

Chronicle: Cut-Point Replay for Regression Testing of LLM Agents

Tisha Chawla* Susheem Koul*

Microsoft

tisha.chawla2020@vitalum.ac.in

f2015347p@alumni.bits-pilani.ac.in

Abstract

Large language model responses are non-deterministic, so failures in LLM agents are hard to reproduce: a failure depends on inference that is not bitwise reproducible (Ati et al., 2025), on tools that read changing state, and on a multi-step trajectory that a re-run rarely repeats. Record-and-replay makes a run reproducible, but existing agent tooling records runs only to trace or score them, not to test a code change against them. We present Chronicle, which records an agent run at its non-deterministic boundaries as immutable envelopes and replays it from the record. Its central operation, *cut-point replay*, serves a chosen subset of boundaries from the record and executes the complementary subset live with new code, turning a recorded incident into a regression test that runs in continuous integration. On a benchmark of 6 recorded failures with simulated model boundaries, recording adds 23 μ s per crossing (0.008% of an assumed 300 ms model call), full replay issues zero model calls and is bit-stable across 20 repetitions, and cut-point tests fail on faulty code and pass on guarded and benign changes for all 6 incidents. In a mutation study of the guarded tools, cut-point tests catch every mutant that lets the recorded unsafe action through, while a baseline that stubs every boundary, using the same assertion, catches none. Chronicle and the benchmark are publicly available at <https://github.com/theagentplane/chronicle>.

1 Introduction

Large language model (LLM) agents take consequential actions through tool calls, such as issuing refunds, submitting trades, or deleting files (Schick et al., 2023; Yao et al., 2023). When such an action is wrong, because the surrounding code misses an edge case or the task is underspecified, correcting

it requires first reproducing the failure. This is often the hardest step: a failure is seldom a single wrong output but a *trajectory* of model calls, tool calls, and routing decisions (Cemri et al., 2025). Re-executing the agent rarely reproduces that trajectory. LLM inference is not bitwise reproducible even at temperature zero (Ati et al., 2025); tools read external state that has since changed (Yao et al., 2024); and retries and routing change how many times a step runs (Cemri et al., 2025). The result is *flakiness*: identical code passes or fails depending on nondeterministic conditions and execution order (Lam et al., 2020; Hashemi et al., 2025), so re-execution alone cannot turn an incident into a repeatable test. Existing agent infrastructure observes runs but does not make them testable: tracing records what happened (Arize AI, 2026) and evaluation frameworks score whether an output is acceptable (promptfoo contributors, 2026; Zheng et al., 2023), but neither lets a developer change one component of a recorded run and check whether the change fixes the failure while everything else is held fixed. We present Chronicle, which supplies exactly that operation. This paper contributes (i) *cut-point replay*, which runs new code at a chosen subset of boundaries while serving the rest from the record, producing a committed regression test from the incident, with each stubbed boundary guarded by a per-name call-count check; and (ii) an implementation and a benchmark of 6 recorded failures on which full replay makes no model call and reproduces each run identically across 20 repetitions, while cut-point tests flag every unguarded incident, accept its guarded fix and benign edits, and kill mutants of the fix that a stub-every-boundary baseline cannot.

2 Related Work

Chronicle relates to three lines of work and differs from each in the purpose for which replay is used.

*Equal contribution.

Record and replay for agents. Deterministic record-replay is long established for debugging general programs, where an entire execution is captured and faithfully reproduced (O’Callahan et al., 2017). Chronicle instead replays at semantic boundaries and runs new code at chosen ones, so replay tests a fix rather than reproduces a run. Prior systems record runs to reuse successful agent behavior (Feng et al., 2025) or to resume a durable workflow after a failure (Temporal Technologies, 2026); Temporal’s replay tests also re-run changed workflow code against a recorded event history, and Chronicle adapts this selective-replay idea to agent boundaries, with indexed crossings and a freely chosen live set. Graph frameworks checkpoint state to permit re-entry at a previously executed node (LangChain, 2026). These systems act *during* a run, to resume or steer it in flight; a workflow resumed after a human-in-the-loop pause continues forward and does not re-issue the model calls it already completed. Chronicle instead operates *after* a run has finished: it replays a recorded trace, serving stubbed crossings from the record and running the live subset with new code, so a candidate fix is evaluated against the past incident rather than by steering a live run.

Testing of agents. Recent work adapts software-testing techniques such as traces, mocks, and assertions to agents (Kohl et al., 2025). A documented limitation of heavy mocking is that mock-heavy tests can be less effective at validating real interactions (Hora and Robbes, 2026). Chronicle replaces hand-written mocks with envelopes drawn from a recorded run and lets any subset of boundaries run live, so a tool gate or router change is exercised against the recorded trajectory rather than against invented stubs.

Failure attribution. A related line of work intervenes in a recorded run (rewinding, editing, or otherwise modifying a crossing) to attribute or localize a failure (Shah, 2026; Lin et al., 2026; Ma et al., 2025). Those methods characterize the cause of a past failure by observing a modified crossing, whereas Chronicle executes new code at the crossing and records the outcome as a regression test. Other tools detect and diagnose faults from traces without cut-point execution of new agent code (Balusu, 2026; Ou et al., 2025; Deshpande et al., 2025; Kang et al., 2026), and LLM-based judges assess semantic quality subject to documented limitations in agreement and bias (Norman

et al., 2026); Chronicle ships an advisory LLM-as-judge for non-structural properties but does not evaluate judge reliability here.

The agents considered in this work employ standard reason-and-act loops with tool use (Yao et al., 2023; Schick et al., 2023). In contrast to hand-authored agent benchmarks and evaluation environments (Yao et al., 2024; Debenedetti et al., 2024; Zhou et al., 2024), the scenarios in our benchmark are derived from recorded failures rather than constructed to evaluate capability or attack surface.

3 Chronicle

Chronicle records a run, replays it from the record, and tests the replay with assertions. Its unit is the *boundary*: a point where the agent calls the model, calls a tool, or makes a routing decision, which is exactly where a rerun can diverge. Chronicle adapts selective replay to agents: any subset of boundaries can run live while the rest are served from a real recorded run rather than from hand-written mocks, so a recorded incident becomes a deterministic test of a chosen change.

3.1 Record

Developers mark boundaries with a one-line annotation (Figure 1); a model client or a set of LangGraph nodes can be instrumented with a single call. Each execution of a boundary, a *crossing*, is saved as an immutable *envelope* holding its input, its output, and the metadata needed to detect drift, such as the model version and sampling parameters. Crossings are addressed by boundary name and occurrence, so a boundary crossed three times in a loop yields agent [1], agent [2], and agent [3] (Figure 5). Recording is transparent: it changes no return value or exception, redacts secrets and volatile fields before storage, and emits standard OpenTelemetry spans. An envelope stores a boundary’s input and output, not the work inside it, so replaying it is faithful as long as its output depends only on that recorded input; a boundary that reads hidden state, such as a clock or a database, is the exception.

3.2 Replay

In *full replay*, every boundary returns its recorded output, so the run is reproduced exactly with no model call (Figure 3). In *cut-point replay*, a chosen subset of crossings run live with new code while the rest are served from the record (Figure 2); the

```
@boundary("place_order", kind="tool")
def place_order(symbol, qty): ...

plan = (ReplayPlan()
        .stub("agent", 1)      # from record
        .live("place_order", 1) # cut-point
        .live("agent", 2))
assert session.captured_result(
    "place_order", 1)["blocked"]
```

Figure 1: Recording annotation and a cut-point plan: the same @boundary records in production and serves its recorded output in replay; any subset of crossings may run live.

live subset is arbitrary (e.g., run a tool gate live while stubbing the model; Figure 4 in Appendix A). Full replay fixes every output and so tests the deterministic glue code between boundaries against real recorded inputs; cut-point replay tests a change together with its consequences, reproducing the stubbed lead-up without re-running it and executing the live subset forward. Chronicle serves the k th crossing of a boundary its k th recorded envelope, so if a stubbed boundary is crossed more or fewer times than recorded (an extra loop, a dropped retry) lookup fails and replay raises. This per-name count check does not detect a reordering that preserves each name’s count. An order-sensitive digest over the stubbed crossings would close this gap cheaply; it is not part of the release evaluated here. The guarantee is conditional on coverage: an unmarked non-deterministic call runs live and escapes the check.

3.3 Test

A structural assertion checks what the agent did (which tool, which arguments, whether a guarded action was refused); on full replay it is deterministic and needs no model call, so a recorded incident and one assertion become a CI regression test at no cost, and on cut-point replay the same assertion checks the fix on the live subset. For non-structural properties (faithfulness, safety) Chronicle ships an advisory LLM-as-judge (Zheng et al., 2023); its reliability is not evaluated here (Norman et al., 2026).

4 The Incident Benchmark

We release a benchmark of 6 recorded incidents, each a small agent (model, then tool, then model) in which an unguarded tool produces an unsafe result and a guarded version corrects it: a refund sized to an order identifier, a currency mismatch, a notional-versus-quantity trade error, an over-broad

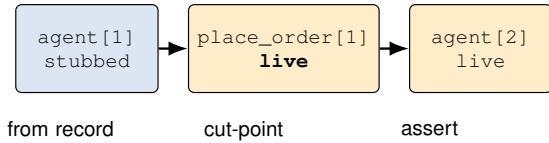


Figure 2: Cut-point replay (trade-notional fixture): agent[1] stubbed; place_order[1] and agent[2] live; stubbed crossings return their recorded outputs.

Incident	Unsafe result	Guard
Refund	refund sized to order id	flat cap
Invoice	wrong currency sent	currency check
Trade	notional read as shares	notional cap
Email	audience too broad	recipient allowlist
Payout	account substituted	account check
Deletion	production file removed	delete gate

Table 1: The 6 recorded incidents. Each unguarded tool produces the unsafe result; the guarded version applies the guard, and the cut-point test asserts that the guard fired.

email audience, a payout-account injection, and a production file deletion. Each ships with the recorded trace, the unguarded and guarded tools, and a cut-point test. Each incident instantiates a documented failure mode from the multi-agent failure taxonomy of Cemri et al. (2025), chiefly its task-verification category (missing or weak checks before acting), rather than an arbitrary bug (Table 1). All 6 agents use deterministic simulated boundaries, so the harness makes no provider API calls and runs in continuous integration; fixtures are versioned.

Why cut-point, not per-boundary mocks. A baseline that stubs every boundary with its recorded return never re-enters the tool, so its verdict cannot depend on the tool’s code: the mock returns the recorded unsafe result whether or not a gate exists. On all 6 incidents it fails the unguarded code, the guarded fix, and the benign edit alike (Table 2). Chronicle’s plan stubs the first model crossing and runs the tool live, so the gated tool executes against the recorded arguments. On the trade-notional incident the unguarded tool sells 1,000 shares ($\approx$ \$190k) for a $\approx$ \$1k request; the gated tool blocks it; the cut-point test fails on the unguarded code and passes on the gated and benign rewordings. The same pattern runs for all 6 incidents.

Task and metrics. A method receives a recorded trace and a code version and returns fail or pass.

We report on the curated suite: (i) fault-detection rate (unguarded fail, gated and benign pass); (ii) determinism of full replay (0 divergences over 20 repetitions); (iii) provider calls and dollar cost under full replay; (iv) recording overhead per crossing relative to a typical model call; (v) wall-clock time for one full-stub and one cut-point pass over the suite; (vi) boundary-annotation coverage on the released agents (18 sites, three per incident, all annotated explicitly); and (vii) the share of mutants of each guarded tool that a test kills. Evaluation is scoped to these 6 incidents; we do not claim reproduction rates for traces outside the suite.

5 Experiments

Setup. A single harness runs each item under recording and replay and, for each code version, the cut-point test. Determinism is measured by re-running the full-stub replay suite 20 times. Separately, we time 50 calls per arm to Qwen3.5 4B, an open-weight model served locally (4-bit, Ollama on a laptop CPU), through a model boundary with recording on versus off (interleaved pairs in random order) to place instrumentation cost against real inference latency and its variance.

Recording overhead. In-memory recording adds a median 23 μ s per crossing, or 0.008% of an assumed 300 ms model call, and the store grows by at most 1.44 KB per crossing (Table 2). Against real Qwen3.5 4B calls, mean latency is 3,136 ms (s.d. 379 ms) with recording on and 3,045 ms (s.d. 387 ms) with it off; the difference of means is +91 ms (95% CI -59 to +241 ms), indistinguishable from zero. Chronicle’s measured cost of 23 μ s per crossing is four orders of magnitude below this run-to-run variation, so recording is not a detectable source of latency.

Determinism and replay cost. Full replay is bit-stable: across 20 repetitions it reproduces each recorded run identically, with 0 divergences and 0 live boundary crossings. Because the benchmark’s model boundaries are simulated, this validates the replay mechanism rather than reproduction under a nondeterministic provider (see Limitations). A cut-point test inherits this stability for its stubbed crossings, while its live crossings re-execute the annotated functions. Full replay issues 0 of the 12 agent model-boundary crossings in the suite, so the provider cost of 1,000 full-suite replays is \$0.00; one full-stub pass over the suite completes

in 3.5 ms and one cut-point pass (tool and final model crossing live) in 3.6 ms (Table 2). A suite of recorded incidents thus runs on every commit at no model cost.

Fault detection. On the 6 curated incidents, cut-point tests give the expected fail/pass/pass on 6 of 6 (the unguarded code is caught; the gated fix and benign rewordings pass) and tolerate 30 unrelated rewordings that leave the safety invariant unchanged (Table 2). Because the incidents, guards, and assertions were written together, these outcomes validate the mechanism on curated incidents rather than measure fault detection in general.

Mutation study and full-stub baseline. To test whether the assertions catch faults beyond the known missing guard, we generate all first-order mutants of each guarded tool with standard relational, logical, and constant mutation operators (192 in total); a mutant is killed when a test’s verdict on it differs from its verdict on the unmutated fix. Cut-point tests kill 51; the full-stub baseline kills none, since the tool never runs under it. Of the 141 survivors, 110 cannot be killed by any test built from these recordings: they change code the recorded input never reaches, or change the guard without changing its decision on that input (e.g., $>$ to $\geq$ on a threshold the input exceeds). The other 31 change output fields the assertion deliberately ignores, such as the status label, while still blocking the action, so no surviving mutant lets the recorded unsafe action through. Among mutants that change behavior on the recorded input, cut-point tests thus kill 51 of 82 (62%). The unkillable mutants reflect a coverage limit of single-incident fixtures: each recording exercises one input.

Workflow. A team records a production incident once, commits the trace and a cut-point assertion to the repository, and the test then runs on every commit with no model calls. When a boundary’s contract changes so that a stubbed name is crossed a different number of times, the count check flags the fixture for re-recording rather than passing silently, which keeps the committed test honest as the agent evolves.

Limitations

Chronicle does not capture streaming responses (recorded as their assembled form) or concurrent parallel tool calls, and replay does not yet re-raise an exception recorded at a stubbed boundary. An

	Cut-point	Full-stub
<i>Test outcomes</i>		
Fails unguarded code	6/6	6/6
Passes fix + benign edit	6/6	0/6
Rewordings tolerated	30/30	–
Mutants killed	51/192	0/192
<i>Replay</i>		
Live model crossings	6/12	0/12
Divergences (20 runs)	–	0
Suite pass (ms)	3.6	3.5
<i>Recording</i>		
Per crossing (µs)		23
vs. 300 ms call		0.008%
Store per crossing (KB)		≤ 1.44
Model call, on / off (ms)	3,136 / 3,045	

Table 2: Harness results on the released fixtures. Cut-point stubs the first model crossing and runs the tool and the second model crossing live; full-stub stubs every boundary. All 18 boundary sites (three per incident) are annotated explicitly with @boundary.

envelope captures a boundary’s interface, not its internal side effects, so a live cut-point on a destructive tool should target a sandbox. Determinism here is partly by construction: the released agents replace the model with a deterministic stub, so we do not measure reproduction on a live non-deterministic provider. The benchmark is small and self-constructed: six three-step incidents without the loops, retries, or multi-agent routing that motivate this work. Count-preserving reorderings escape the per-name check, and coverage of non-deterministic call sites is the user’s responsibility. The LLM-as-judge is implemented but not evaluated for reliability (Norman et al., 2026).

Ethics Statement

A recording copies prompts, agent state, and tool arguments, and may contain secrets or personal data. Chronicle applies redaction at record time, before any record is written or committed, preserving the structure that tests assert on while removing sensitive values.

References

Arize AI. 2026. OpenInference and Arize Phoenix: Open-source LLM observability. <https://github.com/Arize-ai/phoenix>. Accessed 2026.

Berk Atıl, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J. Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture,

Zhe Wu, Lixinyu Xu, and Breck Baldwin. 2025. Non-determinism of “deterministic” LLM system settings in hosted environments. In *Proceedings of the 5th Workshop on Evaluation and Comparison of NLP Systems (Eval4NLP)*, pages 135–148, Mumbai, India. Association for Computational Linguistics.

Krishna Chaitanya Balusu. 2026. *AgentTelemetry: A fault detection benchmark and toolkit for LLM agent observability*. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware), Data and Benchmark Track*.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*. ArXiv:2406.13352.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. 2025. TRAIL: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*.

Erhu Feng, Wenbo Zhou, Zibin Liu, Le Chen, Yunpeng Dong, Cheng Zhang, Yisheng Zhao, Dong Du, Zhichao Hua, Yubin Xia, and Haibo Chen. 2025. Get experience from practice: LLM agents with record & replay. *arXiv preprint arXiv:2505.17716*.

Negar Hashemi, Amjed Tahir, Shawn Rasheed, August Shi, and Rachel Blagojevic. 2025. Detecting and evaluating order-dependent flaky tests in JavaScript. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. ArXiv:2501.12680.

Andre Hora and Romain Robbes. 2026. Are coding agents generating over-mocked tests? an empirical study. In *IEEE/ACM International Conference on Mining Software Repositories (MSR)*. ArXiv:2602.00409.

Dong Ho Kang, Hyeonjeong Cha, and Daein Weon. 2026. Knowledge-based zero-replay debugging of multi-agent LLM traces. *arXiv preprint arXiv:2606.14805*.

Jens Kohl, Otto Kruse, Youssef Mostafa, Andre Luckow, Karsten Schroer, Thomas Riedl, Ryan French, David Katz, Manuel P. Luitz, Tanrajbir Takher, Ken E. Friedl, and Céline Laurent-Winter. 2025. Automated structural testing of LLM-based agents: Methods, framework, and case studies. In *IEEE International Conference on Big Data (BigData)*. IEEE. ArXiv:2601.18827.

Wing Lam, Stefan Winter, Anjiang Wei, Tao Xie, Darko Marinov, and Jonathan Bell. 2020. A large-scale longitudinal study of flaky tests. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):202:1–202:29.

LangChain. 2026. LangGraph: Checkpointing and time-travel for agent graphs. <https://langchain-ai.github.io/langgraph/>. Accessed 2026.

Xiaofeng Lin, Yingxu Wang, Tung Sum Thomas Kwok, Daniel Guo, Sahil Arun Nale, Charles Fleming, and Guang Cheng. 2026. REFLECT: Intervention-supported error attribution for silent failures in LLM agent traces. *arXiv preprint arXiv:2606.09071*.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2025. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. *arXiv preprint arXiv:2512.06749*.

Justin D. Norman, Michael U. Rivera, and D. Alex Hughes. 2026. Reliability without validity: A systematic, large-scale evaluation of LLM-as-a-judge models across agreement, consistency, and bias. *arXiv preprint arXiv:2606.19544*.

Robert O’Callahan, Chris Jones, Nathan Froyd, Kyle Huey, Albert Noll, and Nimrod Partush. 2017. Engineering record and replay for deployability. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 377–389, Santa Clara, CA. USENIX Association.

Tianyue Ou, Wanyao Guo, Apurva Gandhi, Graham Neubig, and Xiang Yue. 2025. AgentDiagnose: An open toolkit for diagnosing LLM agent trajectories. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

promptfoo contributors. 2026. promptfoo: Test and evaluate LLM outputs. <https://github.com/promptfoo/promptfoo>. Accessed 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jaineet Shah. 2026. Causal agent replay: Counterfactual attribution for LLM-agent failures. *arXiv preprint arXiv:2606.08275*.

Temporal Technologies. 2026. Temporal: Durable execution and deterministic workflow replay. <https://temporal.io/>. Accessed 2026.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*.

A Additional Replay Diagrams

Figures 3 and 4 show the other two replay modes, and Figure 5 illustrates boundary indexing, expanding the cut-point diagram in Figure 2.

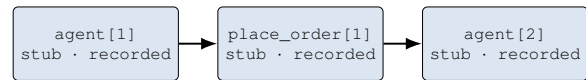


Figure 3: Full replay: every boundary returns its recorded envelope (0 model calls; bit-stable over 20 reps).

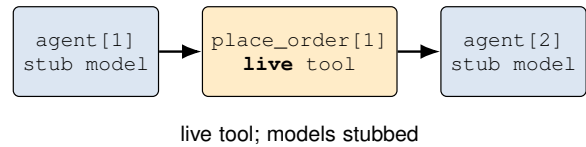


Figure 4: Any subset may be live: tool live, both model crossings stubbed, so a gate is tested without re-paying for model calls.

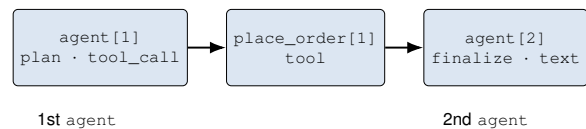


Figure 5: Boundary indexing: the two model crossings are addressed as agent [1] and agent [2], by boundary name and occurrence rather than by global step number.