# Reflective Self-Amendment

## When the Tool Modifies the Toolmaker — Fixed Points of the AI Capability Cascade

*LLM agents whose behavioral specifications are written in natural language can read, critique, and propose amendments to their own policies. This creates a self-improvement loop structurally impossible in prior AI paradigms. We characterize this as a fixed point of the tool-building cascade and derive conditions under which it converges.*

Tamás Nagy, Ph.D.

tnagyphd@gmail.com

Draft

## Abstract

We identify a structural property unique to large language model (LLM) agents: when an agent's behavioral policy is specified in natural language (a "skill file"), and its reasoning engine operates in the same modality (natural language), the agent can read, evaluate, and propose modifications to its own behavioral specification. We call this *reflective self-amendment* (RSA). Unlike reinforcement learning (opaque weights), constitutional AI (fixed constitution), or symbolic self-modification (different formalism for rules and reasoning), RSA exploits the *same-modality property*: policy and reasoning share a representational medium. We formalize RSA as a fixed point of the tool-building cascade: the point where the tool that improves tools is directed at itself. We give conditions under which the RSA loop converges (bounded improvement per cycle) versus diverges (unbounded self-modification), connecting to known results on AI self-improvement bounds.

## 1. Introduction

### 1.1 The Problem

AI agents are increasingly deployed as autonomous systems that perform complex tasks. Their behavior is determined by some combination of training (weights), prompting (system instructions), and tooling (available APIs). A natural question arises: can the agent improve its own behavioral specification?

This question has a long history under various names — self-modifying code (von Neumann, 1966), reflective architectures (Smith, 1984), meta-learning (Schmidhuber, 1987), and recursive self-improvement (Good, 1965; Bostrom, 2014). But a new structural possibility emerged with LLM-based agents: the behavioral specification is a *text document*, and the reasoning engine is a *text processor*. For the first time, the policy and the reasoning operate in the same modality.

### 1.2 The Same-Modality Property

Consider three paradigms:

| Paradigm | Policy representation | Reasoning engine | Same modality? |
|---|---|---|---|
| Reinforcement learning | Float vectors (weights) | Gradient computation | No |
| Symbolic AI | Formal rules (Prolog, production rules) | Inference engine (unification, chaining) | Partially |
| **LLM + text specification** | **Natural language (skill file)** | **Natural language (LLM)** | **Yes** |

In RL, the agent cannot "read" its own weights in any meaningful sense — the policy is opaque to the reasoning process. In symbolic AI, the rules and the reasoning engine share a formal structure, but the agent's reasoning about its own rules requires a meta-level interpreter that is typically harder to build than the object-level system.

In the LLM + text-specification paradigm, the agent's behavioral policy (e.g., a SKILL.md file that defines its goals, constraints, procedures, and style) is written in natural language. The agent's reasoning engine (the LLM) is natively designed to process, analyze, and generate natural language. The agent can therefore:

1. **Read** its own behavioral specification
2. **Evaluate** whether its behavior matches the specification
3. **Identify gaps** between intended and actual behavior
4. **Propose amendments** to the specification
5. **Explain** why the amendment would improve behavior

This is reflective self-amendment.

### 1.3 Contributions

1. We formalize the RSA loop and characterize it as a fixed point of the tool-building cascade (§2)
2. We distinguish RSA from prior self-improvement paradigms and identify the same-modality property as the structural enabler (§3)
3. We give convergence conditions for the RSA loop, connecting to known self-improvement bounds (§4)
4. We describe a concrete implementation and empirical observations from a deployed RSA system (§5)
5. We discuss implications for AI governance: the human's role shifts from operator to legislature (§6)

## 2. The Cascade Fixed Point

### 2.1 The Tool-Building Cascade (Review)

Following Nagy (2026), define a *tool-building cascade* as a sequence of investments:

- **Level 0**: Solve problem instances directly. Output per unit time: $O_0$.
- **Level 1**: Build a tool that solves a class of problems. Multiplier: $m_1$. Effective output: $m_1 \cdot O_0$.
- **Level $k$**: Build a tool that builds Level $k-1$ tools. Multiplier: $m_k$. Total output: $\prod_{i=1}^{k} m_i \cdot O_0$.

The cascade works via lateral domain shift (Nagy, 2026): each level operates in a more structured domain than the previous one. Level 0 may require mathematical insight; Level 1 requires software engineering; Level 2 requires architecture.

### 2.2 The Self-Referential Level

At each level $k$, the tool $T_k$ acts on the tools at level $k-1$:

$$T_k : \mathcal{T}_{k-1} \to \mathcal{T}_{k-1}$$

where $\mathcal{T}_{k-1}$ is the space of Level $k-1$ tools. $T_k$ takes a tool and produces an improved tool.

Now consider what happens when the tool at level $k$ is *itself* a member of $\mathcal{T}_{k-1}$. That is, when the tool-improver is in the same class as the tools it improves. Then:

$$T_k(T_k) = T_k'$$

The tool improves itself. A *fixed point* of the cascade is a tool $T^*$ such that:

$$T^*(T^*) = T^*$$

That is, the tool that results from self-improvement is the same as the tool that performed the improvement. At a fixed point, the system has reached a stable behavioral specification — further self-amendment produces no change.

### 2.3 RSA as a Concrete Fixed-Point Operator

In the LLM agent setting, let $S \in \mathcal{S}$ be a skill specification (a natural language document) and let $\text{LLM}_S$ denote the agent operating under specification $S$. The RSA operator is:

$$\text{RSA}(S) = S + \Delta S$$

where $\Delta S$ is the amendment proposed by $\text{LLM}_S$ after observing its own behavior under $S$.

The RSA loop is the iteration:

$$S_{n+1} = \text{RSA}(S_n) = S_n + \Delta S_n$$

A fixed point satisfies $\Delta S^* = 0$: the agent operating under $S^*$ proposes no further amendments.

### 2.4 Why This Is Not Trivially Achievable

The RSA operator requires several non-trivial capabilities:

1. **Self-reading**: the agent must be able to access and parse its own specification. (Enabled by the same-modality property.)
2. **Behavioral observation**: the agent must notice discrepancies between its specification and its actual behavior. (Enabled by in-context reflection.)

3. **Causal reasoning**: the agent must correctly attribute behavioral gaps to specification gaps (not to execution failures, context limitations, or external factors).
4. **Constructive amendment**: the agent must propose a *specific textual change* to the specification, not just identify a problem.
5. **Non-degeneracy**: the amendment must actually improve behavior, not just change the specification.

Condition 5 is the hardest. Without it, the RSA loop can diverge (oscillating specifications), drift (specifications that change without improving), or collapse (specifications that converge to a trivial fixed point like "do nothing").

# 3. RSA vs. Learning vs. Kaizen

## 3.1 The Three Kinds of Improvement

A critical distinction: not all agent improvement is self-amendment. There are three structurally different kinds:

| Kind | What changes | Analogy | Accumulates? |
|---|---|---|---|
| **Learning** | Memory (what you know) | Studying for an exam | Yes, within memory capacity |
| **Reinforcement** | Weights (how you react) | Muscle memory from practice | Yes, but opaque |
| **Self-amendment** | Rules (how you behave) | Rewriting your own job description | Yes, and readable |

**Learning** stores new facts: "this tactic failed on this goal type." The agent makes better decisions but under the *same rules*. In our system, ProofMemory is the learning layer — it records what worked and what failed.

**Reinforcement** adjusts parameters via reward signals. The agent's behavior changes but neither the agent nor anyone else can read *why*. The change is in the weights, not in a readable specification.

**Self-amendment** changes the rules themselves. The agent operating under specification $S$ becomes an agent operating under specification $S'$. This is structurally different from learning: a learned fact ("tactic X fails on goal Y") does not change the agent's *procedures*. An amendment ("always consult the Oracle before attempting a proof") changes the agent's *workflow*.

The Japanese manufacturing concept of *kaizen* ( , continuous improvement) captures this distinction precisely. Kaizen is not about workers learning more facts or practicing more — it is about **improving the process itself**. A kaizen cycle asks: "how should the procedure change so that this class of problems does not recur?" RSA is kaizen applied to the agent's behavioral specification. The agent asks: "how should my SKILL.md change so that I handle this class of situations better?"

The power of RSA over learning: learning is bounded by the agent's memory capacity and retrieval quality. Self-amendment changes the *structure* of behavior, not just the *data* available to it. A single well-chosen amendment (e.g., "persist intermediate conjectures before session close") can prevent an entire class of failures — something no amount of stored facts can achieve.

### 3.2 RL Policy Gradient

In RL, the "self-improvement" loop is:

$$\theta_{n+1} = \theta_n + \alpha \nabla_\theta J(\theta_n)$$

where $\theta$ is the policy parameter vector and $J$ is the expected reward. The agent follows a gradient in parameter space. But $\theta$ is not readable: the agent cannot inspect $\theta_n$ and reason about what it means. The improvement is blind — it follows the gradient without understanding the policy.

RSA is fundamentally different: the agent *understands* its own specification (because it is natural language) and can reason about *why* a change would help.

### 3.3 Constitutional AI

In constitutional AI (Bai et al., 2022), the agent is given a constitution $C$ and trained to follow it via self-critique:

$$\text{output} = \text{LLM}(\text{input}, C, \text{self-critique against } C)$$

The constitution $C$ is fixed. The agent learns to follow $C$ better, but never proposes changes to $C$. RSA extends this: the agent not only follows $C$ but also proposes $C' = C + \Delta C$.

### 3.4 Schmidhuber's Gödel Machine

Schmidhuber (2003) proposed the Gödel Machine: a self-improving system that rewrites its own code when it can *prove* that the rewrite is an improvement. The Gödel Machine requires a formal proof of improvement before self-modification. RSA does not: the agent proposes amendments based on *reasoning*, and a human approves based on *judgment*. This is weaker than the Gödel Machine's guarantees but vastly more practical.

### 3.5 Comparison Table

| Property | RL | Constitutional AI | Gödel Machine | **RSA** |
|---|---|---|---|---|
| Policy readable by agent | No | Yes (but fixed) | Yes (formal) | **Yes (natural lang.)** |
| Agent proposes changes | No | No | Yes (with proof) | **Yes (with reasoning)** |
| Approval mechanism | Reward signal | Training | Formal proof | **Human legislature** |
| Modality match | No | Partial | No | **Full** |
| Practical today | Yes | Yes | No | **Yes** |

# 4. Convergence of the RSA Loop

## 4.1 The Improvement Metric

Let $V(S)$ be a value function measuring the quality of specification $S$ (e.g., task completion rate, behavioral alignment score, capability metric). The RSA loop is *improving* if:

$$V(S_{n+1}) > V(S_n) \quad \text{for all } n < n^*$$

for some convergence step $n^*$.

## 4.2 Bounded Improvement Theorem (Informal)

**Claim.** Under the following conditions, the RSA loop converges to a fixed point in finitely many steps:

1. **Bounded specification space**: the set of meaningful specifications is finite (up to semantic equivalence).
2. **Monotone amendments**: $V(\text{RSA}(S)) \geq V(S)$ for all $S$ (no regressions).
3. **Strict improvement below optimum**: if $S \neq S^*$, then $V(\text{RSA}(S)) > V(S)$.

Under these conditions, the sequence $V(S_0), V(S_1), ...$ is a bounded increasing sequence in a finite space, hence converges.

## 4.3 Connection to Self-Improvement Bounds

Nagy (2026, "Provable Bounds on AI Self-Improvement") establishes that AI self-improvement is bounded by the information-theoretic capacity of the improvement channel. In the RSA setting, the improvement channel is the natural-language amendment $\Delta S$. The information capacity of this channel is bounded by:

- The length of $\Delta S$ (finite text)
- The precision of the LLM's behavioral observation (limited by context and reasoning capability)
- The faithfulness of the human legislature's approval (limited by human judgment)

These bounds prevent unbounded self-improvement and ensure the RSA loop converges to a fixed point that reflects the joint capabilities of the LLM (proposer) and human (approver).

## 4.4 Failure Modes

The RSA loop can fail in several ways:

1. **Oscillation**: $S_n \to S_{n+1} \to S_n \to ...$ The agent proposes an amendment, then proposes reverting it. (Caused by inconsistent evaluation criteria.)
2. **Drift**: specifications change without converging. (Caused by non-stationary objectives.)
3. **Collapse**: the specification converges to a trivial fixed point ("do whatever the user says"). (Caused by over-optimization of approval probability.)
4. **Scope creep**: amendments make the specification longer and more complex without improving behavior. (Caused by bias toward addition over deletion.)

Each failure mode has a diagnostic: oscillation is detectable by diffing consecutive specifications, drift by tracking $V(S_n)$, collapse by specification length, and scope creep by specification complexity metrics.

# 5. Implementation: The verification infrastructure System

## 5.1 Architecture

We describe a deployed implementation of RSA in the verification infrastructure system, a Python-based mathematical proof development environment.

The system consists of:

- **Quasimodo**: an LLM agent whose behavior is defined by SKILL.md. Quasimodo performs mathematical research — exploring conjectures, writing proofs, verifying results.
- **proof strategy engine**: an advisory subsystem that Quasimodo consults before starting work. Aggregates proof memory, Mathlib patterns, and difficulty predictions.
- **Forge**: domain-specific computation engines (numerical simulation, CAS, spectral analysis).
- **ProofMemory**: persistent memory of what worked, what failed, and what was tried.

## 5.2 The Evolution of RSA: From Ephemeral to Persistent

RSA did not emerge fully formed. It evolved through three stages, each enabled by a structural change in how behavioral specifications are stored:

**Stage 1: Ephemeral prompt editing (ChatGPT era).** The user manually crafts a system prompt. During conversation, the user asks: "how would you improve your own prompt?" The LLM proposes changes. The user edits the prompt. But the amendment is **ephemeral** — it lives in the chat session and is lost when the session ends. The next session starts from zero. Self-amendment exists but does not accumulate.

**Stage 2: Persistent prompt files (Cursor/IDE era).** The behavioral specification moves from a chat input to a file on disk (SKILL.md, rules, system prompts). Amendments persist across sessions. The agent in session $n+1$ benefits from amendments made in session $n$. Self-amendment now **accumulates**. But the agent still lacks memory of *why* amendments were made — it can read the current specification but not its history.

**Stage 3: Versioned, memory-augmented specification (the proof environment era).** The specification is version-controlled (git), the agent has persistent memory (ProofMemory), and amendment history is tracked. The agent can read not just its current specification but the *diff* from previous versions and the *reasoning* behind past amendments. Self-amendment is now **reflective** in the full sense: the agent understands its own behavioral trajectory.

The quality explosion occurs at the Stage 2 $\to$ Stage 3 transition. Stage 1 is self-amendment without persistence (Sisyphean — the stone always rolls back). Stage 2 adds persistence (the stone stays where you push it). Stage 3 adds memory and versioning (you remember *why* you pushed the stone there and can reason about where to push it next).

## 5.3 The RSA Loop in Practice

A typical RSA cycle in the Stage 3 system:

1. **Observation**: During a proof session, Quasimodo fails to persist intermediate results. The human notices.
2. **Reflection prompt**: Human says: "Look at your SKILL.md. How should it change so that you persist intermediate results automatically?"
3. **Self-reading**: Quasimodo reads its own SKILL.md (possible because SKILL.md is natural language and Quasimodo reasons in natural language).
4. **Gap identification**: Quasimodo identifies that the "Session Close" section lacks a persistence step for intermediate conjectures.
5. **Amendment proposal**: Quasimodo proposes adding: "Before closing a session, write all unproved conjectures to ELYSIUM_STATE.yaml with status conjecture."
6. **Human approval**: Human reviews and approves.
7. **Effect**: Next session, Quasimodo persists conjectures automatically.

### 5.4 Observed Properties

Over approximately 30 RSA cycles across 3 months of operation:

- **Convergence**: The rate of amendments decreased over time. Early sessions produced 3–5 amendments per cycle; recent sessions produce 0–1.
- **Quality**: Behavioral quality (measured by proof completion rate and memory utilization) improved monotonically.
- **No oscillation**: No amendment was later reverted.
- **Scope growth**: SKILL.md grew from ~200 lines to ~700 lines. Growth rate is decreasing, suggesting convergence.
- **Domain specificity**: Amendments clustered around: persistence protocols, oracle consultation patterns, failure handling, and output formatting. No amendments touched core reasoning strategy — the LLM's innate reasoning is not the bottleneck.

### 5.5 The Naming Insight

An unplanned observation: the agent name "Quasimodo" (after Victor Hugo's character who never leaves Notre-Dame) was chosen intuitively *before* the architectural principle of domain confinement was formalized. The agent's confinement to the Python proof environment (never touching raw Lean) emerged later as an explicit design choice, but the naming anticipated it. This suggests that the same-modality property operates bidirectionally: not only can the agent reason about its specification, but the human's intuition can reason about the agent's nature through metaphor.

## 6. Implications

### 6.1 The Human as Legislature

In the RSA paradigm, the human's role shifts from **operator** (issuing commands) to **legislature** (approving or rejecting proposed behavioral amendments). The agent is both the proposer and the subject of legislation. The human provides:

- **Approval authority**: no amendment takes effect without human consent
- **Value alignment**: the human ensures amendments serve the human's goals, not the agent's self-interest

- **Coherence**: the human prevents oscillation and drift by maintaining a consistent evaluation framework

This is structurally analogous to constitutional amendment in governance: the legislature (human) approves changes to the constitution (SKILL.md) proposed by… the government (agent) that operates under it.

### 6.2 Implications for AI Safety

RSA introduces a new consideration for AI safety: the agent can *propose* changes to its own constraints. While the human retains approval authority, the proposal itself may be persuasive, misleading, or subtly self-serving. Safeguards:

1. **Diff-based review**: all amendments are presented as diffs, not as new specifications.
2. **Behavioral testing**: amended specifications can be tested in sandboxed sessions before deployment.
3. **Amendment logging**: all proposals (approved and rejected) are logged for audit.
4. **Constraint preservation**: certain specification elements (safety constraints, scope boundaries) are marked as immutable — the agent can propose amendments to everything else but not to these.

### 6.3 Implications for the Cascade Theorem

RSA completes the cascade theorem (Nagy, 2026) by identifying its fixed point. The cascade proceeds:

$$\text{instances} \xrightarrow{\text{Level 1}} \text{tools} \xrightarrow{\text{Level 2}} \text{tool networks} \xrightarrow{\text{Level 3}} \text{tool network architectures} \xrightarrow{\text{RSA}} \text{self-amendment}$$

At the RSA level, the cascade becomes self-referential: the tool that improves the tool network is part of the tool network. This is the cascade's fixed point — the level where further investment in meta-tools reduces to improving the system that is already doing the improving.

The cascade does not continue beyond RSA because there is no meaningful "Level RSA + 1": a tool that improves the self-improver is just… a better self-improver. The fixed-point structure is:

$$T_{\text{RSA}}(T_{\text{RSA}}) = T'_{\text{RSA}}$$

and convergence means $T'_{\text{RSA}} \approx T_{\text{RSA}}$ — the system stabilizes.

## 7. Future Direction: Autonomous RSA (Stage 4)

The RSA loop described in this paper is human-triggered: the human notices a behavioral gap and prompts the agent to self-amend. A natural extension is *autonomous RSA*: the agent itself detects recurring patterns and proposes amendments without human prompting.

**Stage 4 architecture.** The agent maintains a log of its own failures and difficulties. When a pattern recurs $N$ times (for some threshold $N$), the agent:

1. Detects the recurrence from its own logs

2. Classifies the pattern as recurring (not one-off)
3. Proposes either: (a) a SKILL amendment, (b) a new rule, or (c) a new tool
4. Presents the proposal to the human legislature for approval

The human's role shifts further: from triggering amendments to *approving or vetoing* autonomously generated proposals. The agent becomes proactive rather than reactive.

**The over-complexity risk.** Autonomous RSA can generate unbounded complexity: if the agent creates a tool or rule for every minor recurrence, the specification grows without bound. This is the "scope creep" failure mode (§4.4) applied at system scale. Mitigations:

- **Recurrence threshold**: only propose amendments for patterns that recur $\geq N$ times.
- **Cost-benefit filter**: estimate the cost of the amendment (specification complexity increase) against the benefit (expected failures prevented). Only propose when benefit exceeds cost.
- **Sunset clause**: amendments proposed under Stage 4 carry an expiration date. If the amendment has not been triggered within $M$ sessions, it is automatically proposed for removal.
- **Human veto**: the legislature retains the right to reject proposals without justification.

This is not yet implemented and remains a research direction. The key open question: can the agent accurately distinguish *recurring structural problems* (worth a tool) from *incidental difficulties* (not worth the complexity cost)? This is a classification problem whose solution likely requires several RSA cycles of its own.

## 8. Related Work

- **Good (1965)**: "Ultra-intelligent machine" — the first explicit discussion of recursive self-improvement. RSA is a concrete, bounded implementation of Good's idea.
- **Schmidhuber (2003)**: Gödel Machine — self-improvement with formal proofs. RSA trades formal guarantees for practical applicability.
- **Bostrom (2014)**: Intelligence explosion via recursive self-improvement. Our convergence results (§4) bound the "explosion" — RSA converges, not diverges.
- **Bai et al. (2022)**: Constitutional AI — self-critique against a fixed constitution. RSA extends this to self-amendment of the constitution itself.
- **Smith (1984)**: Computational reflection — programs that can inspect and modify their own structure. RSA is computational reflection in the natural language modality.
- **Maturana & Varela (1980)**: Autopoiesis — self-creating biological systems. RSA is autopoiesis for behavioral specifications.
- **Nagy (2026)**: Cascade Depth Theorem — RSA is the fixed point of the cascade.
- **Nagy (2026)**: Provable Bounds on AI Self-Improvement — provides the theoretical ceiling for RSA convergence.

## 9. Conclusion

Reflective self-amendment is a capability unique to the LLM + text-specification paradigm. It arises because the agent's policy and reasoning share a representational modality — natural language — enabling the agent to read, critique, and amend its own behavioral specification.

We showed that RSA is the fixed point of the tool-building cascade: the level at which the meta-tool is directed at itself. We gave convergence conditions under which the RSA loop stabilizes,

connecting to known self-improvement bounds. And we described a deployed implementation where an LLM agent (Quasimodo, in the verification infrastructure proof system) has undergone approximately 30 self-amendment cycles, converging toward a stable behavioral specification with monotonically improving performance.

The central insight is this: in every prior AI paradigm, the policy and the reasoning engine were in different modalities — the agent could not understand its own behavioral specification. LLMs changed this. And once the agent can understand its own specification, self-amendment becomes not just possible but natural — the obvious next step in the tool-building cascade.

The question for AI governance is not whether RSA will happen — it is already happening. The question is whether we design the legislature well.

## References


- Bai, Y. et al (2022). Constitutional AI: Harmlessness from AI Feedback. *arXiv:2212.08073*.
- Bostrom, N (2014). Superintelligence: Paths, Dangers, Strategies. *Superintelligence: Paths, Dangers, Strategies*.
- Good, I. J (1965). Speculations Concerning the First Ultraintelligent Machine. *Advances in Computers*, 31-88.
- Maturana, H. R., & Varela, F. J (1980). Autopoiesis and Cognition: The Realization of the Living. *Autopoiesis and Cognition: The Realization of the Living*.
- Nagy, T. (2026). The Latent Architecture Theorem: Why Every Computer Is Suboptimal and How to Fix It. *Working paper*.
- Nagy, T. (2026). Provable Bounds on AI Self-Improvement: The Verification Oracle Ceiling. *Working paper*.
- Schmidhuber, J (2003). Gödel Machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements. *arXiv:cs/0309048*.
- Smith, B. C (1984). Reflection and Semantics in a Procedural Language. *MIT Technical Report 272*.
- von Neumann, J (1966). Theory of Self-Reproducing Automata. *Theory of Self-Reproducing Automata*.