📕 LangChain vs LangGraph

1. 🎯 Introduction

- **Playlist Context**
  - Video 1 → Difference between **Generative AI & Agentic AI**.
  - Video 2 → Detailed overview of **Agentic AI** (definition, traits, components, hiring example).
  - Video 3 (current) → Focus on **LangChain vs LangGraph**.
- **Main Goals of this Video**

1. Why does **LangGraph exist**? (What problem LangChain couldn't solve).
2. What is LangGraph? (Technical overview).
3. LangChain vs LangGraph – key differences & when to use which.

- **Prerequisites**
  - Basic knowledge of **LangChain** (components, chains, usage).
  - Suggested: watch CampusX's **LangChain playlist (first 2 videos)**.

2. ⚡ Quick Recap of LangChain

- **Definition:**
  LangChain = *open-source library to simplify building LLM-based applications*.
- **Why?**
  - LLMs are now integrated everywhere (chatbots, Chrome plugins, document Q&A).
  - But building such apps = hard → need multiple parts stitched together.
  - LangChain simplifies this via **modular building blocks**.
- **Core Components in LangChain**

1. **Models** → Unified interface to talk to any LLM (OpenAI, Claude, HuggingFace, Ollama).
2. **Prompts** → Helps design & engineer structured prompts.
3. **Retrievers** → Fetch relevant documents from vector stores/knowledge bases.
4. **Chains** → Connect components sequentially (output of one = input to next).
5. **Agents (basic)** → LLMs connected with tools (APIs, Python functions).

- **Applications possible:**
  - Chatbots, summarizers, multi-step workflows, RAG apps, simple tool-using agents.
- **Limitation noted:**
  - Works best for **linear workflows**.

3. 🏢 Example Workflow: Automated Hiring

- Uses the **same hiring scenario** as last video:
  - Create JD → Post job → Collect applicants → Shortlist → Schedule interviews → Send offers → Onboard.
- Represented via a **large detailed flowchart**.
- Key Point:
  - This **flowchart = workflow (static)**, not a true **agent** (dynamic).
  - **Workflows:** predefined paths (developer-designed).
  - **Agents:** dynamically decide steps & order themselves.

4. 🚧 Challenges of Using LangChain for Complex Workflows

The hiring workflow highlights **8 challenges**, but the video first covers 3 in depth:

Challenge 1: Control Flow Complexity

- LangChain chains = **linear only**.
- Hiring workflow = **non-linear** (loops, branches, jumps).
- Example:
  - JD approval → loop until approved.
  - If <20 applications → loop back & modify JD.
  - Branches → Yes/No conditions.
- In LangChain:
  - Must write **custom Python "glue code"** for loops & conditions.

    - Leads to **messy, unmaintainable code**.
- ❌ LangChain weak at **non-linear workflows**.
- ✅ **LangGraph Solution:**
    - Workflow represented as a **graph**.
    - Each task = **node**, flow = **edges**.
    - Supports:
        - Loops (cycle back).
        - Conditional edges (Yes → one path, No → another).
        - Complex branching & jumps.
    - **Zero glue code**, everything defined in LangGraph directly.
    - Much cleaner & scalable.

Challenge 2: State Handling

- In workflows, multiple **data points (state)** must be tracked:
    - Job description (JD).
    - JD approval status.
    - JD posted or not.
    - Number of applicants so far.
    - Shortlisted candidates & details.
    - Offers sent / accepted.
    - Onboarding status.
- In LangChain:
    - No proper state management.
    - Only has **conversation memory (text history)**.
    - Developers must **manually manage state** using dictionaries → messy, error-prone.
    - Hence LangChain workflows = **stateless**.
- ✅ **LangGraph Solution:**
    - Provides a **state object** (typed dictionary / Pydantic model).
    - State = accessible & mutable by every node.
    - Each step updates the state → automatically passed to the next node.
    - Workflows are inherently **stateful**, reliable, and easier to maintain.

Challenge 3: Event-Driven Execution

- Workflows may need to **pause & wait for triggers** before continuing.
- Hiring example:
    - Post JD → wait **7 days** → resume.
    - Modify JD → wait **48 hours** → resume.
    - Send offer → wait for candidate reply → continue.
- In LangChain:
    - Only supports **sequential execution** (no pause/resume).
    - Workaround: split into multiple chains, write custom Python for timers/triggers.
    - Again, lots of **glue code** & manual state transfer.
- ✅ **LangGraph Solution:**
    - Built for **event-driven execution**.
    - Workflow can pause, wait for external trigger, then resume from exact state.
    - Perfect for real-world async workflows.

---

5. 🔑 Key Takeaways

- **LangChain**
    - Great for simple, linear LLM apps (chatbots, summarizers, basic workflows).
    - Weak in handling **complex, non-linear, stateful, event-driven systems**.
- **LangGraph**
    - Designed to handle **complex Agentic AI workflows**.
    - Strengths:
        - Graph-based execution (nodes/edges).

    - Built-in **loops, branching, jumps**.
    - **Stateful execution**.
    - **Event-driven execution**.
- Cleaner code, less glue code, higher maintainability.