01 Why LangGraph

Understanding AI Systems and the Need for a Graph Framework

Let's explore the evolution of AI systems, the challenges of building robust AI applications, and the motivation for creating a graph-based framework like LangGraph. LangGraph continues to evolve as a low-level orchestration framework for stateful, long-running agents, emphasizing controllability, reliability, and scalability. It is trusted by companies like Klarna, Replit, and Elastic for production-grade deployments.

I. Levels of Autonomy in AI Systems

AI systems exist on a spectrum based on their level of autonomy, balancing flexibility with reliability.

- **Deterministic Code**:
  - Systems where developers write all the code without integrating Large Language Models (LLMs).
  - The input, output, and execution steps are precisely known and controlled.
  - **Highly resilient and reliable** due to complete developer control.
  - **Lacks flexibility** because everything is hard-coded.
  - *Latest Insight*: In 2025, deterministic approaches remain foundational for mission-critical systems but are increasingly augmented with LLM sub-tasks for efficiency, as seen in hybrid setups with tools like LangGraph's integration capabilities.
- **Autonomous Agents**:
  - Represent the other end of the spectrum, designed to perform tasks from start to finish with high dynamism.
  - Can define tasks, write and execute code, reorder tasks, and handle complex prompts like "make me the number one YouTuber".
  - **Extremely flexible**.
  - In reality, fully autonomous agents (e.g., AutoGPT, GPT-Engineer, Baby AGI) are not widely used in production.
  - **Lack reliability**: They are too flexible, and developers lack control because they rely too heavily on the LLM. LLMs are statistical models that can "scatter around" and not produce desired outputs.
  - *2025 Update*: Recent critiques highlight ongoing issues like high memory usage and breaking changes in frameworks, pushing developers toward controlled autonomy. LangGraph addresses this by enabling "agent engineers" to design reliable multi-agent systems.

II. Evolving AI System Design with LLMs

Between deterministic code and fully autonomous agents, several approaches integrate LLMs while maintaining varying degrees of control. As of 2025, chaining and routing have advanced with tools like LangGraph's new functional API and dynamic model selection.

- **Integrating LLMs into Deterministic Code**:
  - Developers still write and control the code flow, knowing exactly what will be executed.
  - The LLM is used for specific sub-tasks, such as summarization, information extraction, or entity extraction, controlling only one output within the flow.

    - This approach **gains significant flexibility** from LLM generation while retaining most developer control.
    - *Example*: Basic RAG pipelines now incorporate LangGraph's persistence for stateful sub-tasks.
- **Chaining LLM Calls**:
    - This involves taking the output of one LLM call and feeding it as input to another, composing multiple calls.
    - **Example: Retrieval Augmented Generation (RAG) Flow**:
        - An initial LLM receives a question.
        - Embeddings are used to retrieve relevant documents.
        - The original prompt is augmented with these documents.
        - Everything is sent to a final LLM to generate the answer.
    - Chaining allows LLMs to determine outputs across multiple steps, enabling the creation of more complex systems.
    - *2025 Enhancement*: LangGraph's v0.6 introduces durability mode, ensuring chains persist through failures for long-running tasks.
- **LLM Router**:
    - A specialized type of chain where the LLM's reasoning power is used to decide the execution path.
    - The LLM determines which steps to take, for example, whether to search a database or the web.
    - **Increases flexibility** as the LLM dictates the control flow for the first time.
    - **Key characteristic**: LLM routers **do not have cycles**. Systems below a certain threshold, which introduce cycles, are considered "agents".
    - *Recent Development*: With LangGraph Supervisor (released March 2025), routers can now scale to hierarchical multi-agent setups.

III. Defining Agents and Agentic Applications

The definition of an "agent" or "agentic application" is often debated and lacks a universally agreed-upon formal definition. In 2025, the focus has shifted to "agent engineers" who prioritize observability and controllability.

- **Core Definition of an Agent**:
    - At its most basic level, an agent is a **control flow that an LLM decides where to go**.
    - The LLM's reasoning power is used to determine the path within the flow.
    - *Update*: Agents now emphasize state persistence and human-in-the-loop, as enabled by LangGraph Platform (GA May 2025).
- **Distinguishing Agents from Chains/Routers**:
    - The main difference is the presence of **cycles**.
    - Chains are generally one-directional (moving from left to right).
    - Agents, however, incorporate cycles, which are crucial for their agentic properties.
    - *Pros/Cons*: Cycles enable reasoning but can lead to loops; LangGraph mitigates this with enhanced type safety and debugging via LangSmith.
- **Function Calling in Agents:**
    - Modern agents often use **function calling** to decide which steps to take.
    - Developers send the LLM descriptions of available tools (functions), including their arguments, purpose, and return values.

    - The LLM can then instruct the system to invoke specific functions with chosen arguments based on the query.
    - *2025 Example*: Dynamic tool selection in LangGraph v0.6 allows agents to adapt tools on-the-fly, improving efficiency in production.

IV. The React Paper and its Influence

The React paper introduced a fundamental design for agents that significantly impacted the industry. Its principles remain core, but 2025 implementations address reliability gaps.

- **React Algorithm (Basic Agent Design)**:
    - Start.
    - The LLM decides whether to use a tool (e.g., API call, database query).
    - If a tool is chosen, it's executed with LLM-selected arguments.
    - The answer from the tool is fed back to the LLM.
    - The LLM then decides whether to use another tool or return the final answer to the user.
    - *Modern Twist*: LangGraph builds on ReAct with subgraphs for focused agents, avoiding over-flexibility.
- **Challenges with Highly Flexible Agents**:
    - While agents inspired by React (and frameworks like LangChain's implementation) are very flexible, allowing many permutations of tool invocations, this **flexibility can lead to reliability issues**.
    - A common problem is the agent getting stuck in **endless loops**, repeatedly invoking the same tool.
    - Reasons for such issues include:
        - Incorrect tool definitions.
        - The LLM being non-deterministic.
        - The LLM choosing incorrect tools or arguments.
        - The LLM hallucinating non-existent tools.
    - This highlights a critical trade-off: **flexible agents like React are not always reliable enough for production systems.**
    - *2025 Insights*: Recent discussions note LangGraph's advantages over alternatives like CrewAI in performance and memory management for production.

V. LangGraph: Bridging the Gap between Flexibility and Reliability

LangGraph was created to address the gap between highly flexible but unreliable autonomous agents and more controlled, less flexible systems. As of 2025, it has matured with features like the LangGraph Platform for scalable deployment.

- **Core Motivation**: To achieve systems that are flexible yet much more reliable, suitable for production environments.
    - *Why Now?*: The rise of multi-agent systems demands orchestration; LangGraph provides low-level control without hidden prompts.
- **The LangGraph Approach**:
    - **Reduces LLM freedom by one dimension**: Instead of full autonomy, the LLM's choices are scoped.
    - **Software as a Graph/State Machine**: LangGraph represents agentic software as a graph with nodes and edges, acting as a state machine that can have cycles. This gives applications agentic

properties, making them appear to reason and think.
  - **Developer Control**: Developers explicitly define and control the overall flow of the system.
  - **LLM's Role:** The LLM plays a crucial role by deciding where to go within the developer-defined flow using conditional branching.
  - **Benefit**: By structuring the flow and imposing control, LangGraph significantly **improves reliability and resilience.**
  - *Key Components (2025)*:
    - **State:** Shared context across nodes/edges.
    - **Nodes:** Functional units (e.g., LLM calls).
    - **Edges**: Define paths, including conditional ones.
    - **Reducers:** Merge state updates.
    - **Persistence/Checkpointing**: Save/resume executions.
    - **Super-steps**: Full cycles for multi-agent coordination.
- **Why a Specialized Graph Framework?**
  - While other graph frameworks exist (e.g., Airflow, NetworkX), LangGraph is **opinionated specifically for agentic applications.**
  - It offers **built-in features** crucial for agent development:
    - Controllability.
    - Running nodes in parallel.
    - Conditional branching leveraging LLMs.
    - Built-in persistence to store the graph's state and execution history.
    - Support for human-in-the-loop flows, allowing human feedback to calibrate execution.
    - Time traveling (replaying past executions).
    - Debugging and tracing tools (integrated with LangSmith).
    - *New in 2025*: Context API for runtime injection, LangGraph Supervisor for hierarchies, observability tailored for agents.
  - **Flexibility in Code**: Allows integration of any code, not just LangChain code.
  - **Natural Representation**: Agentic application papers often illustrate solutions as graphs, making this approach intuitive, readable, maintainable, testable, and monitorable.
  - *Real-World Examples*: Used in health agents (e.g., vital tracking with Gemma 3), research agents, and ambient agents (via LangChain Academy courses).
- **State Management in LangGraph**:
  - As a state machine, LangGraph requires a **shared state.**
  - This state is shared across nodes and edges, storing intermediate results and providing useful information to the LLM for its decisions within the flow.
  - *Pros/Cons (2025)*: Excels in memory persistence but can face high usage in large chains; alternatives like Pydantic AI may suit simpler workflows.
- **Comparisons and Alternatives**:

Framework	Strengths	Weaknesses	Best For
LangGraph	Stateful cycles, controllability, scalability (via Platform)	Potential breaking updates, high memory in complex setups	Multi-agent, production-grade agents

Framework	Strengths	Weaknesses	Best For
CrewAI	Simple multi-agent collaboration	Poor performance in production	Quick prototypes
AutoGen	Early pioneer in simple agents	Experimental, long beta periods	Microsoft ecosystem integrations
n8n	Visual workflows	Less LLM-focused	No-code automation

Latest Resources:

- Official Docs: LangGraph Guide
- Blog: LangGraph Release Recap

By leveraging LangGraph, developers can build agents that are not only flexible but also production-ready, addressing today's demands for reliable AI orchestration.