RT-2026-07 · ACTIVE

Provenance-Preserving Traces: Measuring Lineage Completeness for Post-Training

DRAFT An open research track in progress. It sets out a question we are working on inside a live evidentiary environment, reports what has been measured so far, and names what remains untested. Figures will move as experiments run. We are sharing it to find collaborators, not to present a final conclusion.

Evidence: Implemented Partner: Training compute · exporter needed

Post-training on production agent exhaust risks contamination: unknown model versions, lost prompts, evaluation leakage. We instrument calls on the platform's central inference path as lineage-carrying legs, run a bit-exact input replay harness, and explicitly measure the paths that remain incomplete — because provenance is a profile rather than a property, complete at one altitude and empty at another.

Updated August 12, 2026

> **An open research track in progress.** It sets out a question we are working on inside a live evidentiary environment, reports what has been measured so far, and names what remains untested. Figures will move as experiments run. We are sharing it to find collaborators, not to present a final conclusion.

Training on production agent exhaust is attractive because the exhaust is free and in-domain. It is also, in most deployments, unusable as evidence. The model version behind a given output is frequently unrecoverable; the rendered prompt was never persisted, only the template; sampling parameters were defaults that have changed twice since; and nobody can prove the evaluation set was absent from the training mixture, because nothing existed to keep it out.

The argument

The platform’s central inference path persists each model call it handles as a leg carrying its own lineage — platform, node, model identity, weights identity including checkpoint and quantization, template identity, the rendered prompt with sampling parameters, answer and reasoning with content hashes, and fault

provenance for calls that failed. A failed call used to leave no row. A replay harness re-sends historical prompts bit-exactly and never writes back, so replaying an evaluation cannot contaminate the graph it is scored on. We say **bit-exact input replay** and mean it literally: the stored prompt bytes and sampling parameters are reproduced exactly, and the generated output is not guaranteed to match across model versions, quantizations, kernels, or serving stacks. Reproducing the input is what makes a divergence attributable; it is not the same as reproducing the result.

The first thesis is structural. **Provenance is not a property a system has; it is a profile, and a system scores differently on every dimension of it.** We deliberately avoid calling this a lattice — we have not defined an order, a meet, or a join, and borrowing the algebra without the structure would be the kind of claim this track exists to argue against. What we have is a coverage matrix, and the useful axes are artifact, granularity, producer, weights version, call, dataset, and split.

One codebase can hold three states at once along that matrix. Per-model, per-span, per-version *entity* attribution is complete. Per-*call* attribution has enumerated holes, one of which is our highest-value extractor. *Dataset*-level provenance has no mechanism at all. The framing makes a prediction: teams reporting "full lineage" are reporting their strongest axis. A useful claim names its axis and its weakest link. We name ours below.

The second thesis concerns capture discipline. **Silent-loss-tolerant capture is not provenance.** Two of our own writers persist evidentiary state inside bare exception swallows, one wrapped explicitly so it "never fails the run." A row that may or may not exist, with no error signal when it does not, cannot support a contamination argument: absence is indistinguishable from failure. Not hypothetical — an adjacent scheduled task ran **33 consecutive failed ticks** while its dry-run preview reported healthy, because the preview issues a read and never reaches the write path that was raising — reported in full in Doctrine as Schema.

The third concerns how coverage is stated. "One hundred percent of calls are traced" is falsifiable in our own repository, and we falsified it. The defensible form is enumerative: traced on every path through the dispatcher, with N named exceptions — the list itself a maintained artifact.

What we measured

Disk-level provenance reached completeness across four per-version surfaces: entities with current flag state, relations, integrity findings, and sampler strata baked in at write time. Stated with its denominator, because that is the point of this track: sampler strata are near-universal, the integrity surface is the scarce one, and of the units carrying an integrity surface roughly **96% are complete on all four**. The proof is a consumer rather than an assertion — our benchmark sampler reconstructs training-relevant state from those surfaces with both databases offline.

Two of those surfaces were first-time wins that read as confessions. Relations had existed only in the graph database with no relational mirror, so an offline reader saw nodes and zero relationships. And entity flag *state* had never reached disk, because the mirroring path only appended — an analyst's flag toggle stayed invisible until a resync path was added.

Per-model producer attribution replaced a documented lossy collapse: an ensemble had been recorded as its first producer with a null source span, losing provenance for every entity that path produced. Fixing it required a corpus-wide identity repair, because one extractor had been minting node identifiers in a different case convention, splitting single entities in two.

Trace corpus scale is shared under NDA.

And the weakest link, named: our highest-value extractor — the one mining analyst reasoning out of authored documents — **still bypasses the dispatcher**, calling the inference layer inside a hand-rolled failover loop. For every reasoning trace it produces there is no recoverable tuple of rendered messages, sampling parameters, checkpoint hash, quantization, and serving node. “Full lineage on the reasoning corpus” is not measurable, and we do not assert it.

What this does not show

The title of this track used to say “full-lineage capture.” We changed it, because the strongest thing this work has to offer is the measurement of where lineage is *incomplete*, and a title asserting completeness undercuts that. For the same reason: **this work does not demonstrate contamination-free post-training.** Three distinct mechanisms are in play here, and conflating them would be the most damaging overclaim available, so we name them. The *legacy document-set seal gate* — which refused to seal a benchmark draw while its exclusion list was empty — has been retired in favour of per-pipeline control; it no longer runs. The *per-consumer manifest exclusion* that replaces it is specified and not yet implemented — our track on validation density reports what that gap means in practice. And *trace-level split machinery* is specification-only: no locked test split, no manifest, no writer.

Replay compounds the problem. One comparison mode copies stored legs into a derived child trace, preserving the lineage of a reused prompt while creating near-duplicates. Any split rule must exclude replay children explicitly, and that rule is not written.

Population lineage completeness, measured: 27.8% of compute legs carry null lineage — and the distribution, not the average, is the finding. Split by serving path it is bimodal: **2.9% null on one path, 60.3% on another, and 100% on a handful of evaluation-only producers**. Per producer it runs from 2.6% to **80.8%**.

That shape is why we publish the spread rather than the headline. A single coverage percentage would have read as “mostly fine” and hidden the fact that an entire serving path stamps no checkpoint identity at all — which is precisely the exhaust a post-training pipeline would silently ingest. A per-producer null rate localizes the hole to a specific integration, and localizing it is what makes it fixable. We would encourage anyone reporting trace coverage to report it this way; the aggregate is the least informative form of the number.

A second attribution hole sits alongside it, now partially closing. Human actions were historically recorded without a human — no per-analyst identity on any flag, adjudication or assignment, in either store, so the system knew a decision had been made but not by whom. Doctrine authoring now stamps the analyst,

attribution is being threaded through the remaining mutation paths, and the historical records stay permanently unattributable. Machine lineage and human lineage fail the same way, one altitude apart.

Historical legs carry null lineage permanently; their true lineage is unknowable. Two declared fields for prompt-template identity are never populated. Hardware lineage lives on node telemetry, not on the trace row. The multi-turn layer is a schema seam with nothing in it.

The silent-loss writers remain. Until they become must-succeed with a surfaced error, or record a write-outcome ledger, any claim resting on them is weaker than it looks.

Relationship to prior work

General data and pipeline lineage is a mature area with established standards: OpenLineage already models jobs, runs, datasets, source-code versions and parameters, and we are not proposing a new lineage format. A newer strand examines execution provenance and evidence tracing specifically inside LLM agent systems, which is closer to what we do.

Our contribution is narrower than either. It is the carriage of per-call model lineage — weights identity, quantization, template, rendered prompt, sampling parameters, fault state — all the way through to the point where a record becomes post-training data, combined with an explicit measurement of where that carriage fails. The second half is the part we would defend. Instrumentation papers report what their instrument captures; the interesting number is the one describing what it silently does not, and that number is almost never published. We publish ours above, and it is not flattering.

- OpenLineage — open standard for dataset, job and run lineage (openlineage.io)
- Wang et al., *AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents* — ICSE 2026 (arXiv:2503.18666)
- Mavračić, *Policy Cards: Machine-Readable Runtime Governance for Autonomous AI Agents* — arXiv:2510.24383

Where this is going

Route the reasoning extractor through the dispatcher, so the highest-value trace corpus stops being the least attributable one. Stamp lineage on the serving path that accounts for most of the null rate — a localized fix that the per-producer breakdown makes obvious and the aggregate did not. Turn the coverage-exception enumeration into a maintained artifact rather than a claim. Then build the exporter: traces to training records with provenance intact, deduplication on prompt hash, disjoint splits, and explicit exclusion of replay children.

About RedTorch

RedTorch, Inc.® is an investigative intelligence and applied AI firm, operating continuously since 2016 from Inglewood, California under California Bureau of Security and Investigative Services private investigator

license 189236. Two practices run in parallel — investigative services (due diligence, litigation support, digital forensics, fraud investigation, asset recovery) and AI and strategy consulting — across more than twenty-five service lines. Casework runs on a platform we built, on GPU infrastructure we own. Privileged material never leaves our custody, and provenance, sanitisation and regulatory mapping are built into the platform's data model and operating workflows rather than added as a reporting layer.

RedTorch's platform is a closed-loop environment for evaluating, improving, and governing models on real evidentiary work. Evidence is captured with its lineage intact; machine claims are checked against their sources; analyst acceptances *and* rejections are preserved as supervision; the case graph rather than embedding similarity decides what a model reads; competence is measured from production exhaust; and the autonomy a model is granted is gated on that measurement. Each track in this collection documents one stage of that loop. We are building for systems that reason today and act tomorrow — a harder standard than benchmark performance, and a more useful one.

The corpus is what makes the research possible. It combines real privileged investigative text, an industry taxonomy, model-generated claims at scale, and a growing adjudication layer — analyst acceptances, dismissals, ruled-out hypotheses, doctrine assignments and integrity findings. Coverage and attribution vary by signal, and several of those gaps are themselves subjects of these tracks. What cannot be assembled after the fact is the pairing: a decade of privileged casework alongside the machine output produced against it and the expert judgement passed on that output. Corpus metrics are shared under NDA. This track works on the instrumentation underneath all of it — and on what that instrumentation still fails to capture.

Collaborate

The experiment nobody has run is the filter test: train one model on lineage-complete legs and one on unfiltered exhaust under an identical recipe, then measure how much version-confounded regression the filter removes. A cheaper second: are fault legs usable robustness data? RedTorch, Inc.® supplies the instrumentation, a bit-exact input replay harness, fault provenance, and disk reconstruction with both databases offline. We lack training compute and an exporter to feed it. Researchers and labs interested in this work can get in touch.