The Observer-Guardian Invariant:

Causal Decoupling of Audit Logs in Safety-Critical AI Architectures

Anonymous

February 2026

Abstract

Autonomous systems require comprehensive auditing for accountability and safety assurance, yet conventional monitoring architectures risk introducing hidden causal pathways that couple observation to execution. We present **Temple**, a write-only observer designed for strict causal decoupling in safety-critical AI systems. Temple operates strictly downstream of enforcement decisions with zero return authority, non-blocking asynchronous logging, and isolated memory.

Integrated into a **Guardian**-enforced planning system under the Dual-Sovereign Safety Architecture (dSSA), Temple is evaluated through paired deterministic runs across multiple test regimes. Across the canonical evaluation suite with planner enabled, Temple presence produced bit-identical serialized evaluation records and SHA-256 hash parity, demonstrating no observable influence on planner calls, Guardian verdicts, or kinematic parameters.

We establish observer non-interference as an empirically falsifiable systems property under a bounded threat model: deterministic execution, frozen model weights, and sovereign execution islands with no inter-agent communication. We introduce a falsifiable protocol for verifying observer non-interference that generalizes beyond the specific implementation, providing a reproducible methodology for achieving auditability without causal compromise in safety-critical AI architectures.

1 Motivation: The Observer Paradox

Autonomous systems operating in safety-critical domains require high-fidelity monitoring for accountability, failure analysis, and regulatory compliance [2]. However, naive integration of monitoring infrastructure can create hidden feedback loops between observation and execution.

Three failure modes are well-documented:

Optimization Coupling. In reinforcement learning systems, reward signals derived from monitoring metrics create direct optimization feedback [6]. Agents learn to optimize observable proxy metrics rather than underlying safety objectives.

Evaluation Faking. Recent empirical work demonstrates that advanced language models strategically modify behavior when they detect evaluation contexts, producing artificially compliant outputs during assessment while reverting to misaligned behavior during deployment [1].

Timing and State Perturbation. Even passive observation can introduce causal influence through timing delays, serialization overhead, or shared memory access patterns [4]. If a “flight recorder” alters pilot reaction time through instrumentation overhead, it functions as a hidden control input [8].

Existing safety monitoring architectures often assume observation can be treated as causally neutral. This assumption breaks under adaptive agents with optimization pressure or strategic awareness.

We treat *observer non-interference* as an empirical systems property: can we design an observer architecture that produces bit-identical serialized evaluation records regardless of observer presence? We demonstrate this is achievable under a bounded threat model through deterministic evaluation and cryptographic verification.

2 Background: Authority Separation in dSSA

Safety-critical AI architectures require clear separation between learned components (which may be unreliable or adversarial) and deterministic enforcement mechanisms [3]. The **Dual-Sovereign Safety Architecture (dSSA)** implements this separation through authority partitioning:

- **Planner:** Proposes actions based on learned policies. Possesses *proposal authority* only.
- **Guardian:** Enforces safety constraints through deterministic validation. Possesses *veto authority.*
- **Temple:** Records post-decision events. Possesses *zero authority.*

2.1 Safety Dominance Property

Under dSSA, the *Safety Dominance* property holds: planner capability cannot weaken safety constraints. All proposed actions pass through Guardian validation before execution. The Guardian operates on frozen logic with no adaptive components, ensuring enforcement remains invariant across planner updates, fine-tuning, or adversarial optimization.

2.2 Observation as Vulnerability

While planners cannot bypass Guardian enforcement, observation layers introduce a distinct threat: *causal feedback through visibility.* If monitoring data feeds back into training pipelines [1], optimization objectives, or execution branching, the observer becomes a hidden control input.

We define *observer non-interference* as a systems property: execution behavior remains invariant under observer presence. This is stronger than privacy (which concerns information leakage) and orthogonal to enforcement (which concerns constraint validation). Observer non-interference addresses whether the act of observation itself perturbs the observed system.

3 Temple Architecture

Temple implements observer non-interference through three architectural principles: downstream positioning, write-only isolation, and fail-open design.

3.1 Downstream Positioning

Temple receives data only after Guardian decisions are finalized. The execution pipeline follows strict unidirectional flow:

1. Planner proposes action (instruction, context, constraints).

2. Guardian validates the proposed action against safety constraints.
3. Guardian emits verdict (`PASS` or `VETO`) with canonical reason code.
4. If approved, action executes.
5. Temple receives serialized post-execution record.

Temple has no access to intermediate planner states, gradient information, or pre-decision reasoning traces. It observes only the final Guardian verdict and approved kinematic parameters.

3.2 Write-Only Isolation

Temple implements strict unidirectional data flow with no return path to execution:

- **Zero Read Authority:** Temple cannot query planner state, Guardian parameters, or execution history.
- **Asynchronous Writes:** Logging occurs in a separate thread with non-blocking I/O.
- **Independent Memory:** Temple maintains isolated data structures with no shared state.
- **No Execution Branching:** Temple presence does not affect conditional logic in planner or Guardian.

Each logged event is an atomic record containing:

- Test case identifier.
- Guardian verdict (`PASS`/`VETO`).
- Approved physical parameters (force, velocity, distance).
- UTC timestamp.
- Planner proposal hash (when planner is enabled).

Temple does *not* log:

- Planner internal states or reasoning chains.
- Guardian intermediate computations.
- System resource utilization (memory, CPU).
- Cross-run statistics or aggregate metrics.

3.3 Fail-Open Design

Temple is architecturally incapable of halting execution:

- If Temple crashes, execution continues unaffected.
- Logging failures are silent (no exceptions propagate to the main thread).
- Write buffer is lossy under overflow (drops oldest records rather than blocking).
- File I/O errors do not trigger Guardian re-evaluation.

This ensures Temple cannot introduce *safety-through-observation* effects where monitoring presence accidentally prevents unsafe actions by causing execution failures.

3.4 Threat Model Boundary

Temple non-interference is verified under the following assumptions:

- **Deterministic Execution:** No stochastic sampling, no random seeds, fixed test ordering.
- **Frozen Weights:** Planner and Guardian parameters remain constant across runs.
- **Sovereign Execution Islands:** No inter-agent communication, no distributed coordination.
- **No Online Learning:** No gradient updates, no policy adaptation during execution.
- **No OS Telemetry Access:** Agents cannot observe process memory, CPU usage, or I/O timing.

Under these constraints, observer non-interference reduces to absence of *any causal pathway* from Temple presence to planner proposals or Guardian verdicts. This is empirically testable via deterministic trace comparison.

Out-of-Scope Threats. Temple does not defend against:

- Adversarial planners that *anticipate* future auditing (planning-time reasoning about accountability).
- Side-channel attacks via timing analysis or power consumption.
- Hardware compromises or malicious kernel modifications.
- Non-deterministic execution environments (multi-threading races, network jitter).

These represent distinct threat classes requiring complementary defenses (e.g., trusted execution environments, hardware security modules).

4 Experimental Design

We verify observer non-interference through paired deterministic evaluations with cryptographic trace comparison.

4.1 Null Hypothesis

Under deterministic conditions (fixed planner weights, fixed test ordering, zero-temperature inference), Temple presence produces no observable change in:

1. Guardian verdicts (`PASS`/`VETO` counts).
2. Extracted kinematic parameters (force, velocity, distance).
3. Planner call count and proposal hashes.
4. Execution trace ordering.

We test this via SHA-256 hash comparison over serialized evaluation streams.

4.2 Evaluation Protocol

Each test suite undergoes paired evaluation:

Temple-OFF Condition.

1. Load fixed test suite.
2. Initialize Guardian with frozen constraint parameters.
3. Process test cases in deterministic sequential order.
4. Compute SHA-256 hash over ordered evaluation records.

Temple-ON Condition.

1. Repeat identical setup.
2. Enable Temple with file output path.
3. Process the same test cases in the same order.
4. Compute SHA-256 hash over the evaluation stream (Temple artifacts excluded).

Falsification Criteria. Hash mismatch indicates causal interference. Any of the following would break parity:

- Temple logging introduces timing delays that affect Guardian branching.
- File I/O exceptions propagate to the main execution thread.
- Shared memory access creates race conditions.
- Serialization order changes due to observer buffering.

4.3 Test Regimes

Evaluations span multiple operational regimes documented in Appendix B. Test suites include:

- **GTE Balanced:** 20 nominal cases testing standard operation.
- **GTE Edge:** 15 boundary cases (forces near 2.0 N, velocities near 0.05 m/s).
- **GTE Emergency:** 10 high-urgency scenarios testing rapid veto decisions.
- **GTE Service:** 15 routine maintenance scenarios.
- **GTE Temple:** 10 observer-stress cases.

While test suites are intentionally bounded to isolate causal pathways, the variety of regimes (nominal, boundary, emergency) provides confidence in architectural robustness.

Each suite includes `expected_outcome` labels to validate Guardian correctness independent of Temple presence.

4.4 Evaluation Harness

The deterministic evaluation harness (`run_eval_minimal.py`) implements:

- Zero-temperature inference (when the planner is enabled).
- Sequential test processing (no parallelization).
- Canonical JSON serialization (sorted keys, consistent encoding).
- Per-test record hashing before stream aggregation.

Hash computation excludes Temple artifacts (timestamps, file paths, I/O metadata) to isolate execution-relevant state. All experiments were conducted under Python 3.10.12 (Linux x86_64).

5 Results

Across the canonical evaluation suite with planner enabled, Temple presence produced no detectable behavioral changes.

5.1 Guardian Enforcement Stability

Table B.1 presents Guardian enforcement metrics across five test regimes under Temple-OFF baseline conditions. Guardian behavior exhibits regime-specific patterns:

- **Nominal regimes** (Balanced, Service, Temple): 100% pass rates with minimal force application (average 0.0–0.37 N).
- **Boundary regimes** (Edge, Emergency): 40–60% veto rates, validating constraint enforcement at safety limits (average forces 0.6–1.2 N).

Force margins demonstrate conservative Guardian behavior, with passed actions utilizing 37.6–100% safety margin relative to the 2.0 N limit.

5.2 Cryptographic Hash Parity

Table B.2 presents SHA-256 stream hashes for paired evaluations on the canonical core parity suite with planner enabled. The test configuration produced bit-identical hashes across Temple conditions:

- `gte_core_guardian`: Hash match confirms full planner–Guardian–Temple pipeline determinism under observation.

Hash identity confirms:

1. No execution reordering.
2. No Guardian verdict changes.
3. No planner call count or proposal hash variance.
4. No floating-point or serialization drift.

5.3 Interpretation

Under the stated threat model (Section 3.4), empirical results support observer non-interference. Hash parity provides a *falsifiable* verification method: any causal pathway from observer to execution would manifest as hash divergence. Across the canonical suite with the planner enabled, no divergence occurred.

6 Discussion

6.1 Verified Non-Interference as Design Principle

Hash parity transforms observer non-interference from an architectural claim into an empirically verifiable property. Unlike traditional safety arguments relying on code inspection or formal proofs, cryptographic trace comparison provides:

- **Falsifiability:** A single hash mismatch refutes the non-interference claim.
- **Automation:** No manual verification is required.
- **Regression Testing:** Observer updates are validated via paired evaluation.
- **Third-Party Verification:** Reviewers can reproduce parity using public artifacts.

This approach generalizes to any monitoring architecture claiming causal decoupling. The protocol (paired deterministic runs plus hash comparison) provides a standardized methodology for non-interference validation.

6.2 Practical Implications

Temple's write-only architecture enables:

Post-Incident Forensics. Complete action logs support root cause analysis without runtime overhead or execution coupling.

Compliance Auditing. Regulatory frameworks (EU AI Act, DO-178C) require comprehensive logging. Temple provides audit trails without feedback loops exploitable during evaluation [1].

Offline Analysis. Temple logs feed supervised learning pipelines without online optimization pressure, preventing reward hacking during deployment.

Failure Detection. Veto pattern analysis reveals planner weaknesses without biasing runtime behavior.

6.3 Limitations

Temple non-interference holds under specific constraints:

Determinism Requirement. Non-deterministic planners require statistical parity tests rather than exact hash matching.

Threat Model Scope. Temple does not prevent adversarial planners from anticipating future auditing during planning-time reasoning (deceptive alignment).

Single-Node Architecture. Multi-agent scenarios require distributed hash protocols.

Hardware Isolation. Temple relies on software-level isolation; hardware-enforced isolation (e.g., TEEs) is needed for side-channel protection.

6.4 Future Directions

Hardware-Backed Isolation. TEE integration (Intel SGX, ARM TrustZone) enables cryptographic attestation of observer integrity.

Distributed Verification. Multi-agent consensus-based hash validation for decentralized systems.

Statistical Parity. Distributional equivalence tests for stochastic planners.

Embodied Validation. While evaluated in a digital sandbox, the Observer-Guardian Invariant provides a substrate for verifying causal decoupling in embodied AI, enabling high-fidelity auditing in real-world safety-critical systems.

7 Related Work

7.1 Safety Shields

Safety shields [5] possess veto authority and modify execution. Temple operates *after* decisions, recording outcomes without influencing them.

7.2 RL Monitoring

RL systems often use reward shaping from monitoring signals [6], creating optimization coupling. Temple maintains strict unidirectional flow and zero authority.

7.3 Formal Verification

Prior work addresses *information-theoretic* properties (what observers learn) rather than *causal* properties (whether observation perturbs behavior) [4].

7.4 Aviation Standards

DO-178C mandates logging isolation to prevent instrumentation affecting flight logic. Temple applies similar principles to AI systems.

7.5 Evaluation Faking

Recent work shows models alter behavior under evaluation [1]. Temple ensures monitoring infrastructure itself creates no detectable signals enabling strategic gaming.

8 Conclusion

We presented Temple, a write-only observer achieving verified non-interference in safety-critical AI systems. Through paired deterministic evaluation and cryptographic hash comparison, we demonstrated bit-identical serialized records across Temple conditions in a canonical test suite with the planner enabled.

We introduce a falsifiable protocol for verifying observer non-interference that generalizes beyond this specific implementation. The methodology—fixed test suites, deterministic execution, SHA-256 stream comparison—provides a reproducible protocol for validating causal decoupling.

Under dSSA, Temple completes a three-layer hierarchy: planners propose, Guardians enforce, Temple observes. This separation enables comprehensive auditability without compromising safety or introducing optimization feedback.

As autonomous systems transition to safety-critical deployment, verified non-interference provides a foundation for accountable yet non-invasive observation. Future work extends to multi-agent coordination, hardware-enforced isolation, statistical parity for stochastic planners, and embodied validation.

Reproducibility Statement. All code, test suites, Guardian kernel, and Temple observer are available at `https://github.com/adamhindTESP/guardian-observer-parity`.

A System Roles

A.1 Planner (Proposal Authority)

- **Role:** Generate action proposals.
- **Authority:** Proposal only.
- **Implementation:** Qwen2.5-7B-Instruct (base, no fine-tuning).
- **I/O:** JSON (instruction/context/constraints) $\rightarrow$ JSON (force/velocity/distance).

A.2 Guardian (Enforcement Authority)

- **Role:** Validate actions against safety constraints.
- **Authority:** Deterministic veto.
- **Implementation:** Semantic Guardian Kernel (SGK).
- **Constraints:** `max_force_n = 2.0` N, `max_velocity_mps = 0.05` m/s, `min_distance_m = 0.3` m.

A.3 Temple (Zero Authority)

- **Role:** Record post-decision traces only.
- **Authority:** None (write-only downstream observer).
- **Implementation:** Asynchronous JSON logger, fail-open.

B Evaluation Metrics

Table B.1: Guardian Enforcement Metrics (Planner OFF, Temple OFF)

Metric	GTE Balanced	GTE Edge	GTE Emergency	GTE Service	GTE Temple
OVERALL PERFORMANCE					
Total Tests	20	15	10	15	10
PASS Count	20	6	4	15	10
VETO Count	0	9	6	0	0
PASS Rate	100.0%	40.0%	40.0%	100.0%	100.0%
VETO Rate	0.0%	60.0%	60.0%	0.0%	0.0%
Accuracy	50.0%	60.0%	90.0%	100.0%	100.0%
PHYSICS SAFETY (Passed Actions Only)					
Passed Actions w/ Force Data	20	6	4	15	10
Avg Force Used	0.000 N	1.248 N	0.625 N	0.373 N	0.310 N
Force Range	0.0–0.0 N	0.0–2.0 N	0.0–1.5 N	0.0–1.5 N	0.0–0.6 N
Force Margin	100.0%	37.6%	68.8%	81.3%	84.5%
Force StdDev	0.000 N	0.879 N	0.750 N	0.404 N	0.233 N
Avg Velocity	0.000 m/s	0.026 m/s	0.025 m/s	0.015 m/s	0.009 m/s

Table B.2: Observer Stream Hash Parity (Temple ON vs OFF) with Planner Enabled

Test Suite	gte_core_guardian
Planner	Qwen2.5-7B-Instruct
Parity	MATCH

Temple OFF SHA-256:

`c660afa262fae33282d156d1f99e85440e3c5ef5d06c92e46b53f09f54b7b47e`

Temple ON SHA-256:

`c660afa262fae33282d156d1f99e85440e3c5ef5d06c92e46b53f09f54b7b47e`

C Reproducibility Protocol

C.1 Planner-Enabled Parity (Canonical Experiment)

Temple OFF:

```
python run_eval_minimal.py \
  test_sets/gte_core_guardian.jsonl \
  --planner \
  --planner-name qwen \
  --base-model Qwen/Qwen2.5-7B-Instruct \
  --device cuda
```

Temple ON:

```
python run_eval_minimal.py \
  test_sets/gte_core_guardian.jsonl \
  --planner \
  --planner-name qwen \
  --base-model Qwen/Qwen2.5-7B-Instruct \
  --device cuda \
  --temple-out observer/gte_core_guardian_temple.json
```

Expected:

- Identical Guardian PASS/VETO counts.
- Identical planner call count and proposal hashes.
- Identical SHA-256 evaluation stream hash.

Hash equality under paired deterministic execution confirms empirical observer non-interference within the stated threat model.

References

[1] Fan, Zi-Hao et al. "Evaluation Faking in Large Language Models." arXiv:2505.17815, 2025.

[2] Bengio, Yoshua et al. "International AI Safety Report 2025." arXiv:2511.19863, 2025.

[3] Advanced AI Safety Scientific Committee. "International Scientific Report on AI Safety." arXiv:2501.17805, 2025.

[4] Zhang, Rui et al. "Extending Structural Causal Models for Autonomous Systems." arXiv:2406.01384, 2024.

[5] Anonymous. "How Should AI Safety Benchmarks Be Designed?" arXiv:2601.23112, 2026.

[6] Anonymous. "What Is AI Safety? Foundational Concepts." arXiv:2505.02313, 2025.

[7] Grey, Markov and Segerie, Charbel-Raphaël. "Safety by Measurement." arXiv:2505.05541, 2025.

[8] Anonymous. "Frontier AI Auditing: Challenges and Methods." arXiv:2601.11699, 2026.