Multi-Agent Orchestration: uAgents vs LangChain + LangGraph

Practical Route

By - Ayush Bhardwaj
(abhard@uic.edu)
UIN - 651465377

Agenda

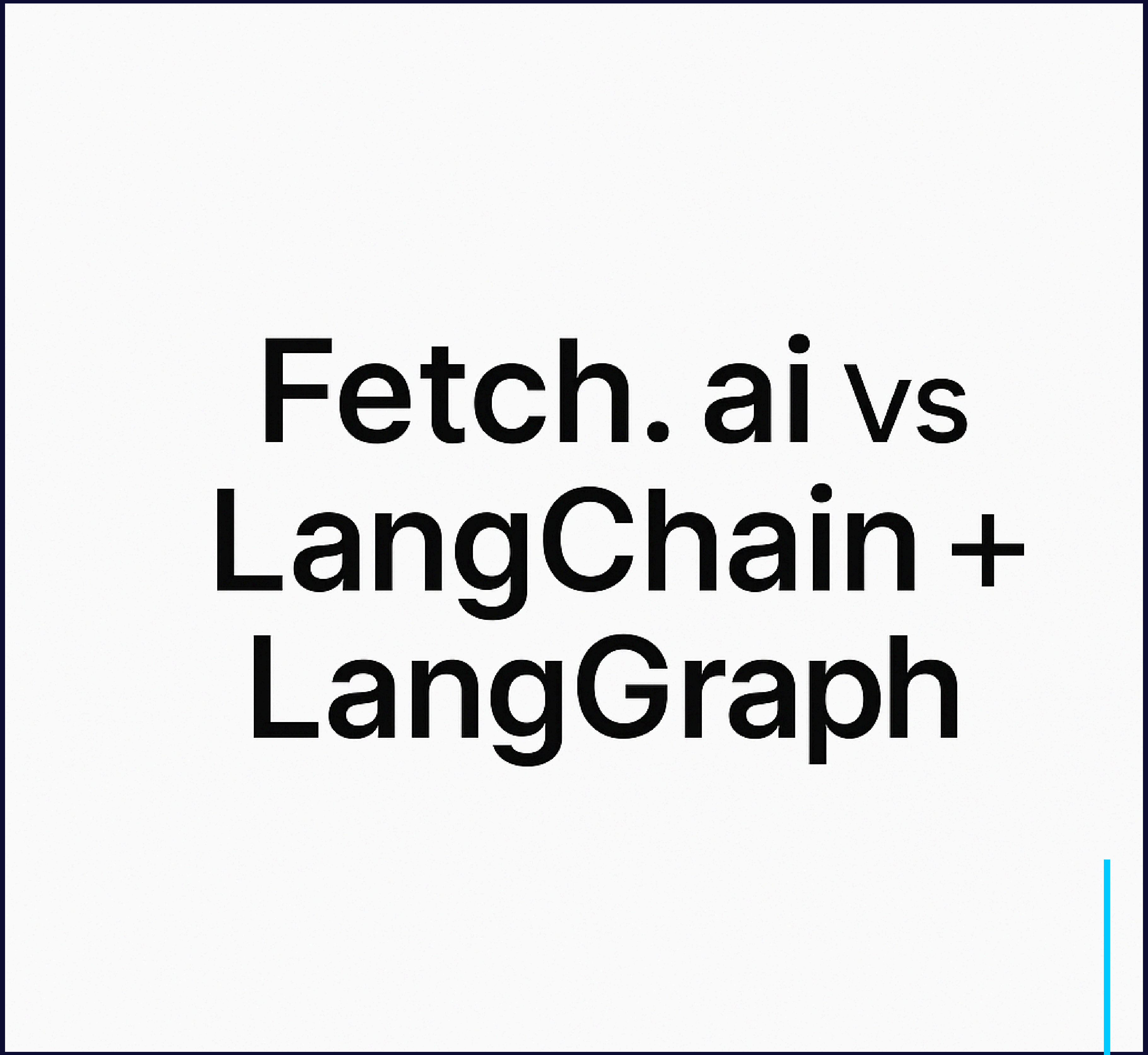
1. Problem both libraries aim to solve
2. Design decisions & philosophies
3. Feature & usability trade-offs
4. When to prefer which
5. Install & quickstart
6. Small working examples

The Core Problem

Goal: Get multiple AI components (agents/tools) to coordinate toward a shared task.

Key challenges:

- How agents communicate (message format, routing, timing)
- How to control the flow (order, branching, retries)
- Where to keep state & memory (and recover after failure)
- How to observe & audit runs (logs, traces, checkpoints)
- How to involve humans safely (approvals) and secure tool use



Fetch. ai vs LangChain + LangGraph

Our Project

uAgents (Fetch.ai)

Independent agents run as separate services and communicate using well-defined protocols (structured message schemas and turn taking rules). This makes it ideal for distributed, cross-service, or cross-organization settings where peers may live on different machines and still interoperate reliably. You get strong decoupling, natural fault isolation, and the freedom to scale by adding agents rather than centralizing everything.

LangChain + LangGraph

Orchestration happens inside a workflow graph of steps (nodes) connected by edges, with checkpoints that save progress and built-in human-in-the-loop pauses for review or approval. This design fits app-internal systems that need reproducibility, auditability, and clean state management across complex, multi-step processes. It simplifies debugging and recovery while keeping all coordination in one controlled place.

Design Philosophy (uAgents)

Mental model

A network of peers. Each agent has an address/identity and inbox. Communication follows protocols (schemas + roles).

Design choices

- Protocols define allowed message types & turns (e.g., Propose → Counter → Accept/Reject)
- Handlers react to incoming messages (event-driven)
- Transport is network-first (HTTP/p2p); agents can live anywhere
- State is local per agent; global views are composed explicitly
- Resilience via isolation (one agent crash ≠ system crash)

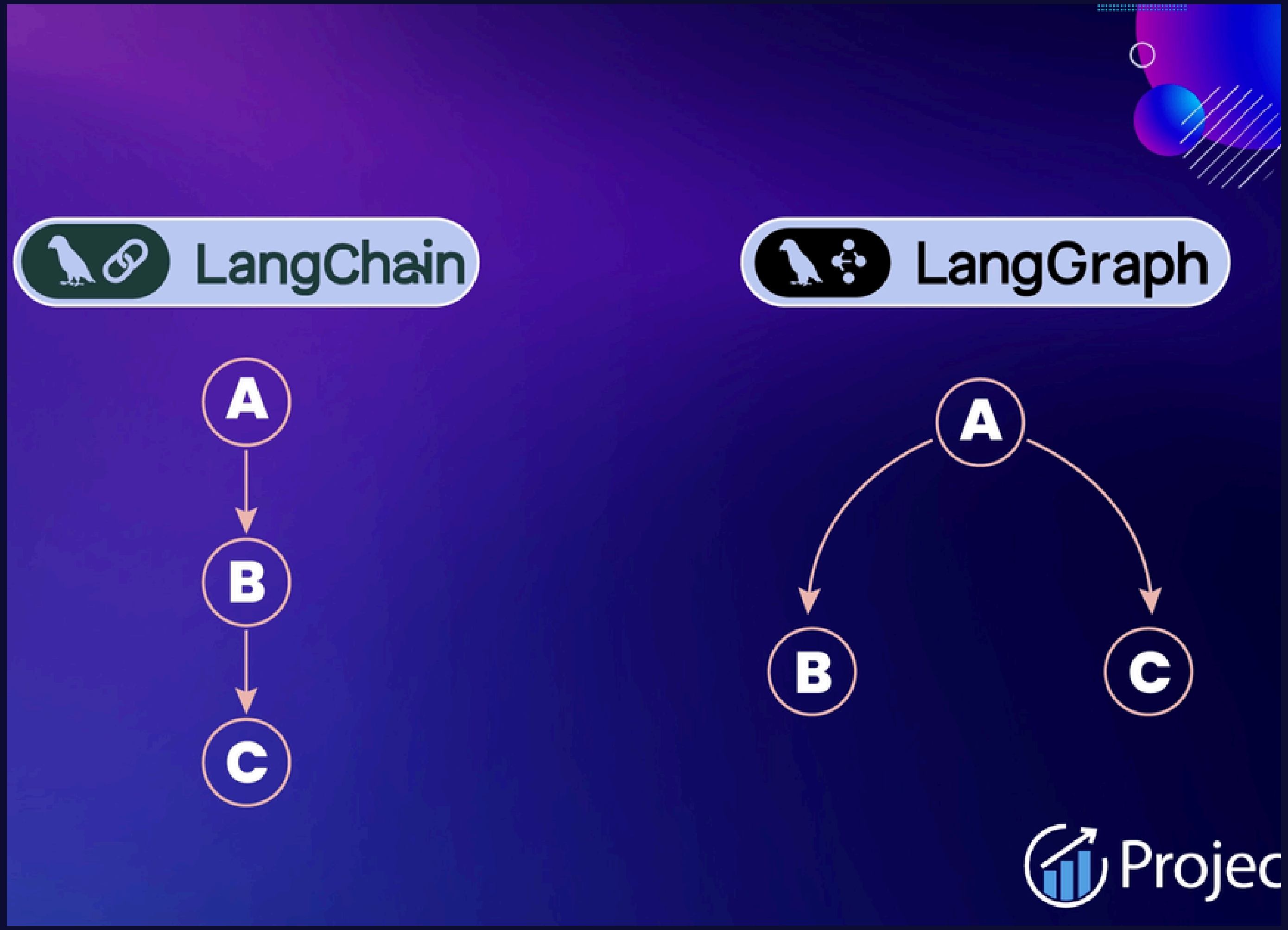
Design Philosophy (LangChain + LangGraph)

Mental model

A graph/state machine of nodes (LLMs/tools/functions) connected by edges. The framework checkpoints state and supports pause/resume with human approvals.

Design choices

- Graph expresses control flow explicitly (DAG/state machine)
- Checkpointers persist inputs/outputs for reliability
- Agents & tools are nodes; LCEL pipelines compose easily
- Human-in-the-loop is first-class (interrupt → approve → resume)
- Observability is centralized (one run history)



Feature Comparison (High-Level)

Dimension	uAgents	**LangChain + LangGraph**
Topology	Many independent services/agents	Central orchestrator with nodes/sub-agents
Communication	Protocol messages (schemas, roles)	Function I/O between nodes; tools
Control Flow	Emerges from who messages whom	Explicit graph (edges, branches, loops)
State	Local per agent	Central state with checkpoints
Reliability	Isolation; mailbox + retries	Built-in retries, timeouts, resume
Human Loop	Model human as another agent	Pause/approve/resume built-in
Observability	Per-agent logs/traces you stitch	Single run timeline/logs
Best For	Distributed/federated systems	App-internal, auditable workflows

Usability & DevEx Trade-offs

uAgents (Fetch.ai)

- Clear protocol boundaries make teams modular and let you scale across machines/orgs; versioned schemas help collaboration.
- Strong fault isolation and independent deploys, but you must own distributed tracing, correlation IDs, and shared-state views.
- Great for federated setups with per-agent ACLs/rate limits, at the cost of higher ops overhead (queues, retries, schema evolution).

LangChain + LangGraph

- Fast iteration with fewer moving parts; built-in checkpoints/retries give easy recovery and reproducibility.
- Centralized observability (one run history) and first-class human-in-the-loop make debugging and audits simple.
- Works best inside one app; less natural for cross-org networks, and large flows can become a "god-graph" unless modularized/sharded.

Usability & DevEx Trade-offs

Prefer uAgents when

- Multiple autonomous services/organizations must collaborate
- Protocol-heavy interactions (negotiation, auctions, marketplaces)
- You need network isolation and horizontal scaling by agents

Prefer LangGraph when

- One app coordinates many steps/tools with checkpoints
- You need human approvals and reproducible, auditable runs
- You want fast iteration and simpler debugging

Installing and Importing uAgents & LangGraph

uAgents (Fetch.ai)

```
pip install uagents
```

```
from uagents import Agent,
Context, Protocol, Model
```

LangChain + LangGraph

```
pip install langchain langgraph
langchain-openai
```

```
from langgraph.graph import StateGraph
from langchain_openai import ChatOpenAI
```

DEMONSTRATION

Fetch.ai
vs
LangChain +
LangGraph

Thank You For Watching

By Ayush Bhardwaj (abhard24@uic.edu)