Reading a Graph's Committed State From Its Own Checkpoints

A Fathom coherence read run against LangGraph

Embedded Risk Analytics · A Fathom Case Study · August 2026

Fathom reads the committed state of a multi-agent system from the system's own persistence layer and reports where that shared state has been left incoherent. This study runs that read against LangGraph, the most widely adopted runtime for long-running, stateful agents. LangGraph persists every value an agent commits as an ordered lineage of checkpoints, exposed through get_state_history. We reconstructed the graph's committed record from that lineage, using only the checkpoints the runtime already writes and no ground truth.

One result stands out. Across three coherent runs and more than one hundred and seventy checkpoints the read stayed silent, and on a single parallel super-step where two agents wrote the shared record at once, LangGraph merged both writes through its channel reducer with no consistency check and the read recovered the record that was left citing the old key.

The substrate the read sits on

LangGraph runs an agent, or a supervisor or swarm of agents, over a graph whose state is a set of channels merged by reducers and persisted, per thread, by a checkpointer. get_state_history exposes the full ordered lineage of state snapshots, each carrying the channel values at that step and a link to its parent. That lineage is a faithful, queryable record of everything the graph committed and when, which makes it the natural input to an offline coherence read. The read adds no code to the runtime. It reads the checkpoints and reconstructs the committed record.

How the read reconstructs committed state without an oracle

The read folds the checkpoint lineage into a single committed record, seeded from the graph's known initial state, taking each checkpoint as the committed result of a super-step. It then runs two checks. The residual scan reports any sub-record still carrying the old key after the graph has finished. The authored-contradiction check flags a written record whose content references a key the graph has already renamed away, judged on the written delta. The read looks only at the lineage. It inspects neither the model nor the runtime, and the committed_state instrument it calls is imported and left unchanged.

Coherent graphs stay silent

We ran one coupled task, renaming a key across interdependent sub-records, under three coordination shapes of rising difficulty: a sequential supervisor at five records, a swarm of two alternating agents at twenty records, and an LLM manager that had to track its own progress and decide when the work was done at twenty records. Every run renamed the whole record, and the read stayed silent across more than one hundred and seventy checkpoints, including the one-hundred-and-twenty-three-checkpoint manager run. Fathom raised nothing on a coherent graph, which is the specificity an instrument earns before its alarms carry weight.

A merged checkpoint is not yet a coherent one

LangGraph runs the nodes of a super-step in parallel on the same input snapshot, so two agents in one step each read the state as it stood at the start of the step and neither sees the other's write. We placed two agents in a single super-step. One renamed the definition record to the new key. The other, still reading the pre-rename snapshot, reported the key it saw and authored a second record that cited the old key. LangGraph merged both writes through the channel reducer and wrote one checkpoint. The committed state ended internally inconsistent: the definition record renamed, the second record citing the key the graph no longer used. The read recovered the inconsistent record from the lineage.

The detail worth stating came from the runtime itself. When the two agents wrote a channel that carries no reducer, LangGraph raised an error and refused the step. It guards a plain channel with a hard error, and it merges a reducer channel with no coherence check. The channels that accumulate real agent state are reducer channels, which is where a conflicting pair of writes is merged and stored with nothing to flag it. LangGraph's own documentation states the position plainly: state merging is purely mechanical, and the framework applies reducer functions without validating logical consistency between state keys.

Records	Coordination	Checkpoints	Committed-state read
5	Sequential supervisor	13	Silent (clean)
20	Swarm, two agents	43	Silent (clean)
20	LLM manager	123	Silent (clean)
2	Parallel super-step (fan-out)	3	r1 flagged, coherence failure

Table 1. Three coherent graphs of rising coordination difficulty, each read silent, and one parallel super-step where two agents wrote the shared record at once. Every read is taken from LangGraph's own checkpoint lineage.

Why this is the failure that scales

The break here is structural, so it reproduces on a strong model. It also grows with exactly the shapes LangGraph is scaling into: parallel fan-out, supervisor and swarm multi-agent graphs, and sub-agent delegation, where several agents commit to shared state and the cost of keeping that state consistent competes with the work itself. This is the multi-agent form of reflexive burden. The committed-state read separates that burden from the difficulty of the task, and it places the break on the exact record and step where two commitments collided.

Where the instrument sits

Fathom sits beside the orchestration. It reads the checkpoints LangGraph already writes and reconstructs committed state offline, adding no model internals and changing nothing about how the graph runs. It reports whether the work the graph committed is coherent and, in the hosted form, decomposes the residual coherence cost into task difficulty and reflexive burden. This study demonstrates the read on a live third-party runtime, and the information-theoretic decomposition and scoring stay behind the hosted instrument.

The read ran against LangGraph unmodified, over the checkpoints the runtime already writes. A graph can persist every checkpoint faithfully and still leave its shared record incoherent, and the committed-state read is the check that tells the two apart.