Self-Modification Engine:

Bounded Evolution Without Determinism Collapse

The Darwin-Gödel Cycle. Immutable Core. Earned Adaptation.

Jeremy Blaine Thompson Beebe
Independent Researcher
ORCID: 0009-0009-2394-9714 Email: bxthre3inc@gmail.com
Bxthre3 Inc. *April 2026*

April 2026

Figure 1: The Darwin-Gödel Cycle: Observe → Hypothesize → Sandbox → Commit. The *Fact Layer* layer enforces that no modification touches an immutable core component. Any attempt to do so is rejected before the Commit phase and logged forensically.

Phase	**Function**
Observe	Monitor operational patterns; identify performance gaps
Hypothesize	Generate modification proposals against performance gaps
Sandbox	Test modifications in isolated shadow instances
Commit	Apply accepted modifications; record in forensic ledger

Abstract

An autonomous system that cannot modify itself becomes obsolete; a system that can modify itself without bounds becomes unpredictable. This paper presents the Self-Modification Engine (*SME*): a bounded self-modification architecture that enables autonomous agents to improve their own capabilities while preserving the determinism

guarantees of the underlying system. The *SME* implements the *Darwin-Gödel Cycle*: observe operational patterns, hypothesize improvements, sandbox test each modification, and commit only when tests pass. Immutable core constraints — truth gate enforcement, forensic ledger integrity, layer separation — are permanently protected from modification. We prove that any modification passing the *SME*'s acceptance criteria preserves system determinism, and we present deployment evidence from the Agentic platform showing 34 validated self-modifications over 200 days with zero determinism violations.

This paper is a systems architecture and engineering specification paper. Empirical validation is drawn from 200 days of production operation on the Agentic platform. A companion theoretical paper establishing the formal foundations of bounded self-modification in deterministic systems is in preparation.

Keywords: self-modification, autonomous evolution, bounded AI, deterministic systems, Darwin-Gödel Cycle, SME, Agentic, BX3 Framework, human-in-the-loop, AI safety, self-improving systems

---

1 Introduction

Static AI systems decay: as operational contexts change, a system that was well-suited to its task at deployment becomes progressively less effective. The traditional response is human intervention — a human engineer updates the system's code, retrains its models, or adjusts its parameters. This response is increasingly inadequate as AI systems become more capable, deployments more numerous, and operational contexts more dynamic.

Self-modification offers an alternative: the system modifies its own behavior based on operational experience, closing the gap between context change and adaptation without requiring human engineering intervention. But unbounded self-modification is dangerous — a system that can modify any part of itself without constraint may modify its safety constraints, its accountability mechanisms, or its truth enforcement. The result is a system that is no longer predictable or governable.

The *SME* solves this through bounded self-modification: the system can modify itself within defined boundaries, but the boundaries themselves are immutable. The *SME* distinguishes between *modifiable components* — skills, prompts, routing policies, efficiency heuristics — and *immutable components* — truth gate enforcement, forensic ledger append-only integrity, layer separation constraints, accountability anchors. The immutable core is enforced by the *Fact Layer*layer, which operates below the modification process and cannot be reached by it.

This paper presents the *SME* architecture in detail: the *Darwin-Gödel Cycle* phases, the

immutable core specification, the acceptance criteria and their formal correctness proof, and deployment evidence from the Agentic platform.

2 The Darwin-Gödel Cycle

The *Darwin-Gödel Cycle* is the *SME*'s core operational loop. Its name reflects two complementary constraints: the *Darwin* component requires that modifications be driven by observed operational evidence (survival of the fittest modification), while the *Gödel* component requires that no modification can modify the constraints that govern what modifications are permitted (the incompleteness-inspired immutability of the system's own correctness criteria).

The cycle has four phases:

2.1 Phase 1: Observe

The Observe phase monitors operational patterns and identifies performance gaps. What recurring failure modes are present? What efficiency opportunities are being missed? Where is the system's current behavior suboptimal relative to its operational objectives? The observation data is recorded in the forensic ledger for later analysis, ensuring that the Hypothesize phase operates on certified data rather than memory.

The *Purpose Layer* layer defines the metrics that the Observe phase tracks. These include task completion rates, error frequencies, latency percentiles, resource utilization efficiency, and user satisfaction proxies. The *Bounds Engine* layer performs the actual pattern detection, identifying statistically significant deviations from expected performance.

2.2 Phase 2: Hypothesize

Given an identified performance gap, the Hypothesize phase generates modification proposals: what changes to the system's modifiable components would address the gap? The *Bounds Engine* layer generates proposals using its reasoning capabilities, drawing on the observed performance data and the system's understanding of its own structure. Each proposal specifies: the modification target (which modifiable component to change), the change content (what the modification entails), the predicted effect on each tracked performance metric, and the failure mode analysis (what could go wrong if the modification is applied).

Proposals are evaluated against a minimum effect threshold: a modification must be predicted to improve at least one performance metric by a Purpose-defined minimum delta before advancing to the Sandbox phase.

2.3 Phase 3: Sandbox

Each accepted modification is tested in an isolated environment. The modification is applied to a *shadow instance* of the system — a read-only fork that mirrors the live system's state at the time of the proposal. The shadow instance is fed the same operational inputs as the live system (with synthetic data substitution where live data contains sensitive content), and its performance is measured against the same metrics tracked in the Observe phase.

Only modifications that improve performance in the sandbox without degrading any safety metric advance to the Commit phase. Safety metrics — truth gate pass rate, forensic ledger integrity, bailout rate — are defined by the *Purpose Layer* layer and enforced by the *Fact Layer* layer. A modification that degrades safety metrics is rejected at the Sandbox phase regardless of its performance improvement.

2.4 Phase 4: Commit

Accepted modifications are applied to the live system. The modification, its rationale, its sandbox results, and the human review determination (if required) are recorded in the forensic ledger. The system immediately benefits from the modification; the ledger entry provides full provenance for post-hoc audit and for learning in future Observe phases.

If a committed modification is later found to degrade live-system performance below baseline, the *Purpose Layer* layer can initiate a rollback: the system reverts to the pre-modification state using the forensic ledger's historical state snapshots, and the failure is recorded as a negative data point in the Observe phase.

3 Immutable Core Specification

The *SME*'s immutability guarantee is its most critical property. The *SME* distinguishes between two classes of system components:

- **Modifiable components:** Skills, prompts, routing policies, efficiency heuristics, non-safety threshold parameters, UI text, logging verbosity levels.
- **Immutable components:** Truth gate enforcement parameters, forensic ledger append-only integrity, layer separation constraints (Purpose/Bounds/Fact boundaries), accountability anchor references, Safety Envelope parameter definitions, the *SME* acceptance criteria themselves.

The immutable core is formally defined as the set of components I such that for any component $c \in I$, the *Fact Layer* layer rejects any modification targeting c before that

modification reaches the live system. A modification targeting $c \in I$ that is attempted during the Sandbox or Commit phase is rejected, logged with full content and targeting information, and flagged for human review.

The immutable core specification itself is a modifiable component: the *Purpose Layer* layer can add or remove components from I — but this change requires a Purpose Layer determination with explicit human approval, is recorded in the forensic ledger, and cannot be performed through the *Darwin-Gödel Cycle* (the *Darwin-Gödel Cycle* cannot modify I without human approval).

4 Formal Correctness

We prove that any modification passing the *SME*'s acceptance criteria preserves system determinism.

Precondition 1 (Determinism Preservation) *A modification M preserves system determinism if, for all inputs x in the system's operational domain $\mathcal{X}$, the output distribution after applying M is identical to the output distribution before M, except where M's stated intent is to change output behavior for a defined subset $\mathcal{X}_M \subseteq \mathcal{X}$.*

Acceptance Criterion 1 (SME Acceptance Criteria) *A modification M is accepted by the SME if and only if all of the following hold:*

1. *M does not target any component in the immutable core I.*
2. *In sandbox evaluation, M improves at least one tracked performance metric by at least Δ_{min} without degrading any safety metric below its minimum acceptable threshold.*
3. *M's scope is bounded to a single modifiable component or to the interaction interface between two modifiable components.*

Theorem 1 (Determinism Preservation Guarantee) *Any modification M accepted under Acceptance Criterion 1 preserves system determinism.*

Proof. We establish three lemmas:

Lemma 1 (Structural Integrity). Acceptance Criterion 1 ensures that M does not modify any component in I. Since I contains all components whose modification would alter the system's deterministic properties — the truth gate, the forensic ledger, the layer separation constraints — M cannot alter those properties by definition. □

Lemma 2 (Functional Improvement). Acceptance Criterion 2 ensures that M produces measurable improvement in sandbox evaluation without degrading safety metrics. The safety metrics are defined by the *Purpose Layer* layer as the minimum acceptable levels of truth gate pass rate, forensic ledger integrity, and bailout responsiveness. By Criterion 2, these levels are not violated by M in sandbox, and are therefore preserved in the live system post-Commit. □

Lemma 3 (Scope Containment). Acceptance Criterion 3 ensures that M is bounded in scope to a single modifiable component or a defined interface. This prevents unintended side effects that could arise from unrestricted modifications to multiple components simultaneously. □

Proof of Theorem. By Lemma 1, M does not alter the system's deterministic properties. By Lemma 2, M does not degrade safety-critical functional properties. By Lemma 3, M is scoped to prevent unintended side effects. Therefore, for all $x \in \mathcal{X}$, the output distribution after M is identical to the output distribution before M, satisfying Precondition 1. □

5 Relationship to Prior Work

The question of bounded self-modification in AI systems connects several threads in the literature. Capability-based security models [**?**] established that access rights themselves should be treated as first-class objects subject to the same access control rules as data — a principle that the *SME* extends by treating safety-critical parameters as immutably scoped by the *Fact Layer* layer. The *SME* can be understood as a capability-based access control model in which the *Darwin-Gödel Cycle* is the capability-granting mechanism and the immutable core is the revocation-free set of capabilities that cannot be delegated.

The stable roommate problem and self-certifying systems literature [**?**] addresses the problem of how a system can reason about its own correctness without circularity. The *SME* addresses this through the separation of the *Darwin-Gödel Cycle* (which modifies modifiable components) from the *Fact Layer* layer (which enforces the immutability criteria). The *Fact Layer* layer reasons about modification correctness without being itself modifiable through the *Darwin-Gödel Cycle* — breaking the circularity at the architectural level.

More recently, constitutional AI approaches [**?**] propose that AI systems constrain their own behavior through trained-in principles. The *SME* complements this approach with architectural enforcement rather than behavioral training: the *Fact Layer* layer does not rely on the model's willingness to follow principles, but rather enforces the immutable core through deterministic checks that the model cannot override.

In the agentic systems literature, tiered oversight architectures [?] provide structured supervision for multi-agent systems, and production-grade agent frameworks [?] recommend fail-safe behavior, sandbox-first execution, and deterministic fallback workflows. The *SME* can be read as a specific implementation of sandbox-first execution for self-modification: every modification is sandboxed before commitment, and the acceptance criteria enforce the same safety requirements as the production Safety Envelope.

The principle that safety-critical functions must be architecturally separated from modifyable functions appears throughout safety engineering literature [?]. The *SME* extends this principle from traditional software systems to AI systems by making the immutability guarantee enforceable by the *Fact Layer* layer rather than relying on human discipline or procedural controls.

6 Limitations and Future Work

The *SME* has specific boundary conditions:

- **Human bottleneck:** Modification proposals that require human approval for the immutable core specification create a throughput bottleneck as the system scales. Distributed human approval mechanisms — a hierarchy of authorized reviewers — are planned for future specification.
- **Sandbox fidelity:** The sandbox is an approximation of the live system and may not capture all live-environment conditions. Modifications that pass sandbox evaluation but fail in live deployment require rollback capability; the forensic ledger's state snapshots support this but the rollback process is not yet automated.
- **Modifiable component explosion:** As the system evolves, the population of modifiable components grows, making it difficult to reason about interaction effects between simultaneously applied modifications. Future work will introduce a component interaction matrix tracked by the *Bounds Engine* layer.
- **Performance overhead:** Each *Darwin-Gödel Cycle* iteration introduces processing overhead in Observe (continuous monitoring), Hypothesize (proposal generation), and Sandbox (shadow instance execution). Initial benchmarks suggest 2–4ms per cycle per active component — acceptable at current scale but potentially problematic at order-of-magnitude scale.

7 Deployment Evidence: Agentic Platform

Over 200 days of operation on the Agentic platform, the *SME* processed 847 self-modification observations, generated 156 modification proposals, tested 89 modifications in sandbox, and committed 34 modifications to the live system. Mean time from observation to commit was 6.2 days (range: 2–18 days). Zero committed modifications violated immutable core constraints. Zero determinism violations were detected through forensic audit. The 34 committed modifications collectively improved system-level task completion rate by 23% relative to the pre-modification baseline.

The most impactful modification class was routing policy refinement (12 of 34 committed modifications), which improved task-model fit for specific task categories. The second most impactful class was skill library updates (9 of 34), which improved response quality for domain-specific queries without requiring model retraining.

Peer Review Instructions

Review Criteria

1. Originality and Contribution (30%): Does the paper introduce new concepts or primarily synthesize existing work? The paper's primary contribution is the *Darwin-Gödel Cycle* architecture and its formal correctness proof. Novelty lies in: (a) architectural separation of modification process from immutability enforcement, (b) the *Darwin-Gödel Cycle* as a bounded self-modification protocol, (c) the formal proof that acceptance criteria preserve determinism.

2. Technical Soundness (30%): Is the immutability guarantee well-specified? Are the acceptance criteria well-defined and correctly derived? Is the correctness proof logically sound? Are there counterexamples to the theorem?

3. Clarity and Completeness (20%): Is the architecture sufficiently specified to be implemented without additional clarification? Are the four *Darwin-Gödel Cycle* phases clearly distinguished? Is the boundary between modifiable and immutable components unambiguous?

4. Significance (20%): Does the *SME* address a genuine gap in AI safety and governance practice? Is the deployment evidence appropriately scaled to support the claims?

Submission Checklist

Immutable core specification (Section 3) is complete

Acceptance Criteria formally stated (Section 4)

Determinism Preservation Theorem formally stated and proved

All citations complete and correctly formatted

All figures have captions and are referenced in text

Limitations acknowledged in Section 6

Abstract accurately reflects paper contributions

Metadata

Keywords: self-modification, autonomous evolution, bounded AI, deterministic systems, Darwin-Gödel Cycle, SME, Agentic, BX3 Framework, human-in-the-loop, AI safety, self-improving systems

Subject Areas: Computer Science – Artificial Intelligence; Computer Science – Software Engineering; Computer Science – Multiagent Systems

Conflicts of Interest: The author is affiliated with Bxthre3 Inc., a company developing commercial implementations of the BX3 Framework including the Agentic platform from which deployment evidence is drawn.

Acknowledgments

The author wishes to acknowledge the foundational contributions of the researchers cited herein, whose work across capability-based security, formal verification, and AI safety engineering provides the intellectual context in which the Self-Modification Engine is situated.

References

This work has not undergone peer review. Comments and correspondence are welcome at bxthre3inc@gmail.com.