AegisTwin vs LangChain / AutoGen / CrewAI

Date: 2026-05-25

Buyers ask this every time, so here's an honest answer.

The short version: **AegisTwin is not a framework competitor.** It's a runtime layer. You could (and people do) run a LangChain agent inside AegisTwin. The comparison is about which problems each tool solves first.

Capability	LangChain	AutoGen	CrewAI	AegisTwin
LLM provider abstraction	✅ Primary focus	✅	✅	➖ Out of scope
Tool / function calling	✅ Extensive	✅	✅	➖ Out of scope
Multi-agent orchestration	⚠️ Via LangGraph	✅ Primary focus	✅ Primary focus	➖ Out of scope
Deterministic replay (record + re-execute byte-for-byte)	❌	❌	❌	✅ **Primary focus**
Policy gates (deny actions pre-execution with audit log)	❌	❌	❌	✅ **Primary focus**
Typed event log (every module-to-module call captured)	❌	❌	❌	✅ **Primary focus**
Episodic + semantic + procedural memory primitives	⚠️ Vector store wrapping	❌	❌	✅
Observability stack (Grafana/Prometheus/Jaeger)	❌ DIY	❌ DIY	❌ DIY	✅ Included
Production-ready Helm chart	❌	❌	❌	✅
Cite-able governance audit trail for SOC 2 / ISO 27001	❌	❌	❌	✅

When you want a framework, not AegisTwin

- You haven't written an agent yet → start with LangChain or CrewAI.
- You want pre-built chains and prompts → LangChain.
- You want multi-agent conversation patterns → AutoGen.
- You want role-based agent crews → CrewAI.

When you want AegisTwin

- Your agent works in development but you can't reproduce a bug from production. **Replay solves this.**
- Compliance / security asked "what did the agent do and why?" and you don't have an answer. **Event log + audit trail.**
- An agent took an action it shouldn't have, and "ask the LLM nicely not to" is not a security control. **Policy gates.**
- You're shipping AI into a regulated industry (healthcare, finance, legal) and need a defensible audit posture. **All three of the above.**

Concrete numbers

From `BENCHMARKS.md` (run on the actual code):

- Replay: 110,000+ events/sec verified — a 10k-event agent run replays in <90 ms.
- Policy gate overhead: ~30 µs per check at 10 policies — invisible next to LLM latency.
- Event bus: 50k–90k events/sec — orders of magnitude headroom over realistic agent workloads.

LangChain / AutoGen / CrewAI publish no runtime benchmarks because they're not in the runtime business. That's the gap AegisTwin fills.

Can I use AegisTwin with LangChain?

Yes. AegisTwin's SDK wraps any callable. The pattern is:

```
from aegistwin import Runtime, policy_gate
from langchain.agents import AgentExecutor
```

```
runtime = Runtime()

@runtime.run
def execute_langchain_agent(query: str):
    agent = AgentExecutor(...)
    return agent.invoke({"input": query})
```

Every LangChain tool call now flows through AegisTwin's event bus, gets policy-checked, and is recorded for replay. You keep LangChain's ecosystem; you gain AegisTwin's runtime properties.

Honest limitations

- AegisTwin does not abstract LLM providers. Bring your own (OpenAI, Anthropic, Bedrock, Ollama).
- AegisTwin does not ship pre-built agents. If you want prompts + chains, use LangChain on top.
- Deterministic replay requires that your tool implementations be either deterministic or that their outputs are recorded into the event log. Non-deterministic side-effecting tools need wrapping.
- AegisTwin is Python-first. The TypeScript SDK is for frontend integration with the control plane, not for writing agents.

Who's already using it

Pre-revenue at time of writing. The benchmark numbers, governance model, and replay primitives have been validated against the 94-test suite included in the repo. This is sold as **code + IP**, not as a managed service.