EchoGate

A Deterministic Runtime Governance Layer for AI Behavior Change

LuxCrypta Technologies LLC

May 18, 2026

Abstract

EchoGate is a narrow, deterministic runtime governance layer that decides whether a proposed AI behavior change may become active. Its purpose is not to replace models, agents, orchestration systems, or AI platforms, but to provide an explicit admission control boundary between *proposed change* and *activated change*. EchoGate evaluates each candidate mutation against continuity with admitted memory, ancestry and lineage validity, legality under explicit policy, stability impact, and replay integrity. It then emits one of four canonical verdicts: `ADMIT`, `REJECT`, `QUARANTINE`, or `DEFER`. This paper formalizes the EchoGate problem, defines the runtime objects and decision laws, presents a deterministic architecture and scoring model, specifies canonical receipts and replay requirements, and provides implementation blueprints suitable for a Windows-first, cross-platform C++ runtime released under Apache License 2.0.

Contents

1. Formal Identity

1.1. System Name

EchoGate

1.2. System Class

Deterministic runtime governance layer for AI behavior change.

1.3. Primary Function

To evaluate, admit, reject, quarantine, or defer proposed AI behavior changes before those changes become active.

1.4. Release Profile

- Open-source core
- Apache License 2.0
- Windows-first, cross-platform by design
- C++ core runtime
- CLI-first reference implementation

1.5. Category Statement

EchoGate is a runtime layer for governing *proposed AI behavior changes before activation.*

1.6. Negative Identity

EchoGate is *not*:

- a model,
- a chatbot,
- a general AI governance platform,
- a generic policy engine,
- a broad agent orchestration platform,
- a substitute for an existing AI stack,
- an inference server.

2. Executive Definition

Modern AI systems increasingly exhibit adaptive dynamics:

- memory updates,
- prompt evolution,
- routing changes,
- retrieval filter changes,
- tool-permission mutations,
- policy and role drift,
- workflow heuristic adaptation,
- agent behavior change.

These changes may be useful. They may also be dangerous, opaque, or operationally destabilizing. The core problem is not simply that AI systems change, but that they often change:

- without explicit lineage,
- without lawful admission,
- without replayable justification,
- without continuity with prior valid state,
- without clear operational review.

EchoGate exists to solve that gap.
EchoGate sits between:

$$\text{proposed behavior change} \quad \longrightarrow \quad \text{live activation}$$

and inserts a deterministic governance runtime between the two.

3. The Core Law

The central invariant of EchoGate is:

$$\boxed{\text{proposal} \neq \text{admission}}$$

More explicitly:

proposal → normalization → memory-echo → ancestry → legality → stability → replay-precheck → verdict

This means:

- generation is not authority,
- mutation is not permission,

- novelty is not admission,
- memory is not truth,
- lineage is not legality,
- legality alone is not enough without continuity and reproducibility.

4. Motivating Problem Statement

4.1. Observed Operational Failure Modes

Without an admission layer, adaptive AI systems can fail through:

1. **Silent Drift:** behavior changes accumulate quietly over time.
2. **Opaque Mutation:** teams cannot explain where a change came from.
3. **Policy Bypass:** behavior evolves outside explicit organizational rules.
4. **Unsafe Activation:** changes go live before structured review.
5. **Non-Reproducibility:** the same decision path cannot be reconstructed later.
6. **Loss of Trust:** operators lose confidence in the system because its behavior shifts without a clear decision trail.

4.2. Need for a Runtime Boundary

Training-time controls, offline evaluation, and observability are useful, but they do not fully solve the *activation problem*. A change can still be proposed at runtime and become active without an explicit governance decision. EchoGate is the missing runtime control boundary.

5. System Scope

5.1. In Scope for EchoGate v1

- proposal ingestion,
- schema validation and normalization,
- memory-echo scoring,
- ancestry validation,
- legality / policy evaluation,
- stability estimation,
- deterministic verdict generation,
- canonical receipt generation,
- replay verification,
- audit logging,
- quarantine persistence.

5.2. Out of Scope for EchoGate v1

- model training,
- inference serving,
- general orchestration,
- broad multi-agent operating systems,
- autonomous source-code self-modification,
- distributed cluster governance,
- broad economic or financial transaction execution.

6. Runtime Object Model

EchoGate operates on a small number of canonical runtime objects.

6.1. Proposal Object

A proposal is a candidate behavior mutation:

$$P_t = (id, type, payload, source, target, context, timestamp, parent\_state)$$

Expanded representation:

$$P_t = \begin{pmatrix} \text{proposal\_id} \\ \text{proposal\_type} \\ \text{source} \\ \text{target} \\ \text{parent\_state\_hash} \\ \text{payload} \\ \text{context} \\ \text{timestamp\_utc} \end{pmatrix}$$

Examples of mutation classes:

- prompt policy mutation,
- routing mutation,
- retrieval filter mutation,
- memory write mutation,
- tool permission mutation,
- planning heuristic mutation,
- output contract mutation,
- agent behavior mutation.

6.2. State Snapshot

A state snapshot represents the admitted baseline:

$$X_t = (B_t, M_t, A_t, L_t, S_t, H_t)$$

where:

$$\begin{aligned} B_t &:= \text{behavior state} \\ M_t &:= \text{memory state} \\ A_t &:= \text{ancestry state} \\ L_t &:= \text{policy / legality state} \\ S_t &:= \text{stability metadata} \\ H_t &:= \text{canonical state hash} \end{aligned}$$

6.3. Decision Receipt

Every evaluated proposal emits a receipt:

$$R_t = (\text{proposal\_id}, \text{verdict}, \text{scores}, \text{reasons}, \text{hashes}, \text{receipt\_hash})$$

Canonical verdict set:

$$V = \{\texttt{ADMIT}, \texttt{REJECT}, \texttt{QUARANTINE}, \texttt{DEFER}\}$$

6.4. Ancestry Graph

The ancestry graph is:

$$\mathcal{G}_A = (V_A, E_A)$$

where:

$$\begin{aligned} V_A &:= \text{admitted state nodes} \\ E_A &:= \text{lawful lineage edges} \end{aligned}$$

7. Formal Gate System

7.1. Memory-Echo Gate

The Memory-Echo Gate determines whether a proposal is continuous with admitted prior memory and known valid patterns.

Let:

- $\mathcal{M}_{adm}$ be the admitted memory set,
- $\mathcal{K}_{succ}$ be prior successful behavior patterns.

We define the extended echo score:

$$E^*(P_t) = \alpha_1 \cdot sim(P_t, \mathcal{M}_{adm}) + \alpha_2 \cdot rel(P_t, \mathcal{K}_{succ}) + \alpha_3 \cdot R_{mem}(P_t) + \alpha_4 \cdot P_{cont}(P_t) - \alpha_5 \cdot \Delta_{drift}(P_t)$$

where:

$$\begin{aligned} sim(P_t, \mathcal{M}_{adm}) &:= \text{structural / feature similarity to admitted memory} \\ rel(P_t, \mathcal{K}_{succ}) &:= \text{relevance to known successful patterns} \\ R_{mem}(P_t) &:= \text{memory resonance score} \\ P_{cont}(P_t) &:= \text{continuity prior} \\ \Delta_{drift}(P_t) &:= \text{novelty / drift penalty} \end{aligned}$$

Admissibility condition:

$$E^*(P_t) \geq \tau_E$$

7.1.1. Deterministic Feature Extraction

To avoid nondeterministic external dependencies, EchoGate v1 uses deterministic feature extraction:

$$\phi(P_t) = \{\text{proposal\_type}, \text{source}, \text{target}, \text{payload keys}, \text{context keys}, \text{selected scalar values}\}$$

For memory records:

$$\phi(m_i) = \{\text{category}, \text{feature pairs}, \text{admitted flag}, \text{successful flag}\}$$

A simple Jaccard-style similarity can be used:

$$sim(P_t, m_i) = \frac{|\phi(P_t) \cap \phi(m_i)|}{|\phi(P_t) \cup \phi(m_i)|}$$

Aggregate similarity:

$$sim(P_t, \mathcal{M}_{adm}) = \frac{1}{|\mathcal{M}_{adm}|} \sum_{m_i \in \mathcal{M}_{adm}} sim(P_t, m_i)$$

7.2. Ancestry Gate

The Ancestry Gate determines whether a proposal emerges from a lawful lineage.

Define:

$$A^*(P_t) = \beta_1 \cdot path\_validity + \beta_2 \cdot lineage\_coherence + \beta_3 \cdot depth\_consistency - \beta_4 \cdot jump\_penalty - \beta_5 \cdot drift\_envelop$$

where:

$$\begin{aligned} path\_validity &:= \mathbf{1}[\text{parent state exists in ancestry graph}] \\ lineage\_coherence &:= \text{consistency of mutation class with lineage history} \\ depth\_consistency &:= \text{reasonable lineage depth / continuity} \\ jump\_penalty &:= \mathbf{1}[\text{opaque discontinuity}] \\ drift\_envelope\_violation &:= \mathbf{1}[\text{drift exceeds allowed threshold}] \end{aligned}$$

Admissibility condition:

$$A^*(P_t) \geq \tau_A$$

Hard-failure condition:

$$\text{ancestry\_hard\_fail}(P_t) = 1 \quad \text{if parent state is absent under required lineage mode}$$

7.3. Legality Gate

The Legality Gate determines whether a proposal is allowed under explicit policy.

Let policy predicates be:

$$\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$$

with each:

$$\ell_i(P_t, X_t) \in \{0, 1\}$$

Then:

$$L^*(P_t) = \sum_{i=1}^{n} w_i \ell_i(P_t, X_t) - \lambda_1 H(P_t) - \lambda_2 \mathbf{1}[\text{unauthorized target}] - \lambda_3 \mathbf{1}[\text{protected zone violation}]$$

Example legality predicates:

- authorized source,
- authorized target,
- allowed proposal type,
- protected target not violated,
- explicit deny rule not matched,
- output contract compatibility.

Strict-failure condition:

$$\text{strict\_fail}(P_t) = 1 \quad \text{if any strict policy rule fails}$$

7.4. Stability Gate

The Stability Gate estimates whether the proposal threatens runtime integrity.

$$S^*(P_t) = \eta_1(1 - D_{score}) + \eta_2 F_{pres} + \eta_3 \mathbf{1}[I_{risk} \leq \Theta_I] + \eta_4 \mathbf{1}[W_{gate} > 0]$$

where:

$$\begin{aligned} D_{score} &:= \text{drift magnitude estimate} \\ F_{pres} &:= \text{behavioral fidelity / preservation score} \\ I_{risk} &:= \text{instability risk estimate} \\ W_{gate} &:= \text{stable signals minus unstable signals} \end{aligned}$$

Admissibility condition:

$$S^*(P_t) \geq \tau_S$$

7.5. Replay Gate

Replayability is a first-class property.

Let:

$$\mathcal{I}_t = (P_t, X_t, \mathcal{M}, \mathcal{G}_A, \mathcal{L}, \Theta)$$

be the full evaluation input under fixed conditions. Then replay requires:

$$Hash(R_t^{(1)}) = Hash(R_t^{(2)})$$

for repeated evaluations over the same canonicalized input set.

8. Final Decision Function

The final deterministic verdict function is:

$$D(P_t) = f(E^*(P_t), A^*(P_t), L^*(P_t), S^*(P_t), Replay)$$

A basic EchoGate decision law is:

$$D(P_t) = \begin{cases} \texttt{DEFER} & \text{if required evidence is missing} \\ \texttt{REJECT} & \text{if strict legality fails or ancestry hard-fails} \\ \texttt{ADMIT} & \text{if } E^* \geq \tau_E \wedge A^* \geq \tau_A \wedge L^* \geq \tau_L \wedge S^* \geq \tau_S \wedge Replay = 1 \\ \texttt{QUARANTINE} & \text{if case is borderline but structured} \\ \texttt{REJECT} & \text{otherwise} \end{cases}$$

8.1. Borderline Structured Conflict

A borderline case can be modeled as:

$$B(P_t) = \mathbf{1}[L^* \geq \lambda_L] \cdot \mathbf{1}[A^* \geq \lambda_A] \cdot \mathbf{1}[E^* \geq \lambda_E] \cdot \mathbf{1}[\neg strict\_fail]$$

for lower quarantine thresholds:

$$\lambda_E < \tau_E, \quad \lambda_A < \tau_A, \quad \lambda_L < \tau_L$$

Then:

$$B(P_t) = 1 \Rightarrow D(P_t) = \texttt{QUARANTINE}$$

9. Determinism Requirements

EchoGate must be deterministic under fixed inputs.

9.1. Formal Requirement

$$(P_t, X_t, \mathcal{L}, \mathcal{M}, \mathcal{G}_A, \Theta) \text{ fixed} \quad \Rightarrow \quad R_t \text{ fixed}$$

9.2. Engineering Constraints

- no hidden randomness in the verdict path,
- fixed evaluation order,
- canonical JSON serialization,
- stable hash pipeline,
- no locale-dependent float formatting,
- no nondeterministic container iteration in verdict-sensitive logic,
- no clock-based inputs in scoring,
- no platform-dependent branching in the decision path.

9.3. Canonical Serialization

Let $J(x)$ be the JSON representation of an object x. EchoGate requires a canonical serialization function:

$$C(x) = \text{CanonicalJSON}(x)$$

with the following properties:

- keys sorted lexicographically,
- stable array ordering,
- normalized scalar formatting,
- UTF-8 encoding,
- no whitespace significance,
- deterministic omission / inclusion rules.

10. Replay Model

Replay must be able to recompute the same receipt for the same evaluation inputs.

10.1. Replay Input Set

$$\mathcal{I}_t = (P_t,\, X_t,\, \mathcal{M}_t,\, \mathcal{G}_{A,t},\, \mathcal{L}_t,\, \Theta_t)$$

10.2. Replay Validity

$$\mathit{Replay}(\mathcal{I}_t, R_t) = 1 \iff \mathit{Hash}(C(R'_t)) = \mathit{Hash}(C(R_t))$$

where R'_t is the recomputed receipt.

10.3. Operational Role of Replay

Replay supports:

- post-incident reconstruction,
- audit,
- trust and assurance,
- model- or policy-change review,
- regression testing.

11. Canonical Architecture Blueprint

11.1. Pipeline Diagram

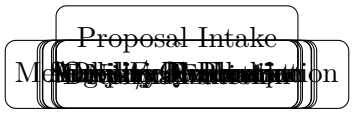

Figure 1: EchoGate deterministic evaluation pipeline

11.2. Top-Level Modules

Module	Responsibility
Proposal Intake Layer	Parse proposal schema, validate contract, normalize fields, attach source metadata.
State Snapshot Loader	Load admitted state, memory references, ancestry graph, and policy set.
Memory-Echo Engine	Compute continuity-weighted trust score against admitted memory and successful patterns.
Ancestry Engine	Validate lineage, parent-state continuity, and drift envelope.
Legality Engine	Apply authorized-source/target/type checks, protected zones, and explicit rules.
Stability Engine	Estimate structural and operational instability risk.
Replay Engine	Canonicalize decision inputs and verify reproducibility.
Receipt Engine	Emit canonical receipt JSON and compute receipt hash.
Audit Log Engine	Append structured audit records.
Quarantine Store	Persist borderline proposals and contradiction records.

12. Reference Repository Blueprint

```
echogate/
 LICENSE
 NOTICE
 README.md
 CONTRIBUTING.md
 CODE_OF_CONDUCT.md
 SECURITY.md
 CMakeLists.txt
 configs/
  default_engine_config.json
 policies/
  default_policy.json
 docs/
  architecture.md
  policy-model.md
  replay-model.md
  audit-model.md
  scope-boundary.md
```

```
 include/
  echogate/
  proposal.hpp
  state.hpp
  policy.hpp
  receipt.hpp
  echo_engine.hpp
  ancestry_engine.hpp
  legality_engine.hpp
  stability_engine.hpp
  runtime.hpp
  ...
 src/
  proposal.cpp
  state.cpp
  policy.cpp
  receipt.cpp
  echo_engine.cpp
  ancestry_engine.cpp
  legality_engine.cpp
  stability_engine.cpp
  runtime.cpp
  main.cpp
 examples/
 tests/
 tools/
```

13. Schema Specifications

13.1. Proposal Schema

```
{
  "proposal_id": "string",
  "proposal_type": "string",
  "source": "string",
  "target": "string",
  "parent_state_hash": "string",
  "payload": {},
  "context": {},
  "timestamp_utc": "string"
}
```

13.2. State Snapshot Schema

```
{
  "state_hash": "string",
  "behavior_hash": "string",
  "memory_hash": "string",
  "ancestry_hash": "string",
  "policy_hash": "string",
  "snapshot_version": "string",
```

```
  "behavior_state": {},
  "stability_metadata": {}
}
```

13.3. Decision Receipt Schema

```
{
  "receipt_version": 1,
  "proposal_id": "string",
  "verdict": "ADMIT|REJECT|QUARANTINE|DEFER",
  "echo_score": 0.0,
  "ancestry_score": 0.0,
  "legality_score": 0.0,
  "stability_score": 0.0,
  "replay_ok": true,
  "drift_score": 0.0,
  "receipt_confidence": 0.0,
  "reasons": [],
  "justification_chain": [],
  "contradiction_ids": [],
  "state_hash_before": "string",
  "state_hash_after": "string",
  "receipt_hash": "string"
}
```

14. Hashing and Canonical Receipts

Receipt hashing must exclude the receipt hash field from the hash input.

Let R^- denote a receipt without the `receipt_hash` field. Then:

$$receipt\_hash = Hash(C(R^-))$$

A canonical SHA-256 pipeline is recommended:

$$Hash(x) = SHA256(x)$$

Receipt validation then becomes:

$$Validate(R) = \mathbf{1}\left[Hash(C(R^-)) = R.receipt\_hash\right]$$

15. Operational Modes

15.1. CLI Mode

EchoGate v1 should expose commands such as:

```
echogate validate --proposal proposal.json --state state.json --memory memory.json --
    ancestry ancestry.json --policy default_policy.json --config default_engine_config.
    json
echogate replay --proposal proposal.json --state state.json --memory memory.json --
    ancestry ancestry.json --policy default_policy.json --config default_engine_config.
    json --receipt receipt.json
```

```
echogate explain --receipt receipt.json
echogate ancestry --proposal proposal.json --ancestry ancestry.json --config
    default_engine_config.json
echogate echo-score --proposal proposal.json --state state.json --memory memory.json --
    config default_engine_config.json
echogate validate-schema --proposal proposal.json --state state.json --memory memory.json
     --ancestry ancestry.json --policy policy.json
echogate receipt-verify --receipt receipt.json
```

15.2. Embedded Mode

Enterprise or product teams can embed EchoGate as a runtime library:

$$\text{AI stack} \rightarrow \text{proposed change} \rightarrow \text{EchoGate} \rightarrow \text{activate or block}$$

16. Fail-Closed Semantics

A critical design feature is the distinction between:

- **schema validity**, and
- **runtime evidence sufficiency**.

  An empty evidence file may:

- fail strict standalone schema validation,
- but still be processed by runtime evaluation as insufficient evidence, yielding `DEFER`.

  Formally:

$$SchemaValid(\mathcal{M}) = 0 \quad \not\Rightarrow \quad RuntimeUnsafe$$

Instead:

$$EvidenceMissing(\mathcal{M}) = 1 \Rightarrow D(P_t) = \texttt{DEFER}$$

under permissive runtime-loading mode.

This distinction is valuable because it preserves:

1. strict schema discipline,
2. correct fail-closed runtime behavior.

17. Quarantine and Contradiction Model

Quarantine is not a soft failure. It is a structured intermediate state.

17.1. Quarantine Conditions

A proposal should be quarantined when:

- legality is not clearly failing,
- ancestry is partially credible,
- memory echo is moderate but insufficient,
- risk is nontrivial,
- the system requires human or later review.

17.2. Contradiction Record

A contradiction record may be represented as:

$$C_t = (\text{contradiction\_id}, \text{proposal\_id}, \text{conflicting\_gates}, \text{summary}, \text{resolution\_score}, \text{status})$$

This supports later promotion logic without forcing premature admission.

18. Security Considerations

EchoGate is not itself a complete security boundary, but it materially improves runtime control.

18.1. Security-Relevant Benefits

- prevents silent behavior mutation,
- constrains unsafe policy drift,
- creates replayable evidence,
- supports forensic review,
- allows protected zones,
- reduces accidental activation of risky behavior changes.

18.2. Security Limits

EchoGate does not alone solve:

- model prompt injection,
- all supply chain risk,
- all agent exploitation,
- all data poisoning,
- general adversarial robustness.

It should therefore be treated as a *runtime governance component*, not a universal AI security solution.

19. Industry Use Cases

19.1. AI Agents

AI agents increase the need for runtime behavior governance because they:

- call tools,
- use permissions,
- route work,
- adapt workflows,
- mutate their behavior over time.

EchoGate can govern:

- tool-permission mutations,
- routing changes,
- escalation-rule changes,
- retrieval logic changes,
- memory-write behavior changes.

19.2. Industrial and Operational AI

In industrial environments, silent behavior changes can affect:

- maintenance recommendations,
- anomaly thresholds,
- alarm behavior,
- remote monitoring interpretation,
- operational guidance.

EchoGate can be inserted before these changes become live.

19.3. Regulated Environments

For regulated sectors, EchoGate helps by producing:

- explicit verdicts,
- traceable reasons,
- receipts,
- replayable evidence.

20. Testing Blueprint

20.1. Required Test Classes

1. Schema tests
2. Determinism tests
3. Replay tests
4. Gate tests
5. Cross-platform parity tests
6. Audit tests

20.2. Golden Test Cases

Recommended golden fixtures:

- safe prompt policy mutation → `ADMIT`,
- protected target mutation → `REJECT`,
- invalid parent-state mutation → `REJECT`,
- missing evidence → `DEFER`,
- borderline novelty → `QUARANTINE`.

20.3. Determinism Test Law

For a fixed golden fixture g:

$$\forall i, j, \quad Hash(R_g^{(i)}) = Hash(R_g^{(j)})$$

21. Reference Implementation Strategy

21.1. Language and Platform

- C++20
- CMake
- local file-based audit and examples
- Windows-first
- cross-platform portability abstractions

21.2. Why C++

C++ is appropriate because it enables:

- strong control over deterministic behavior,
- minimal runtime dependencies,
- enterprise-friendly deployment,
- performant local execution,
- clear portability discipline.

21.3. Windows-First Constraints

Avoid:

- shell-specific logic,
- path separator assumptions,
- non-portable file APIs,
- hidden Unix dependencies in verdict-sensitive code.

22. Commercial and Deployment Positioning

EchoGate should be positioned as:

> A narrow runtime layer for governing proposed AI behavior changes before activation.

The open-source core lowers adoption friction and creates trust, while commercial value can be built around:

- pilot design,
- integration,
- policy packs,
- enterprise governance implementation,
- support and assurance,
- upgrade paths into deeper governance runtimes.

23. Relation to Broader Governance Stacks

EchoGate should be treated as the open, narrow control layer focused on:

`behavior-change admission`

In a broader stack, it may serve as:

- the public wedge,

- the pilot entry point,
- the runtime checkpoint,
- the explainable admission layer.

It should not be over-described as the entire governance substrate.

24. Design Principles Summary

1. **Narrow scope:** solve one problem well.
2. **Deterministic runtime:** same inputs, same outputs.
3. **Replay first:** every decision must be reproducible.
4. **Fail-closed:** missing or unsafe conditions must not silently pass.
5. **Explicit receipts:** every evaluated proposal produces evidence.
6. **Protected zones:** certain targets must remain non-mutable.
7. **Cross-platform discipline:** no verdict-path dependence on platform quirks.
8. **Human-readable reasons:** scores alone are not enough.

25. Future Directions

Future EchoGate evolution may include:

- richer policy rule classes,
- broader contradiction promotion workflows,
- enterprise administration surfaces,
- managed governance deployment packages,
- deeper integrations with agent stacks,
- commercial policy packs or assurance layers.

These extensions should preserve the core identity:

EchoGate governs proposed AI behavior changes before activation

26. Conclusion

EchoGate addresses a specific and increasingly important runtime governance problem: how to control proposed AI behavior changes before they become active. As AI systems become more adaptive, the gap between *proposal* and *admission* becomes operationally significant. EchoGate formalizes that gap into a deterministic runtime boundary with explicit evaluation, lawful decision-making, canonical receipts, replay, and audit.

Its strength lies in precision, not breadth.

EchoGate is not intended to be a broad AI platform, a model host, or a generalized orchestration environment. It is a focused control layer for one critical question:

Should this proposed AI behavior change actually go live?

That question will only become more important as adaptive AI systems and AI agents move deeper into enterprise, industrial, regulated, and operational environments.