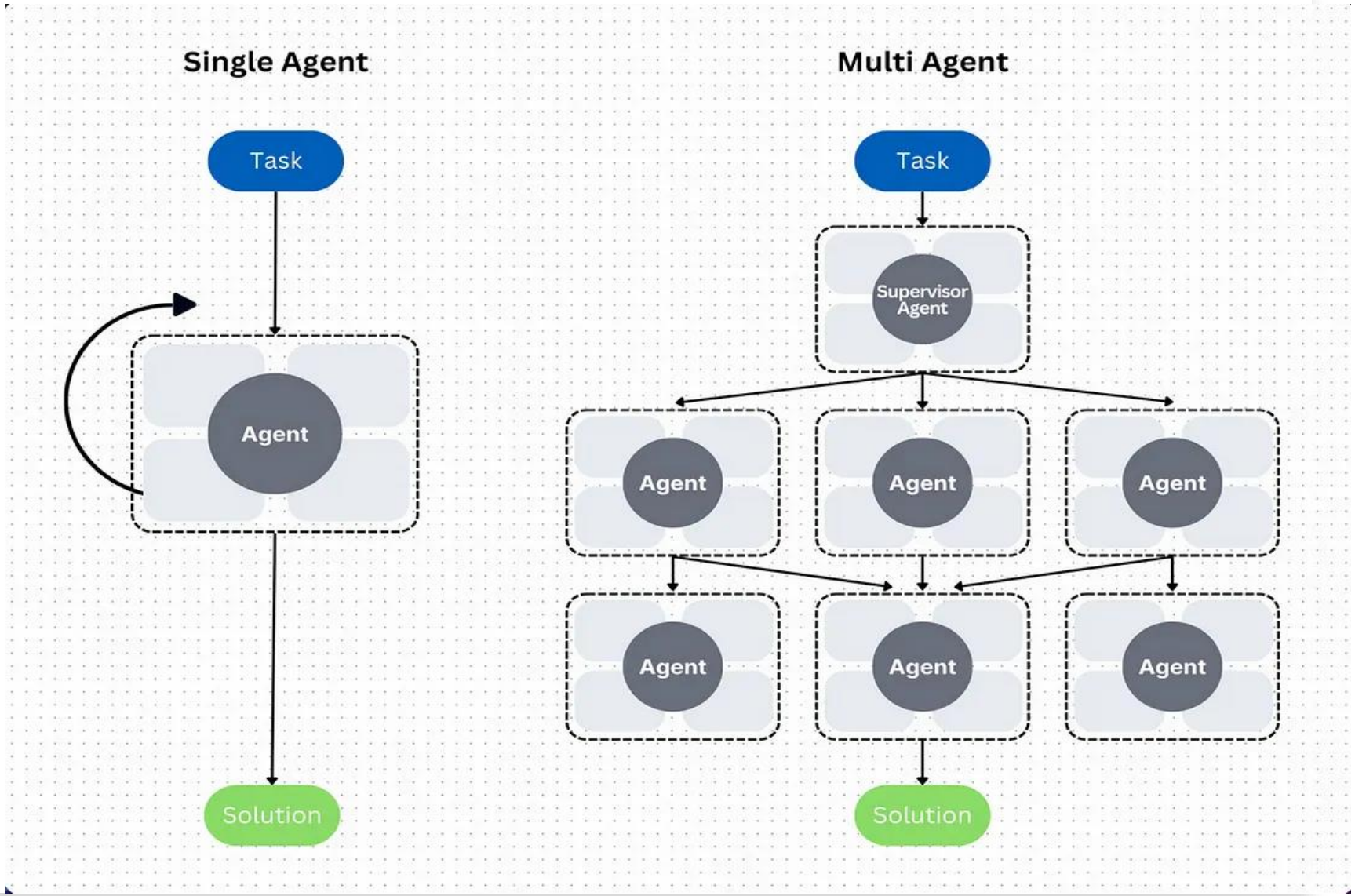
Why we all should learn and serious about AI Agents?

Multi-Agent System: Overview

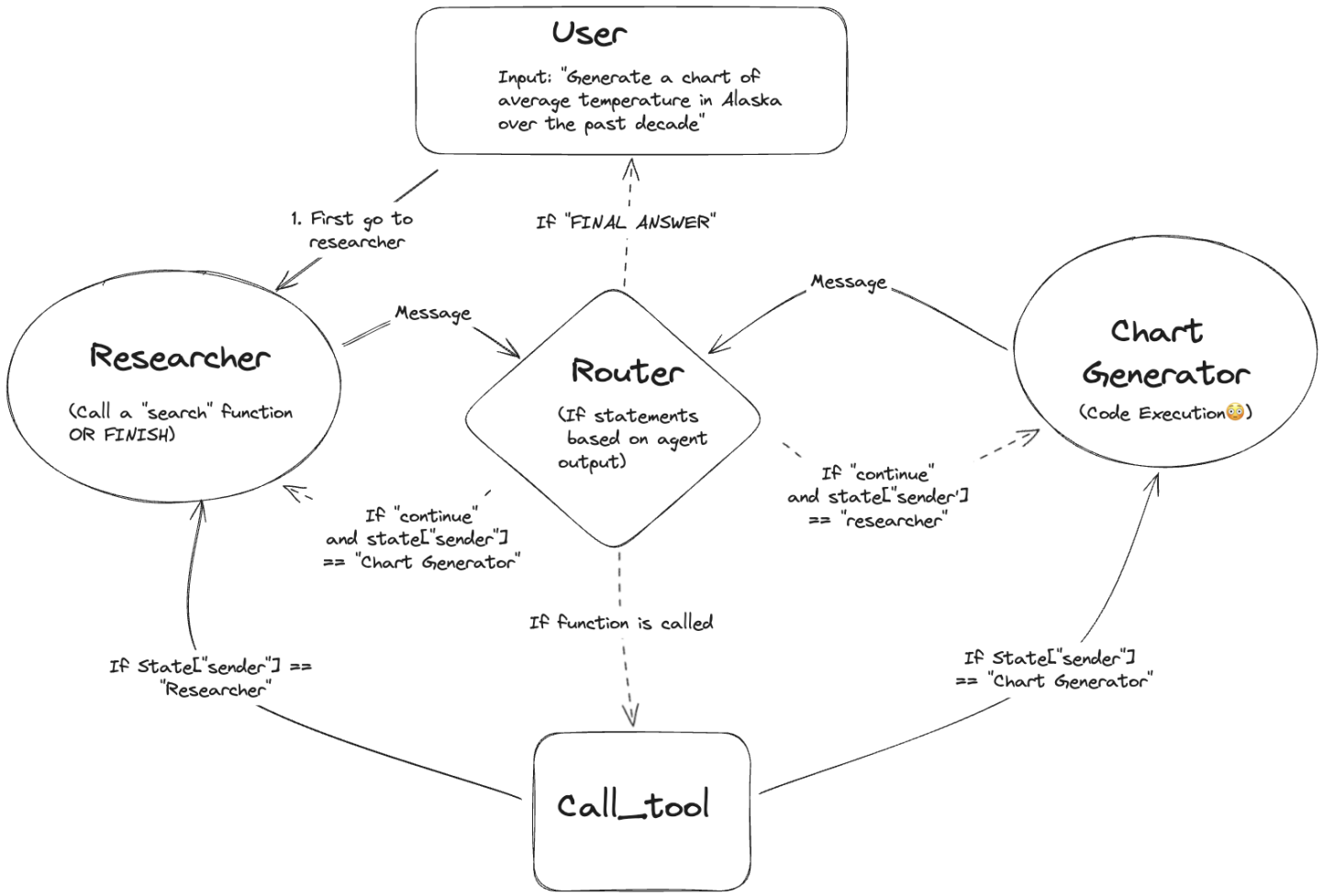
- Involves multiple independent actors, each powered by language models, collaborating in a specific way.
- Have their own persona/role, and a context that is define by the prompts on a specific language model.
- Each agent has access to various tools, to help execute the task given to the agent.
- Multiple agents bring different perspectives and helps make better decisions.

Multi-Agent System: Overview

- Multi-agent systems differ from single-agent systems primarily in the distribution of decision-making and interaction within a system.
- Single-agent system
  - A centralized agent makes all decisions, while other agents act as remote slaves.
  - Normally decides, based on the context. This might miss out the other perspectives.
- Multi-Agent System
  - Involve multiple interacting intelligent agents, each capable of making decisions and influencing the environment. The idea behind multi-agent architecture is to create agents, with different contexts to bring in different perspective, by the role they play.
  - Might be using the same LLM, but due to the role, goal and the context that is defined for that agent, they behave different.

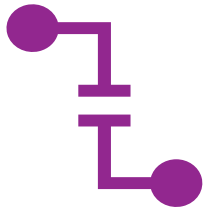


Multi-Agent collaboration



Multi-Agent collaboration

Benefits of Multi-Agent Designs



Separation of Concerns :

Each agent can have its own instructions and few-shot examples, powered by separate fine-tuned language models, and supported by various tools.

Dividing tasks among agents leads to better results.

Each agent can focus on a specific task rather than selecting from a multitude of tools.

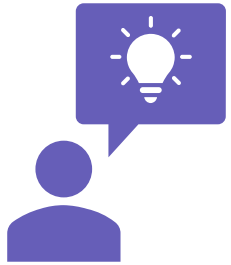
Modularity

Allow breaking down complex problems into manageable units of work, targeted by specialized agents and language models.

Multi-agent designs allow you to evaluate and improve each agent independently without disrupting the entire application.

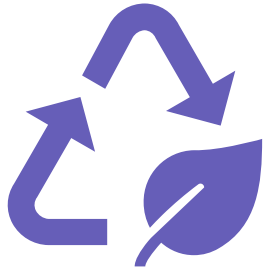
Grouping tools and responsibilities can lead to better outcomes. Agents are more likely to succeed when focused on specific tasks.

Benefits of Multi-Agent Designs



Diversity

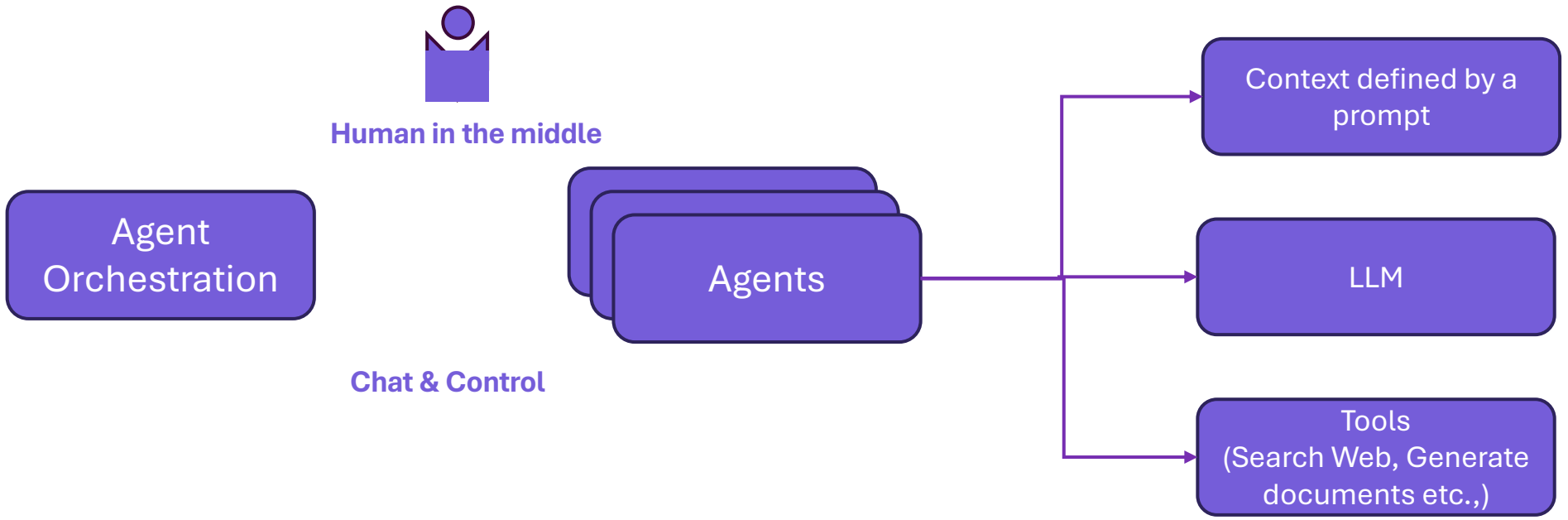
Bring in strong diversity in the agent-teams to bring in different perspectives and refine the output and avoid Hallucinations and Bias.



Reusability

Once the agents are built, there is an opportunity to reuse these agents for different use cases, and think of an ecosystem of agents, that can come together to solve the problem, with a proper orchestration framework.

Multi-Agent Architecture components



Multi-Agent Architecture components

Agents: Intelligent Agents have a very clear role, persona and context, and runs on an LLM.

Connections: How are these agents connected?

Orchestration: Orchestration defines how these agents work together (Sequential, Hierarchical, Bi-directional chat etc)

Human: We will require Human in the middle in most use cases, to help take decisions and evaluate the results.

Tools: Tools that these agents use to run specific tasks such as search the web for more information, or generate/read the document, or upload the generated code to GitHub etc.

LLM: Language models, that the agent uses for inference.

Frameworks to build Multi-Agents

OpenAI Assistant: OpenAI Assistant is one of the first frameworks to support a multi-agent architecture. This framework enables the creation of persistent, multimodal multi-agent systems that can interact with users over extended periods. Agents can access files and tools, including a Code Interpreter, and communicate with other agents to perform tasks. This is ideal for applications that require long-term collaboration/interaction

Autogen: Autogen is one of the popular emerging frameworks by Microsoft. This is an open-source framework, which also produces a very intuitive UI based development tool called Autogen Studio. for building robust multi-agent applications. It allows the creation of LLM agents that use Large Language Models for reasoning and action, which can be augmented with information from custom sources. It provides a very well-defined orchestrator-based approach towards multi-agent architecture**.**

CrewAI: CrewAI is one of the emerging frameworks, which is gaining popularity, and is being compared with Autogen. CrewAI provides a very good framework for orchestrating role-playing, autonomous AI agents. CrewAI fosters collaborative intelligence, empowering agents to work together seamlessly to tackle complex tasks. It is designed to enable AI agents to assume roles, share goals, and operate in a cohesive unit.

LangGraph: LangGraph is another very powerful and promising multi-agent framework for building stateful, multi-actor applications with LLMs, built on top of LangChain. It extends the LangChain Expression Language with the ability to coordinate multiple chains (or actors) across multiple steps of computation in a cyclic manner, inspired by Pregel and Apache Beam. LangGraph has the power of a strong community and LangChain ecosystem

References:

Autogen: https://microsoft.github.io/autogen/
LangGraph: https://python.langchain.com/docs/langgraph
CrewAI: https://www.crewai.io/
OpenAI Assistant: *https://platform.openai.com/docs/assistants*

Building Multi-Agents - Demo

LangGraph Framework

Autogen Framework

CrewAI Framework

LangGraph for Agentic Application

What does it mean to be agentic?

The system that use an LLM to decide the control flow of an application.

Examples of using an LLM to decide the control of an application :

- Using an LLM to route between two potential paths
- Using an LLM to decides which of many tools to calls
- Using an LLM to decide whether the generated answer is sufficient or more work is need.

Why LangGraph?

LangGraph has several core principles that make it most suitable framework for building agentic applications:

Controllability: LangGraph is extremely low level. This gives you a high degree of control over what the system you are building does. This is important because it is still hard to get agentic systems to work reliably and we have seen that the more control you exercise over them, the more likely it is that it will "work".

Human-in-the-loop: LangGraph comes with a built-in persistence layer as a first –class concept. This enables several different human-in-the-loop interaction patterns. Human-Agent Interaction patterns will be the new Human-Computer-Interaction and have built LangGraph with built in persistence to enable this.

Streaming First : LangGraph comes with first class support for streaming. Agentic applications often take a while to run, and so giving the user some idea of what is happening is important, and streaming is a great way to do that.

LangGraph - Graphs

At its core, LangGraph models agent workflows as graphs. You define the behavior of your agents using three key components:

State: A shared data structure that represents the current snapshot of your application. It can be any Python type but is typically a *TypedDict* or Pydantic *BaseModel*. The schema of the State will be the input schema to all Nodes and Edges in the graph. All Nodes will emit updates to the State which are then applied using the specified reducer functions.

Nodes: Python functions that encode the logic of your agents. They receive the current State as input, perform some computation or side-effect, and return an updated State.

Edges: Python functions that determine which Node to execute next based on the current State. They can be conditional branches or fixed transitions.

By composing ***Nodes*** and ***Edges,*** you can create complex, looping workflows that evolve the ***State*** overtime. The real power, though , comes from how LangGraph manages that State. Nodes and Edges are nothing more than Python functions – they can contain an LLM or just Python code.

In short: ***nodes*** do the work. ***edges*** tell what to do next.

LangGraph - Messages

Chat model can use messages, which capture different roles within a conversation.

LangChain supports various message types, including *HumanMessage*, *AIMessage*, *SystemMessage* and *ToolMessage*.

These represent a message

from the user

from chat model

for the chat model to instruct behavior

from a tool call

Each message can be supplied with few properties

content – content of the message

name – who creates the message

response_metadata – a dictionary of metadata that is often specific to each model provider

LangGraph - Tools

Tools are needed whenever you want a model to control parts of your code or call out to external APIs.

Many LLM providers support tool calling.

The tool calling interface in LangChain is simple.

You can pas any Python function into ChatModel.bind_tools()

Multi-agent systems in LangGraph

Agents as nodes

Agents can be defined as nodes in LangGraph. As any other node in the LangGraph, these agent nodes receive the graph state as an input and return an update to the state as their output.

Agents as tools

Agents can also be defined as tools in LangGraph. In this case, the orchestrator agent would use a tool-calling LLM to decide which of the agent tools to call, as well as the arguments to pass to those agents.

Communication in multi-agent systems
Schema – How agents communicate
Sequence – Controls the communication

Shared Persistent State

A significant amount of application architecture knowledge is required when we design generative AI based solutions. It becomes even more important when we are designing multi agent systems. This is probably the first time that machine learning and application architecture are coming together in a significant way. All these days, most organizations had separated application and machine learning architecture teams. Data scientists and ML engineers hardly cared about application architecture and application engineers never had to think of ML engineering process. But in the era of Generative AI, it is important to blend them together the most optimal solution architecture.

Why do we need a shared persistent state in a multi agent system? The person who asked this question comes from a data science background and the concept of a shared persistent state comes from application architecture pattern; more precisely it is one of the famous patterns in an event driven architecture. We called it event sourcing. To understand the role of a shared persistent state, let us first go back in the past to refresh our memory on Event Sourcing.

> *Event sourcing is a design pattern where state changes are logged as a sequence of events. These events are immutable and appended to a log, which serves as a single source of truth for the system.*

Shared Persistent State

A Quick Refresher on Event Sourcing:

Event sourcing is a design pattern where every state change in an application is logged as a sequence of events. These events are immutable, meaning they cannot be altered once recorded, and they are stored in an append-only log that serves as the system's single source of truth. In most applications, data is central. If data could manage itself, we wouldn't need complex applications to interact with it. Typically, we manage data through CRUD (Create, Read, Update, Delete) operations. However, this approach has a significant drawback: it only shows us the current state of the data, losing the entire history of how it evolved to that point. Event sourcing solves this problem by recording every operation and state change in an event store, allowing us to replay these events to reconstruct the current state. This capability is crucial in regulated industries where we need to audit and explain the decisions made based on the data.

Shared Persistent State

Shared Persistent State in Multi-Agent Systems:

In a multi-agent system, a shared persistent state is essentially an implementation of the event sourcing pattern, but with added functionality. Beyond just providing an audit trail, it serves as a mechanism for passing context from one agent to another, enabling seamless collaboration between the agents within the system. Additionally, it allows for the replaying of events, giving us insights into how the agents responded to specific inputs. This feature will likely become integral to LLM observability products, enabling developers to trace and optimize the behavior of multi-agent systems.

Both LlamaIndex and LangGraph have already implemented the concept of shared persistent state. In LangGraph, for example, developers can define the data structure of the shared state, while LlamaIndex manages it through a context object that persists across workflow steps.

Shared Persistent State

Designing the Shared Persistent State:

When architecting a multi-agent system, designing the structure of the shared persistent state is critical. It must be extensible, allowing for the addition of new agents without requiring a complete overhaul of the existing structure. Performance is also a key consideration; the data structure needs to be efficient for both read and write operations.

As multi-agent systems evolve, it's clear that architects have a significant role to play. While innovation is essential, we should also draw on the best practices and lessons learned from existing application and data architecture patterns. We, the architects have a lot to do in the coming days to design the applications and business processes of the future. I imagine reliving the days of "monolithic to microservices" paradigm that happened few years back. But now it will be the evolution to an **"Agency"** architecture that will bring together the semantic and syntactic world to expand the horizon of automation.

Shared Persistent State

Types of Shared Persistent State:

Blackboard System: A common architecture where agents post their findings or partial solutions on a shared board, and other agents can read and contribute.

Add examples: TODO

Distributed Databases: Agents might access distributed database that serve as shared memory where data persisted and can be queried/ updated by all agents.

Add examples: TODO

Cloud based storage: In modern systems, agents could rely on shared cloud storage services for persistent state management, ensuring durability and consistency.

Add examples: TODO

References

Introduction to LangGraph (langchain.com)

https://microsoft.github.io/autogen/

https://python.langchain.com/docs/langgraph

https://www.crewai.io/

https://platform.openai.com/docs/assistants

Demo