Agent Frameworks and Tools

Week 6: LangGraph, AutoGen, and Production Patterns

PhD Course in Agentic Artificial Intelligence

12-Week Research-Level Course

Learning Objectives

Bloom's Taxonomy Levels Covered

- **Remember**: Define state machines (flow control), graphs, orchestration
- **Understand**: Explain LangGraph vs LangChain architecture differences
- **Apply**: Implement a stateful agent using LangGraph
- **Analyze**: Compare framework trade-offs for different use cases
- **Evaluate**: Assess production readiness of agent frameworks
- **Create**: Design a custom agent orchestration system

By end of lecture, you will understand how to select and use production agent frameworks.

Agent Framework Landscape

First Generation (2023)

- **LangChain**: Chain-based composition, extensive tool library
- **LlamaIndex**: Data-centric, RAG-focused framework

Second Generation (2024)

- **LangGraph**: Graph-based state machines, cycles allowed
- **AutoGen**: Multi-agent conversations, code execution
- **CrewAI**: Role-based multi-agent orchestration

Enterprise Frameworks

- **Semantic Kernel** (Microsoft): Enterprise integration
- **Haystack** (deepset): Production NLP pipelines

Framework choice depends on: complexity, multi-agent needs, production requirements.

When to Use vs. Avoid Frameworks

Frameworks Add Value When

- Rapid prototyping of agent architectures
- State persistence and checkpointing requirements
- Human-in-the-loop interactions needed
- Multi-agent coordination patterns required

Build Custom When

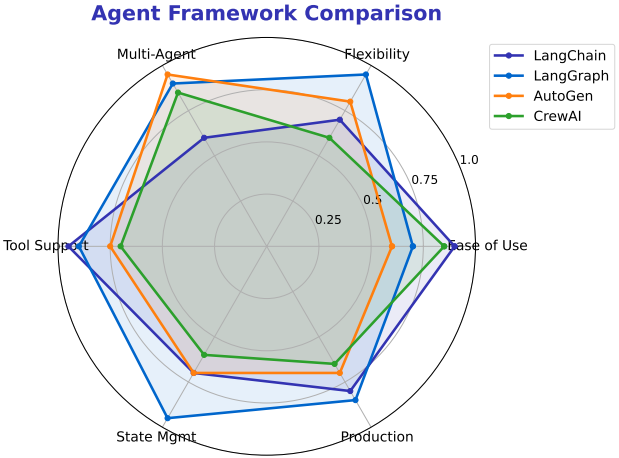
- Simple single-agent pipelines (overhead not justified)
- Extreme latency requirements (<100ms responses)
- Proprietary orchestration logic
- Framework abstractions obscure debugging

Decision Heuristic: Complexity of state management determines framework need

---

Start simple – add framework when state/coordination becomes bottleneck.

Framework Comparison



LangGraph excels at flexibility and state management; CrewAI at ease of use.

LangGraph: Graph-Based Agents

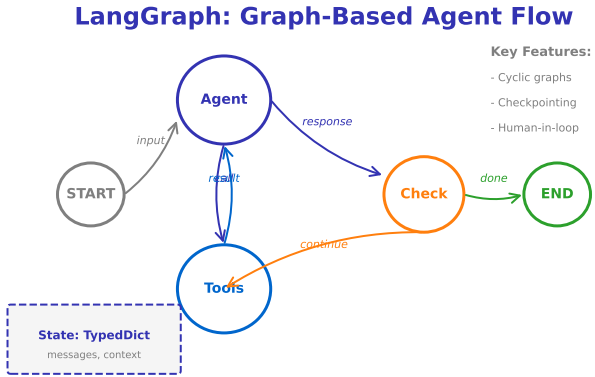
Core Concepts

- **StateGraph**: Define nodes and edges as a directed graph
- **Nodes**: Functions that modify state
- **Edges**: Conditional routing between nodes
- **Cycles**: Enable iterative refinement loops

Key Advantages

- Explicit control flow visualization
- Built-in checkpointing (state snapshots) and persistence
- Human-in-the-loop interrupts
- Streaming support for real-time output

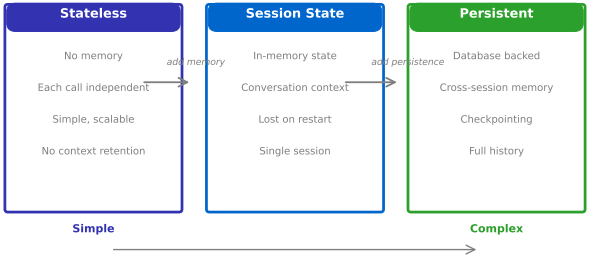
LangGraph treats agent logic as a stateful graph, not a linear chain.



The graph cycles through Agent-Tools-Check until task completion.

State Management in Agents

State Management Patterns



Production agents typically require persistent state for reliability.

Defining Agent State

TypedDict State Schema

- Define state as Python TypedDict for type safety
- Messages list tracks conversation history
- Custom fields for domain-specific context

State Updates

- Nodes return partial state updates
- Reducers merge updates (append, replace, custom)
- Checkpointing persists state after each node

Persistence Options

- MemorySaver: In-memory (development)
- SqliteSaver: SQLite database
- PostgresSaver: PostgreSQL (production)

State schema determines what information flows through the agent graph.

AutoGen: Multi-Agent Conversations

Core Abstraction

- Agents communicate through message passing
- ConversableAgent: Base class for all agents
- UserProxyAgent: Represents human user
- AssistantAgent: LLM-powered responder

Key Features

- **Code Execution**: Docker (containerized)/local code sandbox
- **Group Chat**: Multi-agent discussions
- **Nested Chats**: Hierarchical conversations
- **Custom Replies**: Programmable response logic

AutoGen excels at collaborative multi-agent problem solving.

CrewAI: Role-Based Orchestration

Core Abstraction

- Agents defined by role, goal, backstory (persona)
- Tasks assigned with expected output specifications
- Crew orchestrates agent collaboration

Key Features

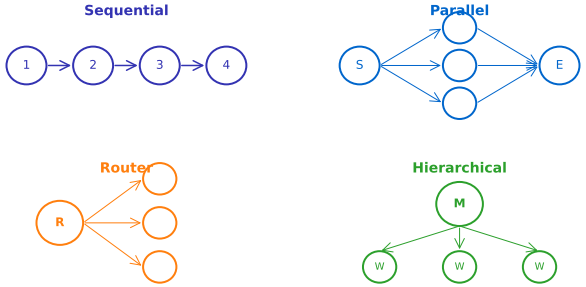
- Role-based reasoning (agents "think" as their role)
- Process types: sequential, hierarchical, consensual
- Memory sharing between crew members
- Tool delegation to specialized agents

Comparison to AutoGen

- CrewAI: Declarative role definitions, simpler setup
- AutoGen: More flexible, lower-level control

CrewAI excels at well-defined team workflows with clear role boundaries.

Orchestration Patterns



Choose pattern based on task parallelizability and coordination needs.

Production Deployment

Reliability

- Retry logic with exponential backoff
- Graceful degradation on API failures
- Timeout handling for long-running tasks

Observability

- LangSmith/LangFuse (trace visualization) for tracing
- Structured logging with correlation IDs
- Metrics: latency, token usage, success rate

Security

- Sandboxed code execution
- Input validation and sanitization
- Rate limiting and cost controls

Production agents require robust error handling and monitoring.

Framework Anti-Patterns

Common Mistakes

- Over-engineering: Using multi-agent for single-agent problems
- State bloat: Storing everything in state graph
- Missing error handling: Assuming LLM calls always succeed

Debugging Challenges

- Non-deterministic execution makes reproduction hard
- Trace visualization essential (LangSmith, LangFuse)
- Log correlation IDs for distributed agent systems

Performance Pitfalls

- Synchronous tool calls when parallel is possible
- No caching of repeated LLM calls
- Unbounded context growth

Most production issues stem from error handling and observability gaps.

Choosing a Framework

Use Case	Recommended	Reason
Simple RAG chatbot	LangChain	Easy setup
Complex workflows	LangGraph	Graph control
Multi-agent debate	AutoGen	Conversation
Role-based teams	CrewAI	Agent roles
Enterprise apps	Semantic Kernel	.NET/Azure
Custom pipeline	Build own	Full control

Decision Criteria

- Complexity of agent logic and state management
- Need for multi-agent coordination
- Production requirements (observability, persistence)

Start simple (LangChain), migrate to LangGraph when you need cycles/state.

Required Readings

This Week

- LangGraph Documentation: `langchain-ai.github.io/langgraph`
- Wu et al. (2023). "AutoGen: Enabling Next-Gen LLM Applications." arXiv:2308.08155

Supplementary

- Wooldridge & Jennings (1995). "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review*.
- CrewAI Documentation: `docs.crewai.com`
- Qiao et al. (2024). "TaskWeaver: A Code-First Agent Framework." arXiv:2311.17541

Focus on LangGraph docs – hands-on implementation is essential.

Summary and Key Takeaways

Key Concepts

- **LangGraph**: Graph-based state machines with cycles
- **AutoGen**: Conversational multi-agent framework
- **State**: TypedDict schema with persistence
- **Patterns**: Sequential, parallel, router, hierarchical

Production Checklist

- Checkpointing enabled
- Error handling and retries
- Observability instrumentation

Next Week

- Advanced RAG: Self-RAG, CRAG, Agentic RAG

Framework mastery = understanding state management + orchestration patterns.