Nick Clark · Published April 25, 2026 · PDF

How to Replay and Audit Why an AI Agent Made a Past Decision

When an autonomous agent makes a decision you later need to explain, you usually have logs of what it did but no faithful way to reconstruct the internal state that produced it. This guide describes an architecture for deterministically replaying a past decision and auditing it against the agent's own established pattern of behavior. It is an architecture disclosed in United States Patent Application 19/647,395, not a shipping library, and it centers on the Integrity and Coherence inventive step: an accumulated normative-consistency trajectory plus lineage that lets a decision be both gated in the moment and reconstructed after the fact.

What You Are Building

You are building the ability to answer a hard question after the fact: *why did this agent decide what it decided, three weeks ago, and was that decision consistent with how the agent had behaved up to that point?*

This is the question that arrives with a disputed output, a compliance review, an incident postmortem, or a customer escalation. Most teams reach for their logs and discover they captured the agent's inputs and outputs but not the internal state that connected them. The decision is visible; the reason is gone.

The architecture described here, disclosed in United States Patent Application 19/647,395, does two things a plain log cannot. First, it lets you reconstruct the exact internal state the agent held at a past moment, deterministically, rather than guessing from surrounding context. Second, it lets you check that reconstructed decision against the agent's own accumulated pattern of normative consistency, so you can tell whether the decision fit the established behavior or broke from it. You will implement this yourself; nothing here is a package you install.

Why the Obvious Approaches Fall Short

The usual instinct is to log harder. Structured logs, trace spans, and a vector store of the agent's prior messages all help you see *what happened*, and they are worth having. The structural gap is that they record outputs and events, not the governed state that produced them. If your agent weights a decision using an internal risk posture or a self-assessed alignment score, and you did not serialize that score at that instant, no amount of surrounding log context recovers its exact value. You can narrate the decision; you cannot reproduce it.

A second common approach is to snapshot everything: persist the full internal state at every step. This is faithful but expensive, and it scales poorly for agents that update many internal fields continuously. You end up storing enormous volumes of moment-to-moment state you will almost never read.

A third approach leans on cryptographic provenance and signatures. Signing a lineage record proves it was not tampered with and that state transitions are authentic and continuous. That is genuinely useful and the architecture below assumes it. But a valid signature only tells you the record is intact; it does not tell you whether the decision the record describes was consistent with how the agent had actually been behaving. An agent can produce a perfectly signed, continuous record and still report an internal state that its own behavioral history contradicts. Signature validation and behavioral-consistency validation are different checks.

The Architecture

The disclosed approach rests on three ideas from the filing: a lineage field that records the governed history, deterministic update functions that make that history replayable, and an integrity trajectory that makes the replayed decision auditable against the agent's established pattern.

A lineage field as the single recorded history. In the disclosed schema, lineage is an intrinsic field of the agent object, not an external log. Per the filing, the system records each proposed mutation, each admissibility determination, and each cognitive-domain field update in the lineage field, "such that the complete behavioral trajectory of the semantic agent is deterministically reconstructible from the lineage field alone." The design goal is that lineage is sufficient: you do not consult five subsystems to rebuild the past, you replay one record.

Deterministic updates so state is replayable, not stored. The internal fields the agent uses to weight decisions are defined to update deterministically. The filing gives the affective-state field as a worked example: because each update is a deterministic function and each triggering observation is recorded in lineage, "the agent's affective state at any historical point is reconstructable by replaying the deterministic update function over the sequence of recorded observations from the lineage." The reconstruction "produces the exact affective state vector that existed at the queried timestamp," on demand, "without requiring persistent storage of moment-to-moment affective state values." This is the mechanism that resolves the snapshot-versus-narrate tradeoff: you store the observations and the update-function specification, and you recompute the state when you need it.

An integrity trajectory to audit against. The Integrity and Coherence inventive step adds an integrity field: a deterministic, multi-domain structure encoding the agent's alignment between its declared values and its actual behavioral record as preserved in lineage. It is tracked across three independent domains, described in the filing as personal (consistency with the agent's own declared values), interpersonal

(consistency with relational commitments), and global (consistency with broader systemic norms). Each domain carries its own score and its own trajectory, meaning direction and rate of change over recent evaluation windows.

The filing also specifies what gets written when the agent deviates. When conditions push the agent into what it calls a Deviation-Activated State, every deviation-class mutation is recorded with a marker and a record that includes the values that produced the deviation. In the disclosed deviation function, `D = (N − T) / (E x S)`, those are the need vector `N`, the ethical threshold `T`, the empathy weighting `E`, and the self-esteem score `S`. The filing states this augmented record "ensures that deviation events are fully auditable and that the agent's integrity trajectory can be reconstructed from the lineage with complete fidelity." So the trajectory itself is a replayable artifact, not a summary statistic.

Gating and auditing are the same lineage read from two directions. In the moment, the disclosed system can evaluate a proposed action against the agent's integrity trajectory, described as "the accumulated pattern of normative consistency recorded in its lineage," and selectively permit, gate, or suspend it. After the fact, the filing's integrity-aware trust-slope validation walks the deviation-log entries in lineage to check that the integrity trajectory "follows a plausible path given the agent's operational history." It is defined to flag specific anomaly classes: an agent reporting high integrity while its lineage shows deviations with no corresponding self-esteem impact or restorative effort (a trajectory discontinuity); deviation entries that disappear or get systematically downgraded in severity (trajectory manipulation); and self-esteem that stays flat despite a pattern of deviations (self-esteem decoupling). The audit question "was this decision consistent with the established pattern?" is answered by the same trajectory the gate used to admit it.

How to Approach the Build

The following ordered steps describe how a developer would implement this architecture. The interface sketches are illustrative and faithful to the filing; they are not a library.

1. **Make lineage a first-class field, not a log sink.** Attach a lineage record to the agent object itself and treat it as the authoritative history. Every governed state change writes here. Resist the temptation to let some subsystems keep private state that never reaches lineage; the reconstructibility property depends on lineage being complete.
2. **Define your update functions to be deterministic and record their inputs.** For each internal field you use to weight decisions, write the update as a pure function of prior state plus recorded observations. Then log the observations, not the resulting state. An illustrative shape:

   ```
   # illustrative only, faithful to the disclosed mechanism
   new_state = update_fn(prior_state, observation)   # deterministic
   lineage.append(observation)                        # record the input, no
   ```

   Version the `update_fn` specification and keep old versions, because a faithful replay of an old decision must use the function that was in force then.
3. **Record the admissibility decision alongside the mutation.** For each proposed action, write the proposed mutation and the permit/gate/suspend determination into lineage. This is what lets an auditor later see not just what the agent did but what it considered and how the gate ruled.
4. **Build the replay reader.** Given a target timestamp, replay walks lineage from a known baseline, applies the recorded observations through the versioned update functions in order, and stops at the target. The output is the exact field state that

existed then. Because it is deterministic, two independent replays of the same lineage must agree; use that as a test.

5. **Track the integrity trajectory as you go.** Maintain the three domain scores and their trajectories, and when a deviation occurs, write the full deviation record, including the `N`, `T`, `E`, and `S` values at activation and the restorative response, if any. Do not collapse deviations into a single counter; the anomaly checks need the individual entries.
6. **Implement the trajectory audit as a separate pass.** Reconstruct the integrity trajectory from deviation-log entries and check it for the disclosed anomaly classes: discontinuity, manipulation, and decoupling. This is deliberately distinct from your signature check. Run signature validation first to confirm the record is intact, then run the trajectory audit to confirm the behavior it describes is coherent.
7. **Close the loop with gating.** Feed the same trajectory into your admission gate so that live decisions are evaluated against the accumulated pattern. When you later audit, you are re-reading the very trajectory the gate consulted, which is what makes the audit answer the "was this consistent?" question directly.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and nothing here "just works" out of the box; you implement every piece, including the update functions, the lineage store, the replay reader, and the anomaly checks. The filing describes the mechanisms; it does not ship them.

The approach is disclosed in a patent filing. It has not been presented here as a benchmarked or production-proven system, and this guide states no performance numbers, because the filing establishes the method, not field results.

The replay guarantee is only as good as your discipline. Reconstruction is faithful *if and only if* every triggering observation is recorded and every update function is deterministic and versioned. Introduce nondeterminism (an unrecorded external call, a wall-clock read, an unlogged input) and the exact-reconstruction property breaks for that field. The filing's determinism is explicitly conditioned on the update being deterministic and the observations being recorded.

The integrity audit checks internal consistency, not ground-truth correctness. It can tell you a decision broke from the agent's established pattern, or that the record was manipulated, or that the coherence feedback stopped functioning. It does not tell you the decision was ethically right in some absolute sense; the filing is explicit that the integrity field encodes consistency between declared and enacted behavior, not morality in a philosophical sense. Finally, none of this applies to a stateless agent that keeps no persistent governed state. The architecture presumes an agent with an intrinsic, mutable, lineage-recording internal state; if your agent is a plain stateless inference call, there is no trajectory to reconstruct.

Disclosure Scope

The approach described in this guide is disclosed in United States Patent Application 19/647,395. This guide is educational: it explains an architectural approach so that a skilled developer can understand and build it themselves. It is not a warranty, a specification of a product, or an offer of software, and it does not grant any license. Every description of how the disclosed approach works is drawn from that filing; where the filing does not state a mechanism, parameter, or result, this guide does not claim one.

Track normative consistency. Detect deviation. Self-correct.

Chapter 3 (/patents/19-647395/chapters/integrity)

PRIMARY TECHNICAL DISCLOSURE

- The Coherence Trifecta: Empathy, Integrity, and Self-Esteem as a Unified Control Loop (/articles/the-coherence-trifecta-empathy-self-esteem-and-integrity-as-a-unified-control-loop)

SECONDARY TECHNICAL

- Three-Domain Integrity Model (/articles/integrity-coherence/three-domain-model)
- Deviation Function D=(N-T)/(ExS) (/articles/integrity-coherence/deviation-function)
- Self-Esteem as Internal Validator (/articles/integrity-coherence/self-esteem-validator)
- Deviation as Deterministic Semantic Mutation (/articles/integrity-coherence/deviation-mutation)
- Integrity Structural Placement (/articles/integrity-coherence/structural-placement)
- Empathy as Distributed Moral Load (/articles/integrity-coherence/empathy-mechanism)
- Coherence Trifecta Control Loop (/articles/integrity-coherence/coherence-trifecta)
- Coping Intercept Patterns (/articles/integrity-coherence/coping-intercepts)
- Integrity Deviation Logging (/articles/integrity-coherence/deviation-logging)
- Integrity Collapse Detection (/articles/integrity-coherence/collapse-detection)
- Redemption Engine (/articles/integrity-coherence/redemption-engine)
- Moral Trajectory Forecasting (/articles/integrity-coherence/moral-trajectory)
- Integrity-Aware Trust Slope Validation (/articles/integrity-coherence/trust-slope-integrity)
- Integrity-Confidence Cross-Primitive Coupling (/articles/integrity-coherence/confidence-coupling)
- Integrity-Modulated Discovery Traversal (/articles/integrity-coherence/discovery-integrity)
- Integrity-Aware Multi-Agent Negotiation (/articles/integrity-coherence/multi-agent-negotiation)
- Biological Signal Coupling for Integrity (/articles/integrity-coherence/biological-integrity)
- Policy-Based Integrity Constraints (/articles/integrity-coherence/policy-constraints)
- Integrity Field Portability (/articles/integrity-coherence/field-portability)
- Predictive Deviation Alerting (/articles/integrity-coherence/predictive-alerting)
- Governed Forgetting (/articles/integrity-coherence/governed-forgetting)
- Predictive Social Modeling (/articles/integrity-coherence/predictive-social-modeling)
- Refusal as First-Class Observation (/articles/integrity-coherence/refusal-as-observation)

- Historical Policy-Version Reconstruction (/articles/integrity-coherence/historical-policy-version-reconstruction)

APPLICATIONS · GENERAL

- Autonomous Vehicle Ethical Decision-Making Through Computable Integrity (/articles/integrity-coherence/autonomous-vehicle-ethics)
- Detecting Strategy Drift in Algorithmic Trading Agents With Computable Integrity (/articles/integrity-coherence/trading-normative-consistency)
- How to Keep a Legal AI Agent's Advice Consistent With Precedent and Its Own Prior Positions (/articles/integrity-coherence/legal-advisory-agents)
- Government AI Policy Agents: Consistency, Equity, and Statutory Alignment by Design (/articles/integrity-coherence/government-policy-agents)
- AI Editorial Agents for Newsrooms: Consistent Standards and Bias-Drift Detection by Design (/articles/integrity-coherence/journalism-editorial-agents)
- Integrity and Coherence for AI Environmental Compliance Agents: Consistent Regulatory Interpretation Across Facilities and Jurisdictions (/articles/integrity-coherence/environmental-compliance)
- Integrity and Coherence for Insurance Underwriting Agents (/articles/integrity-coherence/insurance-underwriting)
- Integrity and Coherence for Social Media Moderation Agents (/articles/integrity-coherence/social-media-moderation)
- Legal-Evidence Reconstruction for Autonomous-Incident Litigation: Court-Admissible Policy-Version Lineage (/articles/integrity-coherence/legal-evidence-reconstruction)
- Regulatory Audit Replay With Historical Policy Versions (/articles/integrity-coherence/regulatory-audit-replay)

APPLICATIONS · SPECIFIC

- Waymo vs a Governed Integrity Layer for Autonomous Behavior (/articles/integrity-coherence/waymo)
- Cruise vs Governed Autonomy: Why AV Safety Needs a Computable Integrity Field (/articles/integrity-coherence/cruise)
- JPMorgan Trading Compliance vs Governed Agents: The Integrity Field Gap (/articles/integrity-coherence/jpmorgan)
- Palantir Governance Alternative: Adding a Computable Integrity Field to Government Analytics (/articles/integrity-coherence/palantir)
- Does the Aurora Driver maintain normative memory across decisions? (/articles/integrity-coherence/aurora-innovation)

- Nuro Alternative: Governed Autonomous Delivery Beyond Per-Trip Safety Records (/articles/integrity-coherence/nuro)
- Zoox vs Governed Autonomy: Tracking Normative Drift the Planner Cannot See (/articles/integrity-coherence/zoox)
- Motional Alternative for Governed Normative Trajectory in Autonomous Driving (/articles/integrity-coherence/motional)
- Argo AI Legacy vs Governed Autonomy: The Missing Deviation Function (/articles/integrity-coherence/argo-ai-legacy)
- comma.ai openpilot and the Governed-Behavior Layer: Integrity Coherence for Learning-Based Driving (/articles/integrity-coherence/comma-ai)
- Apache Iceberg Time Travel vs Governed Replay: Data Without Policy (/articles/integrity-coherence/apache-iceberg-time-travel)

Integrity & Coherence overview → (/integrity-coherence)