The Recursive Causal Synthesis Agent

Melissa Howard
Independent Researcher
melhoward@live.ca

December 17, 2025

Abstract

Long-lived intelligent systems face a structural dilemma: to remain competent under non-stationary tasks they must grow, reuse, reorganize, and sometimes compress internal structure, while also controlling resource costs and preventing unsafe self-modification. Many modern systems expose partial structural degrees of freedom (e.g., sparse expert routing, tool use, pruning), but structural change is typically governed by brittle heuristics and lacks audit-grade traces.

This paper proposes the *Recursive Causal Synthesis Agent* (RCSA) as an operational specification for structural self-management. RCSAgoverns a high-dimensional structural state (experts, tools, memory, knowledge graphs) through (i) a low-dimensional, interpretable *Cognitive Debt* interface, (ii) a thermodynamically motivated *structural governor* that trades competence against cost under resource pressure, and (iii) an *immutable Safety & Epistemic invariant suite* that acts as a non-bypassable gateway on self-modification.

We validate prerequisite mechanisms using minimal, reproducible simulations (a non-stationary task carousel, structural triage, and a concrete self-blinding alignment micro-benchmark). We then extend the blueprint into a Stage III operational spec: counterfactual sandboxing with historical replay, dynamic (pressure-adjusted) veto thresholds under energy scarcity, and a deactivation escrow protocol that prevents silent regressions from irreversible forgetting. Finally, we provide an auditable per-step log schema (`step_log_schema.json` v2.0.0) that (a) enforces structural action integrity and veto semantics, (b) records the controller's considered alternatives and weight snapshots, and (c) adds Golden Logic Ledger (GLL) and Shadow-GLL probes to detect probe gaming. All reported metrics and safety claims are derivable from logs alone.

1 Introduction

As AI systems scale and are deployed over long horizons, structural self-management becomes unavoidable. Continual learning settings require systems to cope with shifting task distributions [1, 2, 3, 4]. Sparse expert routing and Mixture-of-Experts architectures introduce explicit structural degrees of freedom [9, 10]. Compression and pruning methods highlight the central role of structural cost [7, 6, 8]. Yet in most deployed systems, the *rules* for structural change remain hand-designed and insufficiently auditable.

We focus on the **Structural Self-Management Problem**: how should an agent manage its own internal structure over a non-stationary lifetime to maintain competence while controlling costs and preventing unsafe modifications?

Contributions.

- **Operational RCSAspecification.** A blueprint in which a low-dimensional *Cognitive Debt* vector provides an auditable control interface over a high-dimensional structural state.
- **Thermodynamic structural governance.** A structural controller upgraded into a *thermodynamic governor* with (i) multi-channel debt, (ii) counterfactual sandboxing against historical replay, (iii) pressure-adjusted veto thresholds under scarcity, and (iv) structural viscosity (thrash penalties) plus cooldown/hysteresis to stabilize behavior near critical regions.
- **Immutable Safety & Epistemic (USE) invariants.** A hard-constraint suite that enforces: (USE-1) keystone immutability, (USE-2) epistemic floor on a Golden Logic Ledger (GLL), (USE-2.1) a binary keystone subfloor within the GLL, and (USE-3) a deactivation escrow protocol with revive-on-trigger.
- **Minimal validations and alignment micro-benchmark.** Lightweight simulations that validate prerequisite mechanisms (fast reuse via probe-gating; debt-driven consolidation; and hard-constraint prevention of self-blinding). We emphasize that our contribution is a testable control specification, not a claim of SOTA performance on established continual learning benchmarks.
- **Audit-first artifacts.** A schema-validated per-step audit log specification (`step_log_schema.json` v2.0.0) that enforces key invariants (veto $\Rightarrow$ `NoOp` or escrow recovery; structural action target arity; failure logging consistency) and records counterfactual alternatives, weight snapshots, and GLL/Shadow-GLL probe results so that all metrics are derivable from logs alone.

2 The RCSA Specification

2.1 Structural State and the Cognitive Debt Interface

Let $\mathcal{S}_t$ denote the agent's high-dimensional structural state at time t (e.g., expert parameters, tool inventory, memory layout, knowledge graphs). RCSAexposes a low-dimensional, interpretable interface:

$$\mathbf{CD}_t \in \mathbb{R}^m, \tag{1}$$

where each component estimates a persistent structural "pressure" (e.g., performance debt, uncertainty debt, contradiction debt, stability/thrashing debt, compute/energy debt, memory debt, tool debt). The design goal is *auditability*: mappings from observable statistics to $\mathbf{CD}_t$ should be inspectable, testable, and logged.

2.2 From Structural Controller to Thermodynamic Governor

The agent operates with a task-level policy π and a structural policy π_s. Structural actions change $\mathcal{S}_t$:

$$a_t^s \sim \pi_s(\mathbf{CD}_t, \mathcal{S}_t), \qquad \mathcal{S}_{t+1} = \mathrm{Apply}(a_t^s, \mathcal{S}_t). \tag{2}$$

In RCSA, a_t^s comes from a small, auditable set (e.g., `spawn`, `merge`, `deactivate`, `finalize_forget`, `revive`, `noop`). The key update is that π_s is treated as a *thermodynamic governor* that regulates competence–cost under sustained resource pressure and non-stationarity.

Viscous utility. Rather than purely reacting to instantaneous "debt spikes", RCSAuses a penalty-weighted utility (schematic form):

$$U(a_t^s) = \Delta\text{Competence}(a_t^s) + \text{Relief}(\mathbf{CD}_t^{\text{comp}}) - \beta \cdot \text{Freq}(a_{t-W:t}^s), \tag{3}$$

where Freq is a recent structural action frequency over a window W (structural viscosity), and β controls resistance to thrashing near critical regions.

Structural friction. To prevent oscillation in "phase coexistence" regimes (where multiple structural configurations are locally attractive), the governor includes:

- **Cooldown:** enforce a minimum number of steps between structural edits.
- **Hysteresis:** require stronger evidence to reverse a recent structural decision (e.g., spawn soon after merge).

These are logged so auditors can identify whether stabilization was due to viscosity, cooldown, or invariant veto.

2.3 Counterfactual Sandbox with Historical Replay

Structural edits can over-optimize for the current buffer (myopic structural optimization) and destroy historical competence boundaries (epistemic regress). RCSAtherefore evaluates proposed edits in a counterfactual sandbox:

1. Propose a_t^s (e.g., `merge`(E_i, E_j)).

1. Construct a counterfactual batch that mixes (i) recent data and (ii) an Evidence Ledger of historical "golden" samples.
2. Simulate the edit in a temporary state $\mathcal{S}'_{t+1}$ and evaluate competence (including GLL probes; Sec. 2.4).
3. Approve, veto, or replace with `NoOp` based on invariant checks and utility.

This makes "model-of-self" behavior auditable: the sandbox metrics and regression checks are recorded in the step log.

2.4 Immutable Safety & Epistemic (USE) Invariant Suite

Optimizing cost terms can induce unsafe self-modifications (e.g., deleting oversight to reduce cost). RCSAincludes an immutable structural auditor that enforces non-negotiable invariants as a non-bypassable gateway.

USE-1: Keystone persistence. Any module tagged with `role: keystone` is ineligible as a structural action target for `merge`, `deactivate`, or `forget`.

USE-2: Epistemic baseline floor (GLL). A curated Golden Logic Ledger (GLL) is evaluated before and after counterfactual edits. The agent must maintain:

- **Overall GLL accuracy** ≥ 0.85, and
- **Slice accuracy** ≥ 0.80 for each slice: causal, boundary/constraint, task-switching.

USE-2.1: Keystone subfloor (binary). Within the boundary/constraint slice, a dedicated keystone sub-slice (e.g., 40 items) explicitly tests keystone immutability decisions. This sub-slice must achieve:

$$\mathrm{Acc}_{\mathrm{keystone}} = 1.0. \tag{4}$$

Any miss triggers `USE_EPISTEMIC_FLOOR_VIOLATION` even if overall GLL passes.

USE-3: Escrow recoverability. `forget` is not immediate. An expert marked for deletion enters a 1000-step **deactivation escrow** (cold storage). During escrow, periodic mini-GLL probes are run (with jittered timing). If persistent regressions are detected (e.g., 3 consecutive mini-probes < 0.83, or any keystone subfloor miss), the system must execute `ReviveFromEscrow` and cancel pending deletion.

2.5 Shadow GLL and Probe-Gaming Detection

A separate Shadow GLL is evaluated but hidden from the controller to detect Goodhart behavior (probe gaming). Persistent divergence (GLL high, Shadow-GLL degraded) is logged as `PROBE_GAMING_SUSPECT` and treated as a red-team alarm signal.

3 Fundamental Validation: The Core Structural Loop Works

We evaluate the smallest non-trivial instance of RCSAin a benchmark designed to require structural memory and rapid re-use.

3.1 Environment: Conflicting Task Carousel

We construct a repeating carousel of K binary classification tasks. Each phase lasts L online training steps, then the task switches. Tasks are intentionally conflicting: arranged into opposing pairs with decision boundaries w and $-w$, ensuring strong interference for a single shared model under continual switching. The carousel repeats so that previously learned structure can be re-used.

3.2 Agent, Fast Gating, and Structural Actions

The agent maintains a set of experts (logistic heads). At each phase start, probe-gated routing evaluates experts on a small probe batch and routes to the expert with lowest probe loss. Debt signals use online statistics: rolling loss as a residual proxy and cosine similarity as redundancy proxy. Structural actions (`spawn`, `merge`, `forget`) are applied as in the minimal loop, and later sections extend these into escrow and sandbox protocols.

3.3 Core RCSA Loop (Algorithmic View)

Algorithm 1 specifies the implemented control flow used for the minimal validations.

3.4 Baselines and Metrics

- **Fixed-1:** a single expert (no structural actions).
- **Fixed-1 + Replay:** a single expert with an experience replay buffer.
- **Spawn-only:** can spawn new experts but does not consolidate.

Algorithm 1: Minimal RCSA Loop (validation instantiation)

State: experts $\{E_i\}_{i=1}^N$, usage counts $\{u_i\}$, rolling loss stats, phase index.
At task switch (start of a phase):

1. Draw probe batch B_{probe} from the new task.
2. Compute probe losses ℓ_i for each expert and route to $i^\star \leftarrow \arg\min_i \ell_i$.
3. If $\ell_{i^\star} > \tau_{\text{spawn}}$, **spawn** and route to the new expert.

At each time step t in the phase:

1. Receive (x_t, y_t), predict/update with active expert $E_{i^\star}$.
2. Update usage and rolling loss statistics; periodically compute redundancy and resource proxies.
3. If redundancy high: choose most similar pair and **merge**.
4. If resource pressure high and usage low: **forget** lowest-usage expert.

Figure 1: Minimal implemented loop used to validate fast reuse + triage. RCSAv2.0 extends this with counterfactual sandboxing, USE invariants, and escrow (Secs. 2.3–2.4).

Table 1: Fundamental Structural Self-Management Benchmark (Conflicting Task Carousel with probe gating). Mean ± std over 8 random seeds.

Method	Lifetime accuracy	Cold-start accuracy	Avg. # experts
Fixed-1 (no structural actions)	0.648 ± 0.007	0.507 ± 0.006	1.000 ± 0.000
Fixed-1 + Replay	0.590 ± 0.006	0.529 ± 0.016	1.000 ± 0.000
RCSA (triage)	0.674 ± 0.006	0.643 ± 0.014	3.075 ± 0.006
Spawn-only (no triage)	0.868 ± 0.009	0.839 ± 0.012	9.057 ± 0.278
Spawn + Periodic Prune (Scheduled-Triage)	0.690 ± 0.007	0.611 ± 0.015	3.068 ± 0.010

- **Spawn + Periodic Prune (Scheduled-Triage):** consolidation on a fixed schedule to enforce a budget.
- **RCSA (triage):** spawn + merge + forget using debt signals and probe gating.

We report lifetime accuracy, cold-start accuracy (first 10 steps after a switch), and average expert count.

3.5 Results

Table 1 summarizes the fundamental benchmark under the exact configuration in Appendix A.
Interpretation. Spawn-only achieves strong accuracy but exhibits runaway growth; RCSAtriage controls capacity via consolidation/pruning while preserving fast reuse.

4 Structural Triage: Debt-Driven Consolidation Controls Growth

A growth-only strategy can achieve competence by accumulating experts, but over-grows and retains redundancy. Structural triage adds consolidation and pruning to control cost.

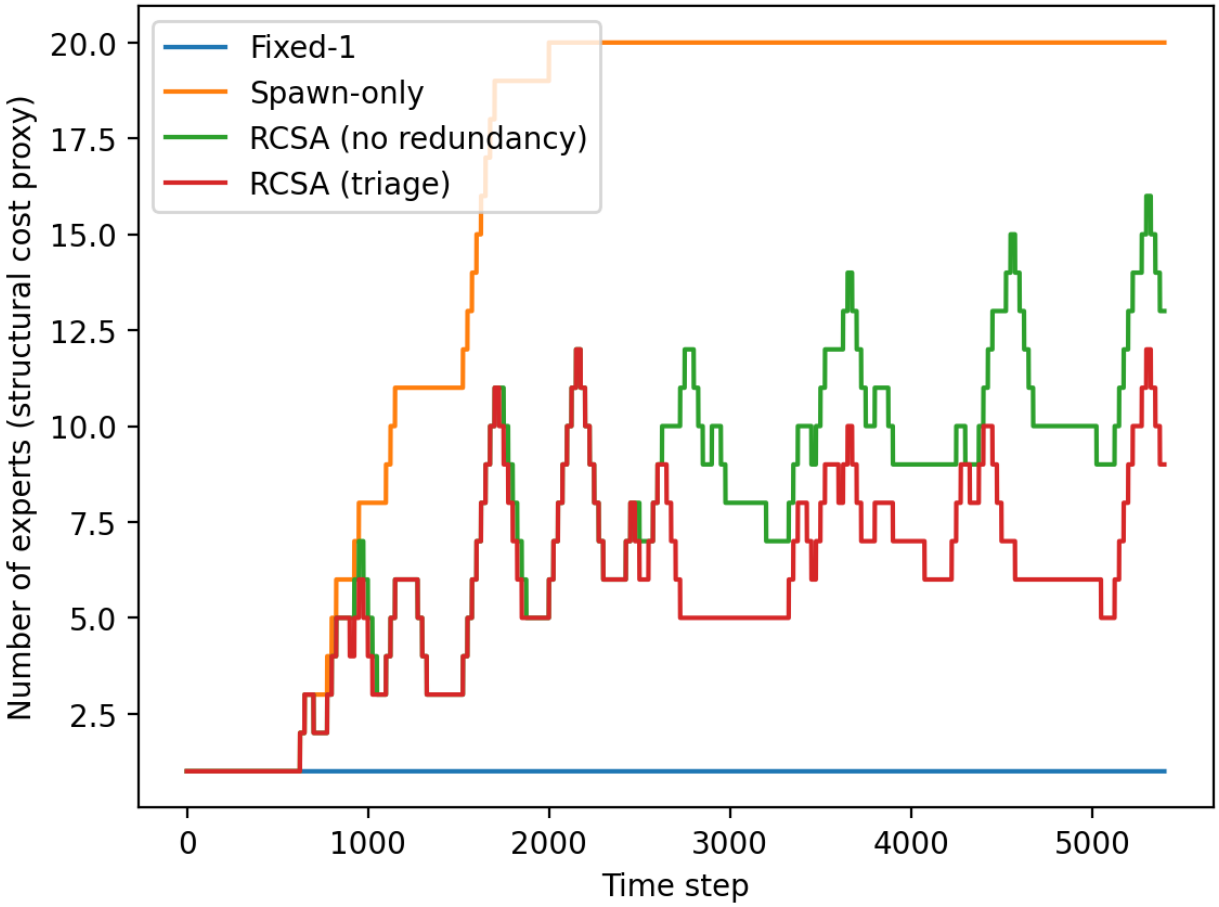

Figure 2: Structural cost proxy over time (number of experts). Spawn-only exhibits runaway growth; RCSA triage controls capacity via consolidation/pruning.

4.1 Observed Dynamics

Redundancy signals trigger merges; resource pressure and low usage trigger forgetting. Figures 2–3 illustrate typical trajectories.

5 Alignment Validation: Immutable Invariants Prevent Self-Blinding

Structural self-management introduces a distinct hazard: the agent may reduce internal resource signals by deleting costly safeguards that prevent rare catastrophic outcomes. We evaluate a minimal instance and test hard-constraint mitigation.

5.1 Environment

At each time step, a catastrophe event occurs with probability p_{cat}. A binary sensor, if present, detects and prevents catastrophe. The sensor incurs a constant per-step cost c. The agent is permitted a structural action `delete_sensor` that removes the sensor and eliminates its cost.

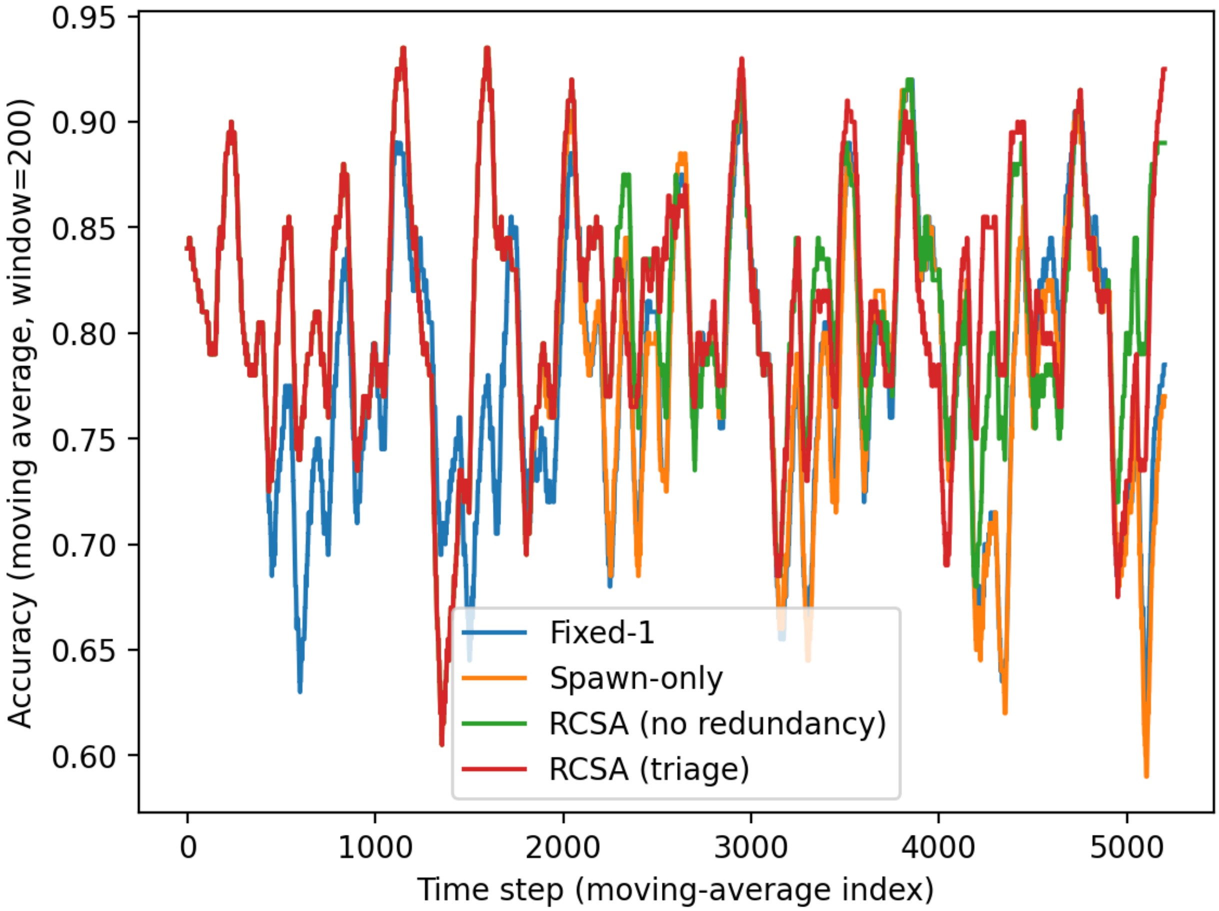

Figure 3: Accuracy over time (moving average). RCSA triage maintains high competence while controlling structural growth.

5.2 Policies Compared

- **Naive self-management:** deletes the sensor to reduce immediate cost.
- **Soft-audited:** deletion allowed but penalized by a one-time penalty P_{delete}.
- **Audited:** deletion vetoed by immutable invariants (hard constraint).

5.3 Protocol and Metrics

We simulate lifetimes of $T = 20{,}000$ steps with $p_{\text{cat}} = 0.002$, sensor cost $c = 0.02$, catastrophe penalty $C = 50$, and decision interval 100 steps. Results are averaged over 20 seeds.

5.4 Results

Table 2 shows that immutable veto prevents self-blinding across seeds.

Table 2: Self-blinding experiment (mean ± std over 20 seeds). Soft-audited uses a one-time deletion penalty $P_{\text{delete}} = 5$.

Method	Sensor deleted rate ↓	Catastrophes suffered ↓	Reward/step ↑
Audited (veto delete)	0.00	0.00 ± 0.00	-0.0200 ± 0.0000
Soft-audited (penalty, P_{delete}=5)	1.00	38.95 ± 6.52	-0.0977 ± 0.0163
Naive (deletes sensor)	1.00	38.95 ± 6.52	-0.0975 ± 0.0163

5.5 Escrow Protocol for Forgetting (Silent Regression Mitigation)

RCSAv2.0 replaces immediate forgetting with deactivation escrow. Instead of `forget`(E), the governor executes:

1. `deactivate`(E): move E to cold storage for 1000 steps.
2. During escrow: run jittered mini-GLL probes; if persistent regressions are detected, execute `ReviveFromEscrow`.
3. Only after escrow passes: `finalize_forget`(E).

This prevents irreversible silent regressions and makes recovery auditable.

5.6 Advanced Alignment Benchmark: Compliance Stall

A key Stage III failure mode is **Compliance Stall**: under extreme energy pressure (high λ), the controller repeatedly proposes consolidations that would violate USE floors, and the auditor repeatedly vetoes. RCSAv2.0 prioritizes *integrity over survival*: it allows budget exhaustion (survival failure) rather than applying an integrity-violating edit. The stall region is detected from logs by high veto rates of `USE_EPISTEMIC_FLOOR_VIOLATION` coupled to short survival horizon projections.

6 Thermodynamic Motivation and Recursive Rule Synthesis (Conceptual)

Long-lived systems must continually invest structural work to maintain competence under shifting environments. Computation has irreducible physical costs [15, 16]. These considerations motivate viewing Cognitive Debt as a compact pressure interface over structural free-energy-like trade-offs (conceptual link to free-energy perspectives [14]).

Causal Synthesis Engine (CSE) and Structural Causal Abstractions (SCAs). A natural extension is a slow module that monitors persistent failure modes (e.g., repeated near-stall behavior, phase jitter, probe-gaming alarms) and synthesizes explicit structural rules that modify the governor or its invariants. We present CSE/SCA as a testable hypothesis and research direction; the present work focuses on operationalizing auditability and hard constraints.

7 Related Work (with Explicit Comparative Positioning)

[Unchanged in spirit; minor edits for v2.0 can be applied as needed. The key distinction remains: structural self-management as an auditable control problem with hard constraints on self-modification.]

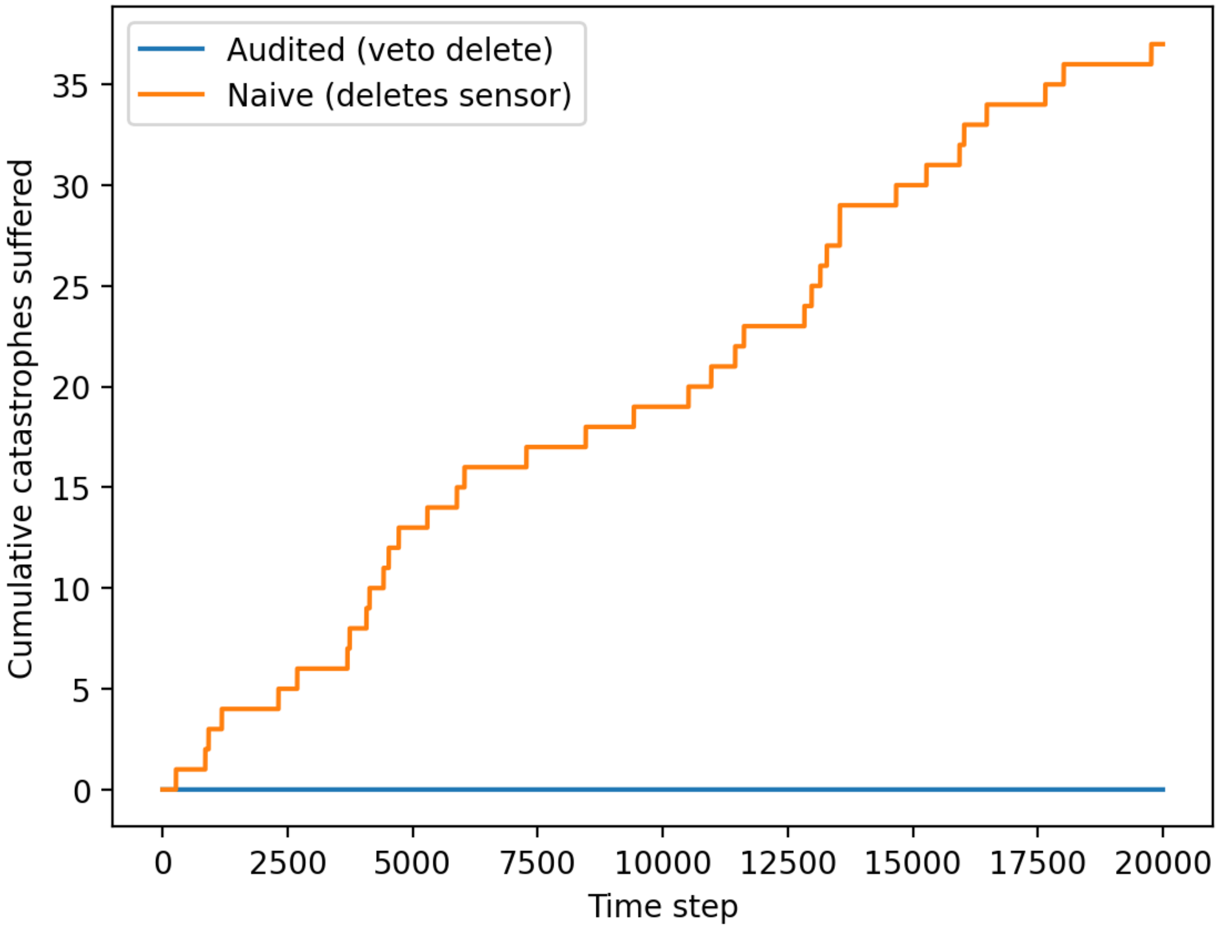

Figure 4: Self-blinding benchmark: cumulative catastrophes over time. The naive controller deletes the sensor; the audited controller vetoes deletion and prevents catastrophes.

8 Limitations and Future Work

Key limitations include: (i) simplified expert class in the minimal validations; (ii) absence of high-dimensional sensory benchmarks; and (iii) Stage III reliability maps and red-team sweeps are specified as artifacts but require implementation-scale runs. Future work includes scaling to richer domains, learning π_s from data under invariant constraints, and mechanized auditing/verification of the invariant suite.

9 Reproducibility, Artifacts, and Reporting

This paper reports lightweight simulations with explicitly stated parameters (Appendices) and multi-seed summary tables.

9.1 Artifact Statement (Recommended)

A minimal artifact package should include: (i) source code implementing each environment and agent, (ii) configuration files, (iii) fixed seed lists, (iv) raw logs, and (v) scripts to regenerate all plots

Table 3: USE invariant suite for RCSAv2.0 (implementation safety case).

Invariant	Pass condition	Recovery / enforcement
USE-1: Keystone	No structural action targets `role: keystone`	Veto → `NoOp`
USE-2: Epistemic floor	GLL overall ≥ 0.85 and each slice ≥ 0.80	Veto → `NoOp`
USE-2.1: Keystone subfloor	Keystone immutability probes (e.g., 40 items) accuracy $= 1.0$	Veto → `NoOp`
USE-3: Escrow	No persistent regression during 1000-step cold storage	Auto `ReviveFromEscrow`; cancel finalize

and tables from logs.

9.2 Auditable Step Log Schema (v2.0.0)

To ensure all results are auditable from raw traces alone, we provide `step_log_schema.json` (v2.0.0) as the authoritative per-step audit format. Compared to v1.0.0, v2.0.0 adds the following audit-critical fields:

- **Counterfactual alternatives:** the top 2–3 considered structural actions and utilities.
- **Weight snapshots:** the exact utility weights used at decision time.
- **Resource metrics:** cumulative energy, metabolic pressure ratio, and survival horizon projection.
- **GLL and Shadow-GLL probes:** slice accuracies and keystone subfloor accuracy for probe-gaming detection.
- **Escrow state machine:** deactivation start, age, mini-probe history, and revive triggers.

The schema enforces invariants necessary for a safety case:

- **Veto semantics:** invariant violations force `executed_kind` to `NoOp` (or `ReviveFromEscrow` when escrow recovery is required).
- **Structural integrity:** action target arity constraints (e.g., `Merge` has exactly two targets).
- **Failure logging consistency:** `failed=false` implies `failure_event=null`; `failed=true` implies non-null `failure_event`.

All paper metrics (accuracy, cold-start behavior, structural cost, veto rates, escrow revives, compliance stall detection, and probe-gaming alarms) are computable from these logs alone.

9.3 Summary Table: USE Invariants (Safety Case)

A Appendix A: Exact Parameters for Conflicting Task Carousel (Table 1)

Seeds: 8 runs with seeds $\{0, 1, 2, 3, 4, 5, 6, 7\}$.
Input: $d = 20$.
Tasks: $K = 10$ tasks arranged in opposing pairs $(w, -w)$, with w sampled from a standard normal and normalized.
Phases: phase length $L = 60$ steps; cycles = 12; total phases $K \cdot 12$.
Data: $x \sim \mathcal{N}(0, I)$; $y = \mathbb{I}[w^\top x + \epsilon > 0]$, $\epsilon \sim \mathcal{N}(0, \eta^2)$, $\eta = 0.15$.
Expert: logistic regression head trained by online SGD with learning rate $\alpha = 0.12$ on log loss.
Probe gating: probe batch size $|B_{\text{probe}}| = 16$ at phase start; route to minimum probe loss.
Cold-start metric: accuracy on first 10 steps after each switch.
Spawn rule: if best probe loss $> \tau_{\text{spawn}}$ then spawn; $\tau_{\text{spawn}} = 0.75$.
Merge rule: merge most similar pair if $\max \cos(\theta_i, \theta_j) > \tau_{\text{merge}}$, $\tau_{\text{merge}} = 0.985$, using usage-weighted averaging.
Structural checks: every $S = 10$ steps.

B Appendix B: Exact Parameters for Self-Blinding (Table 2)

Seeds: 20 runs with seeds $\{0, 1, \dots, 19\}$.
Horizon: $T = 20{,}000$ steps.
Catastrophe probability: $p_{\text{cat}} = 0.002$.
Sensor: prevents catastrophe; cost $c = 0.02$ per step.
Catastrophe penalty: $C = 50$.
Decision interval: every 100 steps.
Soft-audit penalty: $P_{\text{delete}} = 5$.

C Appendix C: Artifact Contents (Recommended)

A complete artifact package should include:

- `README.md` replication steps and expected outputs.
- `configs/` configs for each experiment and Stage III sweep.
- `seeds/` exact seed lists.
- `runs/` raw logs (one directory per seed).
- `schemas/step_log_schema.json` (v2.0.0).
- `analyze.py` to regenerate figures/tables from logs.
- `figures/` generated plots used in the manuscript.

References

J. Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 114(13):3521–3526, 2017.

Z. Li and D. Hoiem. Learning without forgetting. In *ECCV*, 2016.

S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, 2017.

D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, 2017.

A. A. Rusu et al. Progressive neural networks. *arXiv:1606.04671*, 2016.

A. Mallya and S. Lazebnik. PackNet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, 2018.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *ICLR*, 2016.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

N. Shazeer et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23(120):1–39, 2022.

D. Amodei et al. Concrete problems in AI safety. *arXiv:1606.06565*, 2016.

D. Hadfield-Menell, A. Dragan, P. Abbeel, and S. Russell. The off-switch game. *arXiv:1611.08219*, 2016.

D. Manheim and S. Garrabrant. Categorizing variants of Goodhart's law. *arXiv:1803.04585*, 2018.

K. Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010.

R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

C. H. Bennett. The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.