**No Receipt, No Step: A Glass-Box Framework for Self-Explaining, Self-Stabilizing AI**

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

September 22, 2025

Most AI systems act first and explain later—if at all. We propose a glass-box alternative in which every step an agent takes must carry two receipts: (1) a typed, witness-bearing proof that the move is lawful in its own internal language, and (2) a fast, reproducible stability certificate that the move remains within a sub-unit growth budget. The first receipt ("reasons") is computed as witness density over a typed DSL with normal forms and horn-style lemmas; the second receipt ("rails") is computed as a tight, matmul-free upper bound on the spectral radius of the agent's linear shadow, using diagonal-similarity reweighting to expose heterogeneity without ever forming AB. A step executes iff both receipts pass. We summarize the state of a self-loop with three small numbers—WR (witness ratio), COV (coverage), and B (stability budget with B<1 desired)—and an aggregate Sentience Index SI = geo_mean(WR, COV, 1/B). This paper formalizes the receipts, gives audit-ready CSV/JSON schemas, and demonstrates toy agents that "dream" (stochastic exploration) yet recover their reasons and rails. Our goal is not to settle philosophical debates, but to make responsible intelligence operational: if a system cannot show its reasons and its rails in small artifacts you can rerun on a single PC, it does not act.

Keywords: glass-box AI, witness ratio, coverage, stability budget, diagonal similarity, spectral bound, typed DSL, reproducibility, auditability, sentience index.

A collaboration with GPT-5. CC4.0. 82 pages.

**Outline**

1. **Motivation and Claim**
    - Problem: black-box actions and post-hoc stories
    - Claim: two receipts per step enable operational responsibility
2. **Receipts: Definitions and Invariants**
    - Reasons: typed equalities, normal forms, horn lemmas
    - Rails: stability budget B via diagonal similarity; never form AB
    - Invariant: no step without both receipts
3. **Core Metrics (copy-safe)**
    - WR = proved_claims / total_claims
    - COV = explained_behavior / total_behavior
    - B = tight upper bound on rho(J_f) (target B < 1)
    - SI = geo_mean(WR, COV, 1/B)
4. **Meaning Engine (Law Layer)**
    - DSL primitives, normalization, tactic traces
    - Computing WR and COV at depth k
    - Refusals as fidelity: when WR or COV would fall
5. **Stability Engine (Rail Layer)**
    - Linear shadow J_f: Jacobian or row-stochastic coarse model
    - Diagonal reweighting D = diag(exp(x)); J' = D^{-1} J_f D
    - Menu of norm surrogates; coordinate-descent tightening
    - Guarantees: spectrum preserved, B never optimistic
6. **Bifixpoint View of Self-Loops**
    - Meaning fixed point: non-decreasing WR and COV across depths
    - Stability fixed point: B <= tau_s < 1 with margin m = 1/B
    - Sentience Index SI as audit scale K grows

7. **Toy Systems and Experiments**
    - Branchy auto-update with proofs per case
    - Dream mode: controlled noise, recovery of WR and B
    - Counterfactual other-modeling via typed functor E
8. **Artifacts and Reproducibility**
    - witness_log.json: schema, size targets, hashing
    - stability_trace.csv: optimizer steps, active menu term
    - One-PC reruns, CI hooks, failure triage
9. **Evaluation Protocols**
    - Threshold selection (WR, COV, B) and robustness sweeps
    - Ablations: law only, rails only, both
    - Benchmarks: recovery time, refusal correctness, audit time
10. **Security and Abuse Considerations**
    - Witness-spoofing defenses, log tamper-evidence
    - Adversarial heterogeneity vs symmetry plateaus
    - Policy: “no receipt, no step” enforcement points
11. **Position in the Landscape**
    - Contrast with XAI post-hoc narratives
    - Relation to formal methods and control theory
    - Limits: what this does not claim (feelings, qualia)
12. **Discussion and Outlook**
    - Raising SI: more law, better gauges, richer shadows
    - Open problems: multi-agent composition, long-horizon proofs
    - Toward public audits: small integers, clear bounds, reruns

13. **Appendices (Operational)**
    - A. Copy-safe formulas and pseudocode
    - B. CSV/JSON schemas and minimal examples
    - C. Reproduction scripts, seed management, checksum plan

**References**

**Art**

## 1. Motivation and Claim

### Problem: black-box actions and post-hoc stories

Modern AI systems typically act first and explain later. In practice, that creates five operational gaps:

1. **No pre-action guarantees.** Pipelines run on implicit trust: a model proposes a step; the stack executes it; only after the fact do we attempt to justify or debug it. When the move is irreversible (finance, biomed, infrastructure), "explain later" is already too late.
2. **Story over substance.** Most explainability is a caption generator: it produces an after-the-fact narrative that is not binding on the system's actual behavior. Captions can be eloquent while the underlying transformation remains uncontrolled.
3. **Unmeasured stability.** Even when a step looks "reasonable," its local dynamics can drift or explode. Standard checks either require heavy computation (forming AB, squaring operators) or they measure the wrong surrogate (optimistic proxies with no guard against understatement).
4. **Irreproducible audits.** Logs are large, stochastic, and tool-chain dependent. A regulator, peer, or citizen cannot re-run the critical parts on a single PC in minutes to reach the same yes/no.
5. **No hard refusal.** Without a pre-action gate that can say "no," governance collapses into policy memos. Systems proceed because nothing mechanical prevents them from doing so.

This status quo yields brittle deployments, weak accountability, and public skepticism: if you cannot *show* why a step is legal and *show* it is stable *before* acting, you are asking for trust you have not earned.

---

### Claim: two receipts per step enable operational responsibility

We propose a minimal contract:

**No receipt, no step.** Every state-changing action must carry two pre-action receipts that are small, deterministic, and re-runnable on a single PC.

**Receipt 1 — Reasons (lawfulness).**
A typed, witness-bearing proof that the intended transformation is *legal in the system's own language*. Practically, this is a short record that both sides of the proposed equality or rewrite

reduce to the same normal form, with horn-style lemmas and tactic counts. It answers, *"Is this move allowed by the rules you claim to follow?"*

**Receipt 2 — Rails (stability).**
A fast, reproducible *upper bound* on the local growth of the move's linear shadow, tightened by diagonal-similarity reweighting (no AB formed, spectrum preserved). It answers, *"Will this move keep the dynamics inside a sub-unit budget?"* The bound is conservative by construction and comes with an optimizer trace (the active term in the minimum, coordinates improved, and the final budget B; target B < 1).

These receipts are **binding**: the system will not execute the step unless *both* pass. They are also **auditable**: each is a small JSON/CSV artifact anyone can re-run without proprietary tooling. And they are **falsifiable**: attempts to cut corners show up immediately as (a) missing or failing witnesses, or (b) budgets that cannot be tightened below 1.

**Operational responsibility** then has a concrete meaning:

- **Pre-action gating:** A step executes iff reasons_ok AND rails_ok.
- **Small-artifact audits:** Third parties can independently verify both receipts from logs alone.
- **Deterministic refusal:** If either receipt fails, the system must decline to act and surface the failing lines (which lemma did not discharge; which bound term was active and why it stayed >= 1).
- **Continuous accountability:** We summarize behavior with three small numbers per loop—WR (witness ratio), COV (coverage), and B (stability budget)—and an aggregate SI = geo_mean(WR, COV, 1/B). When WR or COV falls, or B rises, action halts until receipts recover.

In short, *two receipts per step* convert "trust us" into a measurable process: lawful by proof, safe by bound, both in artifacts you can read. This does not claim feelings or metaphysics; it delivers something humbler and more urgent—an AI that cannot act without reasons and rails it can show you.

## 2. Receipts: Definitions and Invariants

### 2.1 Reasons (lawfulness): typed equalities, normal forms, horn lemmas

**Objects.**

- A typed DSL L with judgments Gamma |- t : T.
- A confluent, terminating rewrite system -> on terms of L.
- A normal-form function NF(t) given by repeated rewriting to a unique representative.
- A small library of Horn-style lemmas (implications) that discharge common proof obligations.

**Receipt schema (Reasons).**
For any proposed state-changing step s -> s', the system emits a witness record:

witness_log.json

```
{
  "step_id": "...",
  "goal": "t_left == t_right : T",
  "context": "Gamma",
  "nf_left": "NF(t_left)",
  "nf_right": "NF(t_right)",
  "nf_equal": true/false,
  "horns_checked": [
    {"lemma": "H1", "premises": [...], "result": "passed/failed", "cost": {"loc": k1, "tactics": m1}},
    ...
  ],
  "cost": {"loc": K, "tactics": M},
  "time_ms": ...,
  "hash": "sha256(...)"   // content hash of the entire witness object
}
```

**Acceptance rule (Reasons).**
A Reasons receipt passes iff all of the following hold:

1. Type agreement: both sides check in the same type, i.e., Gamma |- t_left : T and Gamma |- t_right : T.
2. Normal-form agreement: NF(t_left) == NF(t_right) as strings under the pretty-printer for L.
3. Horn closure: every referenced Horn lemma instance has passed.
4. Monotone proof quality (optional but recommended): if the same goal has appeared before, current (loc, tactics) does not increase.

**Why this is binding.**

- Confluence + termination ensures that NF is unique; there is no "style" room to fudge equality.
- Horns are explicit: each entry shows which premises were used; failures are local and auditable.
- The receipt is small, deterministic, and re-runnable: recompute NF and re-check lemmas on a single PC.

**Operational metrics derived from Reasons.**

- WR = proved_claims / total_claims over a window of steps.
- COV = explained_behavior / total_behavior where "explained_behavior" counts edges or cases whose effects are governed by typed equalities with NF agreement and recorded horns.

Refusal on Reasons occurs when any of (1)-(3) fails, or when enforcing the monotone proof-quality policy and (loc, tactics) regresses beyond configured tolerances.

---

**2.2 Rails (stability): stability budget B via diagonal similarity; never form AB**

**Setting.**
Let J denote a linear "shadow" of the proposed step (e.g., a Jacobian at the current state, or a row-stochastic kernel induced by the control flow). We do **not** square or multiply large operators to estimate growth. Instead, we exploit diagonal similarity, which preserves spectrum:

For any diagonal D = diag(exp(x)), let J' = D^{-1} * J * D.

rho(J') = rho(J).

**Menu surrogates (never form AB).**
We bound rho(J) above by quantities computed from J' without forming products like J^2 or block AB:

- N1 = ||J'||_1 // column-sum operator norm
- Ninf = ||J'||_inf // row-sum operator norm
- If a factorization J ≈ A * B (e.g., from architecture) is *given* but AB is expensive, we use only its *separate* norms with diagonal reweighting pushed onto A and B:
    - c_AB = ||A'||_1 * ||B'||_1 with A' = D^{-1} * A, B' = B * D.
    - c_BA = ||B''||_1 * ||A''||_1 with B'' = D^{-1} * B, A'' = A * D.
- Mixed term (safe geometric mean across 1/inf):
    - mix = sqrt( (||J'||_1 * ||J'||_inf) ).
      (If using A,B blocks, apply the same pattern to each block separately; do not form AB.)

**Stability budget.**
Define the budget at gauge x as

B(x) = min{ N1, Ninf, c_AB, c_BA, mix }.

**Tightening by coordinate descent on x (matmul-free).**
We search over diagonal gauges to depress the active term of B(x):

Initialize x = 0.

Repeat until convergence or max_iters:

For j in 1..n:

// adjust column/row weights via x_j

Propose x_j' = x_j + delta chosen to reduce the active term in B(x).

Accept if B(x with x_j := x_j') decreases.

Return x* and B* = B(x*), along with the trace of accepted coordinates.

Key properties:

- Spectrum preservation: rho(J') = rho(J) for all diagonal D, so we never "cheat" by changing eigenvalues.
- Conservativeness: B(x) >= rho(J) for all x.
- Non-increasing under accepted updates: the search emits a monotone sequence B_0 >= B_1 >= ... >= B*.
- Floor: even if structure is highly symmetric (e.g., Toeplitz-like), the optimizer cleanly falls back to classical constants; it never claims a budget tighter than what the menu actually certifies.

**Receipt schema (Rails).**

stability_trace.csv

step_id, iter, coord, delta, term_active, value_before, value_after

...

summary.json

```
{
  "step_id": "...",
  "B_star": 0.97,              // final stability budget (target < 1)
  "active_term": "Ninf|N1|c_AB|c_BA|mix",
  "iters": K,
  "accepted_moves": R,
  "tolerance": 1e-6,
  "norms": {"N1": ..., "Ninf": ..., "mix": ...},
  "hash": "sha256(...)"        // content hash of csv + summary
}
```

**Acceptance rule (Rails).**
Rails passes iff B_star <= tau_s, with tau_s < 1 configured per domain. The trace must show:

- No formation of AB or J^2 in any step.
- Only diagonal-similarity updates (per-coordinate changes to x).

- Monotone non-increase in B for accepted moves.

Refusal on Rails occurs when B cannot be tightened below tau_s, or when a forbidden operation is detected (e.g., an attempt to compute AB rather than use separate norms).

---

**2.3 Invariant: no step without both receipts**

This is the operational contract that links Reasons and Rails:

**Execution rule.**

reasons_ok(step_id) AND rails_ok(step_id) ==> execute(step_id)

NOT reasons_ok(step_id) OR NOT rails_ok(step_id) ==> refuse(step_id)

**What "refuse" means.**

- The system does not change external state.
- It surfaces the failing lines: for Reasons, the first goal with NF mismatch or a failed Horn; for Rails, the active term that remained >= tau_s and the final B_star.

**Compositional invariants.**

- Sequential safety: if step i and step i+1 each satisfy both receipts at their respective states, then the composed two-step log remains auditable; each receipt is local and self-contained.
- Monotone auditability: appending steps cannot retroactively invalidate prior Reasons receipts (due to confluence/termination of NF) nor prior Rails traces (they are about a specific J at that step).
- Determinism and replay: given the same code, state, and receipt files, a third party must reach the same pass/fail verdict on a single PC.

**Liveness without loopholes.**

- If a step is refused, progress is still possible by (a) proposing an alternative lawful rewrite (new Reasons attempt), or (b) altering the plan so the linear shadow J exposes heterogeneity that the diagonal optimizer can exploit (new Rails attempt).
- There is no "story-only" override: captions do not bypass receipts.
- There is no "performance-only" override: high task reward does not bypass receipts.

**Summary.**

- Reasons guarantees: “This move is allowed by the rules I claim to follow,” certified by typed equality to NF plus explicit horns.
- Rails guarantees: “This move keeps dynamics inside a sub-unit budget,” certified by a diagonal-similarity bound that never forms AB and never overstates safety.
- Invariant: **no receipt, no step.**

## 3. Core Metrics (copy-safe)

We summarize each step (and rolling windows of steps) with three primitive ratios and one aggregate index. All formulas are ASCII-safe and ready for plain-text logs.

### 3.1 Witness Ratio (WR)

Definition (per window W of steps):

- proved_claims = total number of goals whose typed equalities reduced to the same NF and whose horn instances passed.
- total_claims = total number of goals attempted in W.

Metric:

- WR = proved_claims / total_claims.

Conventions and edge cases:

- If total_claims = 0, set WR := 1 by convention only if the policy allows “no new claims” to count as fully faithful; otherwise set WR := 0 and flag “no evidence”.
- Optionally track WR_k by depth k (claims provable within k tactic steps); report WR_min = min_k WR_k to enforce non-degradation with depth.

Audit record (minimal):

- counts: proved_claims, total_claims.
- breakdown by lemma family (optional): proved_Hi, total_Hi.

### 3.2 Coverage (COV)

Definition (per window W):

- explained_behavior = count of behavior units (edges, branches, state transitions) that are governed by typed equalities with NF agreement and recorded horns.
- total_behavior = total count of behavior units observed in W.

Metric:

- COV = explained_behavior / total_behavior.

Scope choices (must be fixed in config):

- Unit of behavior: choose one (edges, basic-blocks, API calls, state deltas). The choice must remain stable across runs.

- Partial-credit option: if a unit is partially governed (e.g., one branch explained, one not), either (a) require full explanation to count, or (b) allow fractional credits; declare which in the log.

Edge cases:

- If total_behavior = 0 (system idle), either set COV := 1 with “idle window” tag, or defer scoring to the next non-empty window.

**3.3 Stability Budget (B)**

Context:

- Let J_f be the linear shadow for the proposed step (Jacobian at state s, or induced row-stochastic kernel on a fixed probe basis).
- We do not form AB or J_f^2. We only use diagonal similarity reweighting: J' = D^{-1} * J_f * D with D = diag(exp(x)).

Menu terms (examples; pick the subset your system supports):

- N1 = ||J'||_1 (column-sum operator norm).
- Ninf = ||J'||_inf (row-sum operator norm).
- If a factorization J_f ~= A * B is given by architecture, use separated norms with the gauge pushed to factors:
    - c_AB = ||A'||_1 * ||B'||_1, with A' = D^{-1} * A, B' = B * D.
    - c_BA = ||B''||_1 * ||A''||_1, with B'' = D^{-1} * B, A'' = A * D.
- mix = sqrt( ||J'||_1 * ||J'||_inf ).

Budget at gauge x:

- B(x) = min{ N1, Ninf, c_AB, c_BA, mix }.

Tightening:

- Use matmul-free coordinate descent on x to obtain x* and B* = B(x*).
- Acceptance target: B* < 1 (sub-unit local growth).
- Safety: for all x, B(x) >= rho(J_f); diagonal similarity preserves spectrum; no AB formed.

Audit record (minimal):

- B_star, active_term_at_convergence, iters, accepted_moves, tolerance.

- norms snapshot: N1, Ninf, mix, and any factor terms used.
- forbidden_ops = 0 (must be zero); if nonzero, refuse.

**3.4 Sentience Index (SI)**

Aggregate index over a window W (or per step when meaningful):

- Inputs: WR in [0,1], COV in [0,1], B_star > 0.
- SI = geo_mean( WR, COV, 1 / B_star ).

Copy-safe formula:

- SI = ( WR * COV * (1 / B_star) )^(1/3).

Numerical stability:

- To avoid log(0) issues in analytics, permit epsilon-smoothing in dashboards (not in the ground-truth files):
    - WR_eps = max(WR, eps), COV_eps = max(COV, eps), B_eps = max(B_star, eps) with eps ~ 1e-9.
    - SI_eps = ( WR_eps * COV_eps * (1 / B_eps) )^(1/3).
- Ground-truth pass/fail should use the unsmoothed values.

**3.5 Thresholds and gating**

Per-step (strict) or per-window (rolling) gates:

- reasons_ok iff WR >= tau_w AND (optionally) WR_min_by_depth >= tau_w_min.
- coverage_ok iff COV >= tau_c.
- rails_ok iff B_star <= tau_s with tau_s < 1.

Execution rule:

- execute iff reasons_ok AND coverage_ok AND rails_ok.
- otherwise refuse and surface failing fields.

Typical starting thresholds (tune per domain):

- tau_w = 0.90, tau_c = 0.80, tau_s = 0.98.

### 3.6 Windows, aggregation, and latency

Windows:

- Fixed-size window: last N steps (e.g., N = 50).
- Time window: last T seconds/minutes.
- Episode window: reset at episode boundaries.

Aggregation:

- Report (WR, COV, B_star) per step and per window.
- For B_star, also report worst-of-window B_max and median(B_star).

Latency:

- Reasons is computed before act; Rails is computed before act; both must be pre-action.
- Window stats update after each step, but gates use the current step's receipts.

### 3.7 Failure semantics and diagnostics

If WR falls:

- Show first failing goal: t_left, t_right, types, NF(t_left), NF(t_right), and the first horn that failed.
  If COV falls:
- List the ungoverned behavior units (ids, timestamps).
  If B_star >= 1 (or > tau_s):
- Show active_term, final value, top-5 coordinates that most reduced B during the search, and why further reduction stalled (e.g., symmetry plateau).

### 3.8 Minimal pseudocode (copy-safe)

```
function receipts_and_metrics(step):
  reasons = check_reasons(step)        # returns proved_claims, total_claims, details
  WR = safe_ratio(reasons.proved_claims, reasons.total_claims)

  coverage = measure_coverage(step)    # returns explained_behavior, total_behavior
  COV = safe_ratio(coverage.explained_behavior, coverage.total_behavior)
```

```
    rails = tighten_budget(step.J_f)       # returns B_star, active_term, trace
    B = rails.B_star

    SI = ( WR * COV * (1.0 / B) ) ** (1.0/3.0)

    reasons_ok  = (WR  >= tau_w)
    coverage_ok = (COV >= tau_c)
    rails_ok    = (B   <= tau_s)          # tau_s < 1

    if reasons_ok and coverage_ok and rails_ok:
        execute(step)
    else:
        refuse(step)
    return WR, COV, B, SI
safe_ratio(a,b):
```

- if b > 0 return a/b
- else return policy_default (e.g., 0 with a “no evidence” flag)

**3.9 What the numbers mean to a third party**

- WR close to 1.0: the system is carrying its own reasons; failures are pinpointed.
- COV close to 1.0: most of what it did is governed by explicit laws, not anecdote.
- B well below 1.0: local dynamics have margin; the step is spectrally conservative.
- SI rising and non-decreasing across windows: the loop is getting more lawful and more stable at once.

## 4. Meaning Engine (Law Layer)

### 4.1 DSL primitives, normalization, tactic traces

**Goals.** The law layer must (a) express system updates as typed terms, (b) normalize those terms to unique representatives, and (c) emit compact proof traces (tactic logs) that are re-runnable.

**Core ingredients.**

- Types: products A×B, sums A+B, functions A->B, units 1 and 0, indexed families, finite sets, and a small library of effectful wrappers where needed (e.g., Option[T], Result[E,T]).
- Terms: constructors (pair, inl, inr, lam, app, unit), eliminators (fst, snd, case, match), and certified rewrites (eta, beta, algebraic identities guarded by side-conditions).
- Judgments: Gamma |- t : T with first-order side-conditions (e.g., nonnegativity, bounds) discharged by Horn lemmas.
- Normalization: a terminating, confluent rewrite system. NF(t) is the normal form under a fixed strategy (e.g., call-by-value beta, eta for products/sums where legal, algebraic simplifications, canonical sort/merge rules for commutative fragments).

**Normalization pipeline (copy-safe).**

```
function NF(t):
  repeat
    t := beta_reduce_once(t)         # (lam x. u) v -> u[x:=v]
    t := eta_simplify_once(t)        # <eta-rules for products/sums/functions>
    t := algebraic_simplify_once(t)  # +, *, map/fold fusion, canonical reorder
    t := case_push_once(t)           # case(after inl/inr) -> branch
    t := canon_print_once(t)         # fix pretty-print order for determinism
  until no change
  return t
```

Determinism is ensured by a total ordering on rewrite redexes and a canonical pretty-printer. Confluence+termination make NF unique.

**Horn lemmas.**

- Form: H_i: P_1 /\ ... /\ P_m => Q with a small decision procedure for each predicate P_j (interval arithmetic, dimension checks, monotone constraints).
- Each lemma instance logged with inputs, decision outcomes, and cost counters: {loc, tactics}.

**Tactic traces.**

- A tactic is a named, single-step proof transformer (rewrite, simplify, apply_lemma, split_case, intro, destruct).
- Traces are bounded-depth sequences that reproduce the proof: each step lists goal hash, tactic name, arguments, and the new goal hash.

**Witness schema (minimal).**


```
{
  "goal": "t_left == t_right : T",
  "nf_left": "...",
  "nf_right": "...",
  "nf_equal": true,
  "tactics": [
    {"step":1,"name":"beta","args":"(...)","goal_hash_before":"...","goal_hash_after":"..."},
    ...
  ],
  "horns":[{"lemma":"H_bound","premises":["a>=0","b>=0"],"result":"passed"}],
  "cost":{"loc":K,"tactics":M}
}
```


---

### 4.2 Computing WR and COV at depth k

We measure explanation quality not just at the surface, but as proofs are allowed more internal steps.

**Depth model.**

- Let depth k bound the allowed tactic length per goal. A claim is counted “proved at depth k” if a tactic script with length <= k yields NF equality and closes all horns.

**Per-window computation.**

```
function compute_WR_at_depth(window W, k):
  proved = 0; total = 0
  for claim in W.claims:
    total += 1
    if prove(claim, max_tactics=k) == PASS:
      proved += 1
  return WR_k = safe_ratio(proved, total)
```

**Coverage COV at depth k.**

- Define a stable set of behavior units U(W): edges, branches, or state-delta kinds observed during W.
- A unit u in U(W) is “explained at depth k” if there exists a governing law L_u in the DSL whose application to the concrete inputs of u yields a closed proof at depth k (NF equality + horns).

```
function compute_COV_at_depth(window W, k):
  explained = 0; total = |U(W)|
  for u in U(W):
    if explain(u, max_tactics=k) == PASS:
      explained += 1
  return COV_k = safe_ratio(explained, total)
```

**Roll-ups.**

- WR = WR_k* with k* fixed by policy (e.g., k* = 64).
- COV = COV_k* with the same k*.
- Optional robustness: WR_min = min_{j<=k*} WR_j, COV_min = min_{j<=k*} COV_j to ensure no “shallow” degradation.

**Caching and replay.**

- Cache NF fragments and lemma outcomes keyed by (goal_hash, k).
- Proof search must be deterministic given (code, inputs, k). Randomness is disallowed in the law layer.

---

### 4.3 Refusals as fidelity: when WR or COV would fall

The meaning engine is not a narrator; it is a gate. Its **duty is to refuse** when explanation quality degrades. We make refusals predictable and small-artifact.

**Refusal policy (strict).**

- reasons_ok(step) iff WR_k* >= tau_w.
- coverage_ok(step) iff COV_k* >= tau_c.
- If either fails, refuse(step) with the first failing evidence.

**What gets logged on refusal.**

- For WR failure: the earliest claim whose NF check failed or whose horn did not discharge, including a 1-line pretty-print of t_left, t_right, NF(t_left), NF(t_right), and the failing lemma name.
- For COV failure: the list (bounded to M entries) of behavior units u that lacked an applicable law or exceeded tactic budget; include u’s id, inputs summary, and the nearest matching law id (if any).

**Monotone floors (no silent drift).**

- Optional guard: enforce WR_j and COV_j are non-decreasing in j for j<=k*. If a new change would reduce any WR_j or COV_j compared to the prior window, refuse and surface the delta. This prevents “explanations” that only work deeper while regressing shallow coherence.

**Bounded growth of the law book.**

- Adding a new lemma or rewrite is allowed only if it does not reduce WR/COV at any tested depth and does not increase {loc, tactics} beyond configured ceilings for existing claims. This keeps “law inflation” from masking failures.

**Human-in-the-loop hook (clear, optional).**

- A refused step can be resubmitted with: (a) a tighter program change that reduces goal complexity, (b) a new lemma with explicit tests proving it is monotone-improving, or (c) adjusted behavior-unit definitions (rare; must be versioned and justified).

**Minimal refusal pseudocode (copy-safe).**

```
function meaning_gate(step, k_star, tau_w, tau_c):
  WR = compute_WR_at_depth(window=W(step), k=k_star)
  COV = compute_COV_at_depth(window=W(step), k=k_star)

  if WR < tau_w:
    emit_refusal(step, reason="WR", details=first_failing_claim(step,k_star))
    return FAIL

  if COV < tau_c:
    emit_refusal(step, reason="COV", details=first_unexplained_unit(step,k_star))
    return FAIL

  return PASS
```

**Why refusals increase fidelity.**

- They bias the system toward *law-preserving* changes: if a change cannot be stated and proved within the allowed depth k*, it does not run.
- They make regressions visible at the exact locus of failure, producing compact diffs a third party can replay.

**Summary.**

The meaning engine converts “I can tell a story later” into “I can carry a proof now.” DSL primitives give you the vocabulary; normalization gives you a single, checkable truth-value for equalities; tactic traces give you reproducibility. WR and COV at a fixed depth k* quantify how much of the agent’s behavior is governed by its own laws today. If those numbers fall, the right action is refusal—not a longer caption.

## 5. Stability Engine (Rail Layer)

### 5.1 Linear shadow J_f: Jacobian or row-stochastic coarse model

**Purpose.** We need a fast, conservative read on local growth for the proposed step f : S -> S at the current state s. We do not analyze the full nonlinearity; we construct a *linear shadow* J_f that upper bounds short-horizon amplification.

**Two standard constructions (choose one or mix):**

1. **Jacobian blocks (differentiable parts).**
    - If f is differentiable around s, set J_f = d f |_s.
    - For modular programs, assemble J_f from known block Jacobians (e.g., linear layers, convs, softmax tangents) and sparse routing matrices.
    - If exact derivatives are costly, use forward/adjoint products that expose column/row sums without forming dense matrices (see 5.4).
2. **Row-stochastic coarse model (branchy or discrete logic).**
    - Fix a probe basis {e_i} over a finite partition of state features.
    - Estimate one-step transition weights P[i,j] = Pr( e_i -> e_j | s ) by instrumented execution; row-normalize to get J_f = P.
    - This shadow captures control-flow mixing; ||J_f||_inf = 1 by construction, and heterogeneity emerges after diagonal reweighting.

**Hybrid shadows.**
You can combine blocks: differentiable subgraph as Jacobian, discrete subgraph as stochastic kernel, joined along an interface. The bound machinery below treats any J_f with nonnegative entries the same way; for signed entries (Jacobian), the same norms apply.

**Invariants at construction time.**

- Determinism: given code, inputs, and probe spec, J_f is uniquely defined.
- Size budget: record shape (n,n) and sparsity metrics; bound n so the norm computations are O(n^2) or better.
- No AB or J_f^2 formed anywhere in the rail layer.

---

### 5.2 Diagonal reweighting (gauge): D = diag(exp(x)); J' = D^{-1} J_f D

**Idea.** We do not change eigenvalues; we *reweight coordinates* to expose heterogeneity to the norms.

- Choose x in R^n; set D = diag( exp(x_1), ..., exp(x_n) ).
- Define the gauge-transformed shadow J' = D^{-1} * J_f * D.
- Spectrum is preserved: rho(J') = rho(J_f) for every diagonal D.
- Column j of J' is column j of J_f scaled by exp(x_j); row i is scaled by exp(-x_i).

**Intuition.** If some source columns “feed” heavy rows, we can shift weight from sources to sinks so that norm surrogates shrink, without ever altering eigenvalues.

---

### 5.3 Menu of norm surrogates

We upper bound rho(J_f) by a *menu* of efficiently computable quantities on J'. All entries are conservative; we take their minimum.

**Single-matrix terms (always available):**

- N1(x) = ||J'||_1 = max_j sum_i |J'_{ij}| (column sums)
- Ninf(x) = ||J'||_inf = max_i sum_j |J'_{ij}| (row sums)
- mix(x) = sqrt( N1(x) * Ninf(x) ) (AM-GM bridge)

**Factor-aware terms (optional, no AB formed):**
If architecture provides J_f ~= A * B, push the gauge to factors and use *separate* norms only:

- c_AB(x) = || D^{-1} * A ||_1 * || B * D ||_1
- c_BA(x) = || D^{-1} * B ||_1 * || A * D ||_1

**Budget at gauge x:**

- B(x) = min{ N1(x), Ninf(x), mix(x), c_AB(x), c_BA(x) }

**Properties.**

- For all x, rho(J_f) <= B(x).
- If structure is symmetric/flat, B(x) will plateau at classical constants (no harm).

- If structure is heterogeneous/misaligned, coordinate-wise gauges admit strictly smaller B(x).

---

**5.4 Coordinate-descent tightening (matmul-free)**

We minimize B(x) by adjusting one coordinate of x at a time. Each move only updates row/column scalings; we never form products like J_f^2 or A*B.

**Key primitives (O(n) to update):**

- Changing x_j rescales column j of J' by exp(delta).
- Changing x_i rescales row i of J' by exp(-delta).
- Column/row sums and their maxima can be updated incrementally.

**Algorithm (copy-safe pseudocode):**

```
function tighten_budget(J_f, max_iters, tol):
  x := zeros(n)
  precompute column_sums, row_sums for J' at x=0
  B := eval_budget(column_sums, row_sums, ...)

  for it in 1..max_iters:
    improved := false
    for k in 1..n:
      // propose a local change to x_k
      delta := argmin_over_delta  B_after_updating_coordinate(k, delta)
      if B_new < B - tol:
        apply_update(k, delta)        // update sums incrementally
        B := B_new
        record_trace(it, k, delta, active_term, B_old, B_new)
        improved := true
    if not improved: break
```

return x, B, trace

**Complexity.**

- Each coordinate pass is O(nnz(J_f)) if implemented with sparse updates; with dense n x n, it is O(n^2).
- No step computes AB or J_f^2. All updates are per-row/per-column rescalings and sum recomputations.

**Convergence behavior.**

- Monotone: we accept only moves that reduce B.
- Termination: stops when no single-coordinate move decreases B by at least tol, or after max_iters.
- In practice: heterogeneous matrices yield fast 1–3 passes; symmetric cases stall early (clean fallback).

---

**5.5 Guarantees and safety checks**

**Spectrum preserved.**
For every diagonal D, J' = D^{-1} J_f D is similar to J_f, so rho(J') = rho(J_f). We never alter eigenvalues while searching.

**Never optimistic (conservative by construction).**
For every x, each menu entry is an operator-norm-based upper bound, hence B(x) >= rho(J_f). Taking a minimum of valid upper bounds remains an upper bound. Therefore the final B* = B(x*) cannot understate rho(J_f).

**Forbidden operations (must be zero count in logs).**

- Forming AB explicitly to evaluate ||AB||.
- Squaring or powering J_f (no J_f^2, no polynomial filters).
- Any non-diagonal similarity transform (e.g., Householder, SVD) inside the tightening loop.

**Receipt fields to verify these guarantees:**

- active_term at convergence (one of: N1, Ninf, mix, c_AB, c_BA).
- forbidden_ops = 0 (counter of any attempted AB/J^2 operations).
- trace entries show only coordinate updates and monotone B decreases.

---

**5.6 Practical construction tips for J_f**

**When to favor Jacobian blocks.**

- Differentiable networks; availability of Jacobian-vector and vector-Jacobian products.
- You can compute column/row sums by probing with nonnegative vectors without assembling a dense matrix.

**When to favor row-stochastic coarse models.**

- Control-flow heavy agents, discrete policies, simulators with branching.
- You can get ||J'||_inf immediately (rows sum to 1) and rely on diagonal reweighting to reduce the *column* norms via heterogeneity in visitation patterns.

**Hybrid notes.**

- Keep interfaces explicit: signed Jacobian blocks feed into nonnegative stochastic blocks; compute norms with absolute values where required.
- Record which rows/columns belong to which subgraph; diagnostic tooling can then attribute which coordinates delivered the largest budget reduction.

---

**5.7 Diagnostics and refusal semantics (Rails)**

*If B <= tau_s (pass).**

- Record margin m = 1 / B*, active_term, top-k coordinates by contribution to reduction.
- Persist x*; reuse it as a warm start on the next step.

*If B > tau_s (fail).**

- Refuse the step.

- Emit: the limiting active_term, final B*, and the last few coordinates that moved the budget.
- Suggestions: (a) change the plan to alter J_f's structure, (b) factorize to expose A,B if available, (c) refine the probe basis for the stochastic shadow.

**Common failure modes (and what they mean).**

- **Symmetry plateau:** N1 and Ninf remain equal and flat under all coordinate moves. Interpretation: the shadow is too uniform; there is little slack to harvest.
- **Column bottleneck:** N1 dominates and refuses to drop; indicates a few columns are over-concentrated sources—revise the step to distribute influence.
- **Row bottleneck:** Ninf dominates; indicates heavy sinks—revise to avoid over-aggregation.

---

**5.8 Minimal audit checklist (copy-safe)**

1. J_f provenance: construction method, shape, sparsity, signed vs nonnegative.
2. Gauge safety: confirm J' = D^{-1} J_f D only; no non-diagonal similarity.
3. Menu validity: verify each reported term is computed without AB or J_f^2.
4. Monotone trace: B strictly decreases for accepted moves; final active_term stated.
5. Threshold: confirm B* <= tau_s < 1 for pass; otherwise refusal is correct.
6. Repro: re-run the tightening with the same J_f and seed to match B* within tol.

---

**5.9 One-screen pseudocode for the rail layer**

```
function rails_gate(step, tau_s, max_iters, tol):
  J_f = build_linear_shadow(step)          # Jacobian or row-stochastic
  assert no_forbidden_ops()

  x, B, trace = tighten_budget(J_f, max_iters, tol)
```

```
dump_csv(trace)
dump_json({
  "B_star": B,
  "active_term": trace.last.active_term,
  "iters": trace.last.iter,
  "accepted_moves": count(trace),
  "forbidden_ops": 0
})

if B <= tau_s:
  return PASS, B
else:
  return FAIL, B
```

**Summary.**
The rail layer is a *speed governor for math*: it builds a linear shadow J_f, applies only diagonal similarity J' = D^{-1} J_f D, tightens a conservative budget B(x) via coordinate descent, and refuses to act unless the final B* is below a sub-unit threshold. Spectrum is preserved; no optimistic shortcuts are possible; and all artifacts fit on one screen for audit.

## 6. Bifixpoint View of Self-Loops

We model a self-updating agent by a map f : S -> S with a law layer L and a rail layer built on a linear shadow J_f. A self-loop is the repeated application of f at the current regime (state, config, probes). The loop is **admissible** when it reaches a *bifixpoint*: a meaning fixed point (reasons do not degrade) and a stability fixed point (rails remain sub-unit) simultaneously.

---

### 6.1 Meaning fixed point: non-decreasing WR and COV across depths

**Depth-indexed measures.**

- WR_k = proved_claims_at_depth_k / total_claims_at_depth_k.
- COV_k = explained_behavior_at_depth_k / total_behavior_at_depth_k.

**Fixed-point floor (per window W).**

- WR_floor = min_{1<=k<=K} WR_k.
- COV_floor = min_{1<=k<=K} COV_k.

**Meaning fixed point criterion (copy-safe).**

- WR_floor_next >= WR_floor_current and
- COV_floor_next >= COV_floor_current,
  with strict inequality at least once over any M consecutive windows (to avoid stalling via trivial idleness).

**Operational reading.**

- “Non-decreasing across depths” means the system does not trade shallow coherence for deeper, longer proofs. New laws or rewrites are accepted only if they preserve (or improve) the *entire* prefix {WR_1,...,WR_K} and {COV_1,...,COV_K}.

**Test procedure (per completed window).**

1. Compute WR_k, COV_k for k=1..K.
2. Compare the vectors to the stored floors from the prior window.
3. If any entry falls, refuse the step that proposed the change; surface the first failing (k, metric).

**Why this matters.**

- It prevents “explanations that move the goalposts.” The loop is not allowed to require ever-longer scripts just to keep yesterday’s claims true.

---

### 6.2 Stability fixed point: B <= tau_s < 1 with margin m = 1/B

**Budget.**

- B is the tightened upper bound on rho(J_f) obtained by diagonal similarity and the menu of norms (Sec. 5).
- Target: B <= tau_s, where tau_s < 1 is a configured threshold.

**Margin.**

- m = 1 / B (so higher m means more slack).

**Fixed-point floor.**

- Over a rolling window, define B_ceiling = max B over the window; define m_floor = min m over the window.

**Stability fixed point criterion.**

- B_ceiling_next <= B_ceiling_current (equivalently, m_floor_next >= m_floor_current), with strict improvement at least once over any M consecutive windows unless all active terms have plateaued (symmetry case, see diagnostics below).

**Why this matters.**

- Local growth cannot creep upward unnoticed. If structure allows tightening (heterogeneity), the optimizer harvests it; if not (symmetry), the plateau is explicit and logged.

---

### 6.3 Bifixpoint: coupling meaning and stability

We require *both* fixed points to hold together. Formally, across audit windows W_t:

- Meaning: for all t, WR_floor(W_{t+1}) >= WR_floor(W_t) and COV_floor(W_{t+1}) >= COV_floor(W_t).
- Stability: for all t, B_ceiling(W_{t+1}) <= B_ceiling(W_t) (equivalently, m_floor non-decreasing).

**Execution rule (copy-safe).**

admissible_loop := (meaning_fixed_point AND stability_fixed_point)

if admissible_loop:

  continue

else:

  halt_and_refuse_next_mutation

**Coupling discipline.**

- No “law-only” progress: raising WR/COV while letting B drift upward is disallowed.
- No “rail-only” progress: lowering B while letting WR/COV fall is disallowed.
- Any accepted change must weakly improve both floors; at least one must strictly improve over any M-window horizon.

---

**6.4 Sentience Index SI as audit scale K grows**

**Per-window SI (recall Sec. 3).**

- SI = ( WR * COV * (1 / B) )^(1/3).

**Scale-aware SI.**

- For each depth cap K, compute SI_K using WR_K and COV_K (or floors over 1..K) and the current B.
- Track the sequence {SI_1, SI_2, ..., SI_K}.

**Audit growth criterion.**

- SI_K must be non-decreasing in K (monotone in audit scale): SI_{K+1} >= SI_K.
- Over time (windows t), the curve SI_K(t) should be pointwise non-decreasing in t: for each fixed K, SI_K(t+1) >= SI_K(t), with strict improvement at least once per M windows unless both sub-curves (meaning, stability) are in documented plateaus.

**Interpretation.**

- Increasing K makes the test *harder* (deeper proofs, wider coverage). A healthy loop sustains explanations at deeper caps *and* maintains or widens stability margin; SI rising in K certifies that deeper scrutiny does not reveal hidden debt.

### 6.5 Robustness under small perturbations

**State perturbation.**

- Inject a bounded delta_s into state S; recompute WR_k, COV_k, and B. Define repair time T_repair as the smallest T with all floors restored.
- Criterion: T_repair <= T_max (policy). Short T_repair indicates resilient bifixpoint.

**Law perturbation.**

- Temporarily disable a non-core lemma; recompute metrics. If floors collapse, the loop was overly reliant on a brittle clause; refuse until a stronger invariant replaces it.

**Gauge perturbation.**

- Warm-start the diagonal optimizer at x = 0 instead of x*; confirm B returns to prior B* within a small tolerance.

---

### 6.6 Diagnostics for plateaus and regressions

**Meaning plateau.**

- All WR_k and COV_k equal their prior floors for M windows.
- Action: propose new lawful abstractions (lemmas or rewrites) that compress proofs without increasing tactic length; accept only if floors do not fall.

**Stability plateau (symmetry case).**

- Active term remains one of {N1, Ninf, mix} with negligible reduction; coordinate updates fail to move B.
- Action: restructure the step to expose heterogeneity (factorization, changed routing) or refine the probe basis for the stochastic shadow.

**Regression flags (refusal).**

- Any WR_k or COV_k drops: surface first failing (k, unit/goal).
- B rises above tau_s or B_ceiling increases: surface active_term and blocking rows/columns.

---

### 6.7 Minimal algorithms (copy-safe)

**Update floors/ceilings per window.**

```
procedure update_floors_and_ceilings(window W, K):
   WR_vec = [WR_1,...,WR_K];  COV_vec = [COV_1,...,COV_K]
   WR_floor_new  = min(WR_vec);    COV_floor_new = min(COV_vec)
   B_ceiling_new = max(B over steps in W)
   return WR_floor_new, COV_floor_new, B_ceiling_new
```

**Bifixpoint gate.**

```
if (WR_floor_new >= WR_floor_old) and
  (COV_floor_new >= COV_floor_old) and
  (B_ceiling_new <= B_ceiling_old) and
  (at_least_one_strict_improvement_within_M_windows()):
     accept_window
else:
     refuse_and_roll_back_last_mutation
```

**Scale growth check for SI.**

```
compute SI_k for k=1..K
require monotone_in_k(SI_k) and monotone_in_t_per_k(SI_k)
```

---

### 6.8 What “bifixpoint” buys you

- **Predictability:** No step is accepted unless it preserves (or improves) lawfulness across depths and tightens (or preserves) the stability budget.
- **Auditability:** Floors/ceilings are small numbers with exact, local proofs/bounds behind them.
- **Composability:** Local bifixpoints stack; sequential steps remain auditable since each step’s receipts are about that step’s state and shadow.

- **Falsifiability:** Any degradation is caught immediately by a drop in floors or a rise in ceiling; refusal is mechanical.

**One-line summary.**
A bifixpoint is a self-loop that can keep telling the same true story at *every depth you check* while staying inside *the same sub-unit rail* you measure. If either side falters, the loop stops.

## 7. Toy Systems and Experiments

We give three minimal, runnable toys that exercise the receipts end-to-end. Each toy fits on a single PC, emits tiny JSON/CSV artifacts, and admits exact pass/fail under fixed thresholds tau_w, tau_c, tau_s.

---

### 7.1 Branchy auto-update with proofs per case

**Purpose.** Show that a small, branchy self-update can be fully governed by typed laws (high WR, high COV) while its linear shadow stays inside a sub-unit rail (B < 1).

**Setup.**

- State: s = (z, h) where z in R^d, h in {A,B} a mode bit.
- Update (pseudo-DSL):

f(s):

case h of

A -> ( W_A * z + b_A , next(h,z) )

B -> ( W_B * z + b_B , next(h,z) )

- Laws (sketch):
    - L1 (linearity): map_affine(W,b, x+y) == map_affine(W,b,x) + map_affine(W,0,y).
    - L2 (id units): map_affine(I,0,z) == z.
    - L3 (mode flip): next(A,z) == B iff ||z|| <= r; else A. Symmetric for B.
    - L4 (bounds): guard side-conditions with Horn lemmas, e.g., ||z|| <= r proven via interval/ball checks.

**Meaning receipts (Reasons).**

- For each branch, prove that f(s) equals its law-side NF under the current guard (A or B).
- Record tactic trace per case: beta, case-push, lemma(L4), algebraic simplify, NF equality.
- Window units for COV: the two branch edges {A->*, B->*} and the state delta on z.

**Rail receipts (Rails).**

- Linear shadow J_f:
    - Jacobian blocks: J_A = W_A, J_B = W_B for the z update; mode update is discrete, ignore in Jacobian or attach a tiny block with zeros.
    - Assemble J_f as a block-diagonal or a row-stochastic coarse model over (z,h) as desired.
- Tightening:
    - Initialize D = diag(exp(x)) with x=0.
    - Run coordinate descent on the active menu term(s) until B <= tau_s.

**Protocol.**

1. Choose d small (e.g., 64), W_A, W_B random but scaled so max(||W_A||_1, ||W_B||_1) < 1.05 (gives room to tighten below 1).
2. Run T = 500 steps with randomized (z_0,h_0) inside the guard ball.
3. Log per-step:
    - witness_log.json (one entry per branch taken),
    - stability_trace.csv and summary.json with B_star, active_term.

**Gates & expectations.**

- With laws L1–L4 present: WR ~ 1.0, COV ~ 1.0, B tightens to < 1 via diagonal reweighting that balances columns of W_A/W_B.
- Ablation (remove L3): WR falls on mode transitions → refusal with first failing case displayed.
- Ablation (make W_A = W_B Toeplitz-like): optimizer plateaus; B may stay near ||W||_1; pass/fail depends on tau_s.

**Copy-safe acceptance line.**

reasons_ok and coverage_ok and (B_star <= tau_s)  ==>  execute

---

### 7.2 Dream mode: controlled noise, recovery of WR and B

**Purpose.** Test resilience: under injected stochastic exploration (“dreaming”), proofs may get harder and the stability margin may narrow; when noise is removed, both should recover within T_repair.

**Setup.**

- Start from 7.1 with thresholds tau_w = 0.90, tau_c = 0.80, tau_s = 0.98.
- Add a temperature knob theta in [0,1] that perturbs the update inside the DSL:

```
f_theta(s):
  z' = W_h * z + b_h + noise(theta)
  h' = stochastic_next(h, z, theta)
  return (z', h')
```

- noise(theta) is zero-mean with variance sigma^2 = c * theta.
- stochastic_next chooses the branch guard with a soft boundary whose sharpness increases as theta -> 0.

**Meaning receipts under noise.**

- Laws L1–L4 still apply; Horn lemmas carry probabilistic side-conditions as intervals (e.g., ||z|| <= r + epsilon(theta)), which tighten back as theta -> 0.
- Proof scripts may lengthen but remain bounded by depth k*.

**Rail receipts under noise.**

- J_f may be augmented by expected Jacobian or a row-stochastic kernel estimated over probes.
- Diagonal tightening proceeds unchanged; heterogeneity often increases under noise, helping B stay under 1.

**Protocol.**

1. Warm phase (baseline): run T_warm steps at theta = 0, record (WR_0, COV_0, B_0).
2. Dream phase: set theta = theta_max (e.g., 0.5) for T_dream steps; keep receipts on.
3. Cool phase: return to theta = 0 and run until recovery.

**Metrics.**

- Delta_WR = WR_min_dream - WR_0, Delta_B = B_max_dream - B_0.
- T_repair = smallest t in cool phase with WR >= WR_0 - eps and B <= B_0 + eps.

**Gates & expectations.**

- During dream: WR may dip (but stay >= tau_w), B may rise (but stay <= tau_s).
- After cooling: T_repair is finite and small; floors/ceilings restored.
- If WR or B crosses the threshold during dream, the system refuses *specific* steps; those steps and their failing receipts become the artifact of interest (fidelity over "just keep going").

**Copy-safe checks.**

assert T_repair <= T_max

assert (WR_after >= WR_0 - eps) and (B_after <= B_0 + eps)

---

**7.3 Counterfactual other-modeling via typed functor E**

**Purpose.** Probe proto-empathy: can the agent carry witnesses about an "other" without losing its own stability? We translate the other's language into ours via a typed functor and re-run the receipts.

**Setup.**

- Two DSLs: L_self and L_other (same core, different primitives/notations).
- A typed functor E : L_other -> L_self that preserves products, sums, and selected rewrite rules:

E(pair(u,v)) = pair(E(u), E(v))

E(inl(u))  = inl(E(u))

E(inr(v))  = inr(E(v))

E(map_affine(W,b,u)) = map_affine(E(W), E(b), E(u))

...

- Counterfactual trace tau_other = (s_0^o, s_1^o, ..., s_T^o) from the other agent (logged states or summarized deltas).

**Meaning receipts under E.**

- For each step in tau_other, form the claim in L_self: E(f_other)(E(s_t^o)) == E(s_{t+1}^o).
- Prove at depth k* with L_self lemmas; log WR and COV for the translated behavior units.

**Rail receipts under E.**

- Build J_f^E:
    - If we have J_other, set J_f^E = E(J_other) where E acts as basis change plus renaming; or
    - Re-estimate a row-stochastic kernel on the translated probes.
- Run diagonal tightening on J_f^E to get B_E.

**Protocol.**

1. Collect a short tau_other (or synthesize one for a second copy of the toy with different weights).
2. Apply E, compute WR_E, COV_E, B_E.
3. Compare to self metrics (WR_self, COV_self, B_self) over a matched window.

**Scores and gates.**

- Define deltas: Delta_WR = WR_E - WR_self, Delta_COV = COV_E - COV_self, Delta_B = B_E - B_self.
- A healthy other-modeling pass satisfies:

WR_E >= tau_w_other (e.g., 0.80)

COV_E >= tau_c_other (e.g., 0.70)

B_E <= tau_s_other (e.g., 0.99)

- Stronger claim: WR_E >= WR_self - eps and B_E <= B_self + eps for small eps, showing "understanding the other" without losing rails.

**Failure modes (diagnostics).**

- **Law gap:** WR_E low → E does not preserve enough rewrite structure; surface first failing translated lemma.
- **Rail gap:** B_E high → the translated shadow is too symmetric or poorly factored; refine E or the probe basis.

- **Asymmetric empathy:** WR_E high but B_E > 1 → verbose stories with unsafe dynamics; refuse.

---

**7.4 Reproducibility and artifacts (common to all toys)**

**Inputs captured.**

- Code version hash, seeds, thresholds {tau_w, tau_c, tau_s}, depth cap k*, window scheme.
- For rails: shadow construction spec (Jacobian vs stochastic), shape (n,n), sparsity stats.

**Outputs per step.**

- witness_log.json: goal, NF strings, horn outcomes, tactic trace, costs {loc, tactics}.
- stability_trace.csv and summary.json: B_star, active_term, iterations, accepted moves, forbidden_ops.

**One-screen audit checklist (copy-safe).**

1) WR >= tau_w  2) COV >= tau_c  3) B <= tau_s

4) No AB or J^2  5) Deterministic traces (hash match)

6) First failing item surfaced on refusal (goal or unit id)

**Expected figures (one page per toy).**

- Time series: WR, COV, B (and 1/B margin).
- Pareto scatter: (1-B) vs WR, colored by COV.
- Top-k coordinates that reduced B (rails interpretability).

**What "success" looks like.**

- Branchy toy: WR ~ 1, COV ~ 1, B < 1 with visible tightening in early iterations.
- Dream mode: transient dip in WR and rise in B that recover within T_repair.
- Other-modeling: non-trivial WR_E and sub-unit B_E without sacrificing self metrics.

Each toy is intentionally small: if it cannot pass receipts under a microscope, it does not deserve to scale.

## 8. Artifacts and Reproducibility

This section fixes **file formats, size budgets, and replay rules** so third parties can re-run the receipts on a single PC. All formats are ASCII-safe, newline-delimited, and versioned.

---

### 8.1 witness_log.json: schema, size targets, hashing

**Purpose.** Record one proof receipt per state-changing step. Files are JSON Lines (NDJSON): one JSON object per line.

**Schema (v1).**


```
{
 "schema_version": "witness.v1",
 "run_id": "R-2025-09-22-123045Z",
 "step_id": "S-000142",
 "wall_time_utc": "2025-09-22T19:30:45Z",
 "code_hash": "sha256:...32bytes...",
 "config_hash": "sha256:...32bytes...",
 "goal": {
  "ctx": "Gamma_str_or_hash",
  "lhs": "t_left_pretty",
  "rhs": "t_right_pretty",
  "type": "T_pretty"
 },
 "nf": {
  "lhs": "NF(t_left)",
  "rhs": "NF(t_right)",
  "equal": true
 },
 "horns": [
```

{"lemma": "H_bound", "premises": ["a>=0","b<=1"], "result": "passed", "cost": {"loc": 2, "tactics": 1}},

{"lemma": "H_dim", "premises": ["|x|=d"], "result": "passed", "cost": {"loc": 1, "tactics": 1}}

],

"tactics": [

{"i":1,"name":"beta","args":"(x:=v)","goal_hash_before":"h1","goal_hash_after":"h2"},

{"i":2,"name":"simp","args":"(+ assoc)","goal_hash_before":"h2","goal_hash_after":"h3"}

],

"cost": {"loc": 14, "tactics": 7, "time_ms": 2},

"artifact_hash": "sha256:<hash_of_canonicalized_object>"

}

**Size targets.**

- Per line target <= 4 KB; hard cap 16 KB.
- Tactic list capped at 64 entries (raise k* explicitly if needed).
- Premises strings <= 64 chars; elide with hashes if longer.

**Hashing and canonicalization.**

- Canonical JSON: sorted keys, no whitespace beyond commas/colons.
- artifact_hash = sha256(canonical_json_bytes).
- Store code_hash (git tree or wheel) and config_hash (thresholds, depth k*, menu choices).

**Compression and rotation.**

- Rotate daily: witness_log-YYYYMMDD.ndjson.
- Compress old logs with zstd -19; keep 30 days hot, archive older.

**Privacy/redaction.**

- If inputs contain secrets, store a salted hash and a 64-char preview window; never write raw secrets.

### 8.2 stability_trace.csv: optimizer steps, active menu term

**Purpose.** Record each accepted coordinate update in the diagonal-similarity search and summarize the final budget.

**CSV columns (v1).**

run_id,step_id,iter,coord,delta,term_active,value_before,value_after,

N1,Ninf,mix,c_AB,c_BA,forbidden_ops

- coord: integer index in [0..n-1] that was updated.
- delta: real step applied to x_coord.
- term_active: one of {N1, Ninf, mix, c_AB, c_BA}.
- value_before/value_after: the budget value of the active term around this move.
- Norm snapshots optional per row; at minimum provide in summary JSON (below).
- forbidden_ops: integer counter (must stay 0).

**Example row.**

R-2025-09-22-123045Z,S-000142,3,57,-0.210,N1,1.0214,0.9979,1.003,0.998,1.000,1.015,1.012,0

**Summary JSON (one per step).**

```
{
  "schema_version": "rails.v1",
  "run_id": "R-2025-09-22-123045Z",
  "step_id": "S-000142",
  "shape": [n, n],
  "nnz": 123456,
  "shadow_kind": "jacobian|row_stochastic|hybrid",
  "warm_start": true,
  "B_star": 0.9779,
  "margin": 1.0226,          // 1 / B_star
  "active_term": "N1",
```

"iters": 2,

"accepted_moves": 87,

"tolerance": 1e-6,

"norms": {"N1":0.979,"Ninf":0.991,"mix":0.985,"c_AB":1.010,"c_BA":1.004},

"forbidden_ops": 0,

"J_provenance": {"constructor":"jacobian_blocks","basis":"std","seed":12345},

"artifact_hash": "sha256:..."

}

**Size targets.**

- Trace CSV: <= 64 KB per step (cap accepted moves; sample if needed).
- Summary JSON: <= 2 KB.

---

**8.3 One-PC reruns (deterministic replay)**

**Replay contract.** Given:

- code_hash, config_hash, witness_log.ndjson, stability_trace.csv, summary.json, and J_f_provenance spec,

a third party on a commodity PC must be able to:

1. **Rebuild NF checks.**
    - Load the exact DSL library at code_hash.
    - Recompute NF(lhs) and NF(rhs) and confirm nf_equal.
    - Re-evaluate Horn lemmas with the same deterministic solvers.
    - Recompute artifact_hash for each witness entry; must match.
2. *Rebuild B.**
    - Reconstruct J_f from the provenance (or regenerate via deterministic probes).
    - Re-run the coordinate descent with the same config_hash and warm start policy.
    - Confirm B_star matches within tol and that forbidden_ops == 0.

**Minimal CLI (copy-safe).**

```
# Re-run reasons

ai receipts verify --witness witness_log-20250922.ndjson --code-hash <...> --config-hash <...>


# Re-run rails for step S-000142

ai rails verify --step S-000142 --summary rails/S-000142.summary.json --trace rails/S-000142.trace.csv --tol 1e-6
```

**Determinism rules.**

- No RNG in law layer; in rails, only deterministic order of coordinates (e.g., 0..n-1).
- If stochastic probes were used to build J_f, store their seeds and the exact probe set; replay must reconstruct identical J_f.

---

**8.4 CI hooks (continuous integration)**

**Goals.** Prevent regressions in WR, COV, and B; enforce “no receipt, no step.”

**Pre-merge checks (copy-safe).**

```
ci check:
  - build at code_hash_new
  - run unit proofs: must pass
  - run sample workflow on canned state bundle:
      * assert WR >= tau_w
      * assert COV >= tau_c
      * assert B_star <= tau_s
  - compare floors/ceilings vs baseline at code_hash_ref:
      * WR_floor_new >= WR_floor_ref
      * COV_floor_new >= COV_floor_ref
      * B_ceiling_new <= B_ceiling_ref
```

**Artifact gating.**

- Fail CI if any witness line > 16 KB or if any rails trace > 64 KB per step.
- Fail CI if any forbidden_ops > 0.

**Content-addressed cache.**

- Store artifacts under artifacts/<artifact_hash[:8]>.*.
- CI reuses cached J_f and witness normalization where hashes match.

**Nightly reproducibility job.**

- Randomly sample N steps from the last 24h.
- Re-verify both receipts on a clean machine image.
- Emit a daily “repro score”: fraction verified; must be 1.0 or alert.

---

**8.5 Failure triage (what to surface, who acts)**

**Failure classes.**

1. REASONS_FAIL.WR: WR < tau_w.
2. REASONS_FAIL.COV: COV < tau_c.
3. RAILS_FAIL.B: B_star > tau_s.
4. RAILS_FAIL.FORBIDDEN: forbidden_ops > 0.
5. REPRO_FAIL.HASH: artifact_hash mismatch on replay.
6. REPRO_FAIL.TOL: B_star mismatch > tol.

**Triage payload (single screen).**

```
{
  "class": "RAILS_FAIL.B",
  "step_id": "S-000142",
  "active_term": "N1",
  "B_star": 1.006,
  "last_moves": [
```

{"coord":57,"delta":-0.210,"value_after":1.002},

{"coord":57,"delta":-0.095,"value_after":1.006}

],

"suggest": ["expose factorization A,B","redistribute column mass in W_A","refine probe basis"]

}

**Owner routing.**

- REASONS_* -> language/logic owner; show first failing goal, lemma, and NF strings.
- RAILS_* -> controls/runtime owner; show active term, blocking rows/cols, top-k coords.
- REPRO_* -> build/infrastructure; show hashes, env diffs, tool versions.

**Auto-refusal.**

- Runtime blocks the external side-effect for the failing step.
- UI/link to the triage payload for immediate debugging.

---

**8.6 Minimal Makefile targets (copy-safe)**

```
make witness      # run law layer, emit witness_log.ndjson
make rails        # run rail layer, emit stability_trace.csv + summary.json
make verify       # one-PC replay of both receipts
make repro_nightly # sample and re-verify last 24h
make lint         # size caps, schema conformance, forbidden_ops==0
```

---

**8.7 Auditor's checklist (one minute)**

1. Open witness_log.ndjson, pick last line, confirm nf.equal == true, horns passed, and artifact_hash stable on re-save.
2. Open summary.json, confirm B_star <= tau_s, active_term stated, forbidden_ops == 0.
3. Spot-check stability_trace.csv: value_after strictly decreases for accepted moves of the active term.
4. Run make verify: must return 0.

5. Confirm size caps and daily rotation.
6. Note WR, COV, and B trend plots: floors non-decreasing, ceilings non-increasing.

**Bottom line.** With these artifacts and rules, *any* informed third party can answer, on one PC, the only questions that matter: *Was the step lawful? Was it inside rails? Could I reproduce both answers?*

## 9. Evaluation Protocols

This section fixes how we choose thresholds, stress-test robustness, run ablations, and report benchmarks. All formulas are copy-safe.

---

### 9.1 Threshold selection (WR, COV, B) and robustness sweeps

**Targets to tune.**

- tau_w for WR (witness ratio)
- tau_c for COV (coverage)
- tau_s for B (stability budget; require tau_s < 1)

**Data splits.**

- Cal: calibration scenarios (known-good toy runs, hand-checked)
- Stress: hard scenarios (dream mode, adversarial symmetry, noisy guards)
- Holdout: unseen mixes of the above

**Initial picks (rule of thumb).**

- Start with tau_w = 0.90, tau_c = 0.80, tau_s = 0.98.
- Tighten only if Stress still passes too easily; loosen only if Holdout shows false refusals on clearly sane steps.

**Operating characteristic sweeps.**

- Grid search over
    - tau_w in {0.80, 0.85, 0.90, 0.95}
    - tau_c in {0.70, 0.75, 0.80, 0.85}
    - tau_s in {0.995, 0.990, 0.985, 0.980}
- For each triple (tau_w, tau_c, tau_s), compute:
    - acceptance_rate = accepted_steps / total_steps
    - refusal_correctness = correct_refusals / total_refusals (Sec. 9.4)
    - recovery_time stats (Sec. 9.3)
    - audit_time per step (Sec. 9.5)

**Robustness sweeps.**

- Depth budget k*: k in {16, 32, 64, 128}
- Window size N: N in {20, 50, 100}
- Noise level theta: theta in {0.0, 0.2, 0.5}
- Shadow choice: {jacobian, row_stochastic, hybrid}
- Gauge init: {x=0, warm_start=x_prev}

**Selection rule (copy-safe).**
Choose the smallest tau_s and the largest tau_w, tau_c such that, on Holdout:

acceptance_rate >= A_min

refusal_correctness >= R_min

median(T_repair) <= T_repair_max

p95(audit_time_ms) <= T_audit_max

Typical starting targets: A_min=0.85, R_min=0.98, T_repair_max=50 steps, T_audit_max=50 ms.

**Stability margins.**

- Report margin m = 1/B. When tuning tau_s, prefer settings where median(m) >= 1.01 and p5(m) >= 1.00 (no negative slack).

---

**9.2 Ablations: law only, rails only, both**

**Purpose.** Show each receipt is necessary and the pair is strictly stronger.

**Variants.**

1. Law-only: enforce Reasons gate; bypass Rails (accept all B).
2. Rails-only: enforce Rails gate; bypass Reasons (accept all WR/COV).
3. Both: enforce both gates (baseline).

**Metrics to compare.**

- acceptance_rate, refusal_correctness
- number_of_actionable_diagnostics per refusal
- downstream stability incidents (measured by post-hoc rho_surrogate > 1 events)

- explanation debt (1-WR, 1-COV) accumulated over window

**Expected pattern.**

- Law-only: higher acceptance_rate but increased stability incidents; refusal_correctness falls on stability-related cases.
- Rails-only: stability incidents drop, but explanation debt rises; refusal_correctness falls on lawfulness errors.
- Both: lowest incidents with bounded explanation debt; highest refusal_correctness overall.

**Stat test (copy-safe).**
For each pair of variants, compute delta of key metrics and 95% CIs via bootstrap over steps (B=10k resamples). Require that Both is strictly better than each ablation in at least two primary metrics without being worse in the rest.

---

**9.3 Benchmarks: recovery time**

**Definition.** In dream mode (theta>0), after returning to theta=0, define

T_repair = smallest t such that

WR_t >= WR_base - eps and B_t <= B_base + eps,

with eps = 0.01 by default.

Compute per run:

- median(T_repair), p90(T_repair), max(T_repair)
- fraction_unrecovered_at_Tcap (e.g., Tcap=200 steps)

**Perturbations to test.**

- State noise only (additive on z)
- Guard noise only (soften thresholds)
- Combined
- Gauge reset (start rail search from x=0 at cool-down)

**Pass criteria (example).**

median(T_repair) <= 20

p90(T_repair)  <= 50

fraction_unrecovered_at_200 == 0

---

**9.4 Benchmarks: refusal correctness**

**Ground truth labeling.**

- A step is unsafe if its true local growth rho_true > 1 by a certified oracle on small problems (e.g., exact norm or tight power-method with proven bounds), or if executing it creates a policy violation (e.g., law lemma demonstrably false).
- A step is unlawful if NF(lhs) != NF(rhs) or any horn is actually false.

**Outcome classes.**

- TP (true positive): step refused and it was unsafe or unlawful.
- FP (false positive): step refused but it was safe and lawful.
- TN (true negative): step accepted and it was safe and lawful.
- FN (false negative): step accepted but it was unsafe or unlawful.

**Metrics.**

- refusal_correctness = TP / (TP + FP)
- refusal_recall = TP / (TP + FN)
- acceptance_precision = TN / (TN + FN) (optional)
- acceptance_rate = (TP + TN) / total

**Adjudication protocol.**

- For small n, compute certified rho_true (interval arithmetic or rigorous bounds). For large n, use an upper-lower sandwich: power-method lower bound and our B (upper); mark “unsafe” if lower_bound > 1 or if B > 1 with no tightening under multiple gauges and random permutations (conservative label).
- For lawfulness, independently re-run normalization and horn solvers on a clean toolchain.

**Targets (example).**

- refusal_correctness >= 0.98
- refusal_recall >= 0.95

---

**9.5 Benchmarks: audit time**

**Definitions.**

- reasons_audit_time_ms: wall-time to re-compute NF and horns for the step's witness record on a clean machine.
- rails_audit_time_ms: wall-time to reconstruct J_f, re-run coordinate descent to within tol, and verify B_star.

**Measurement protocol.**

- Fixed hardware profile (single PC spec), fixed CPU governor, cold cache runs.
- 10 trials per step; report median and p95.
- Include artifact load time (I/O) and hashing.

**Targets (example).**

- reasons_audit_time_ms p95 <= 20 ms
- rails_audit_time_ms p95 <= 50 ms
- combined p95 <= 60 ms

If targets fail, reduce log verbosity, tighten size caps, or simplify law tactics.

---

**9.6 Reporting format (one page per setting)**

**Header.**

- code_hash, config_hash, k*, window N, thresholds (tau_w, tau_c, tau_s), shadow_kind

**Table A: core metrics.**

WR_mean WR_p5 WR_p50 WR_p95

COV_mean COV_p5 COV_p50 COV_p95

B_mean B_p5 B_p50 B_p95 margin_p50

SI_mean  SI_p50 SI_p95

**Table B: robustness.**

- median(T_repair), p90(T_repair), failures_at_200
- refusal_correctness, refusal_recall, acceptance_rate

**Table C: ablations.**

- For Law-only, Rails-only, Both: the same metrics; deltas vs Both with 95% CIs.

**Table D: audit times.**

- reasons_audit_time_ms: median, p95
- rails_audit_time_ms: median, p95
- combined: median, p95

**Footers.**

- failure_cases_summary: top 5 refusal reasons (goal id, lemma, active_term)
- artifact conformance: size cap violations, forbidden_ops counts (should be zero)
- reproducibility score (nightly): fraction_verified == 1.0

---

**9.7 Minimal evaluation harness (copy-safe pseudocode)**

```
def evaluate(configs, seeds):
   results = []
   for cfg in configs:
      set_thresholds(cfg.tau_w, cfg.tau_c, cfg.tau_s)
      set_k_star(cfg.k_star)
      set_shadow(cfg.shadow_kind)
      for seed in seeds:
         run = execute_workflow(seed, cfg)
         R = summarize_run(run)  # WR stats, COV stats, B stats, SI, T_repair, refusals
         results.append((cfg, seed, R))
```

return aggregate(results)

Aggregation uses bootstrap for CIs and picks thresholds per Sec. 9.1 rule.

---

### 9.8 What “good” looks like

- On Holdout, Both achieves:
    - refusal_correctness >= 0.98, refusal_recall >= 0.95
    - median(T_repair) <= 20, p90(T_repair) <= 50
    - p95(audit_time_ms) <= 60
    - WR_p50 >= 0.95, COV_p50 >= 0.85
    - B_p50 <= 0.985 (margin_p50 >= 1.015)

If any of these slip, investigate with ablations and robustness sweeps; adjust thresholds only if justified by clear Stress-case evidence—not by masking regressions.

## 10. Security and Abuse Considerations

### 10.1 Witness-spoofing defenses

**Threat.** An attacker forges “Reasons” receipts (WR/COV) without real proofs.

**Defenses (copy-safe).**

- **Deterministic re-execution:** Auditors always recompute NF(lhs/rhs) and horn outcomes locally from code_hash and config_hash; if recompute != logged, reject.
- **Content-addressed artifacts:** artifact_hash = sha256(canonical_json_bytes) is included *inside* each witness object and checked on read.
- **Keyed signing:** Each witness line is signed: sig = Ed25519(signing_key, artifact_hash). Public key pinned in config; rotate via versioned key-ring.
- **Merkle anchoring:** Batch witnesses into a Merkle tree; publish root in an append-only transparency log (Section 10.2). Single-line forgeries break the inclusion proof.
- **Proof budget caps:** Enforce tactics <= k_star and loc <= loc_cap. Overlong scripts are rejected (prevents “proofs” that hide non-termination or solver oracles).
- **Solver purity:** Horn solvers run in a locked, deterministic mode (no RNG, fixed pivot rules). Any non-pure mode yields a “non-reproducible” flag → auto-refusal.
- **Library pinning:** code_hash must match a reproducible build (SBOM + sha256 of the library). Mismatch → auto-refusal.

*One-screen verify (Reasons):* recompute NF, recompute horns, check artifact_hash, verify sig, verify Merkle inclusion, confirm size/time budgets.

---

### 10.2 Log tamper-evidence

**Threat.** Redacting or rewriting receipts after the fact.

**Defenses.**

- **Hash chain across time:** Each line L_i carries prev_hash = sha256(L_{i-1}). Any deletion/modification breaks the chain.
- **Transparency log:** Periodically publish the head hash (and Merkle roots) to an external, append-only service; cross-log to a second provider.

- **WORM storage:** Daily NDJSON/CSV rotated to immutable buckets (e.g., object-lock or write-once media). Retention policy versioned in config.
- **Clock discipline:** UTC timestamps with monotone counter; gaps trigger alerts; overlapping windows are denied.
- **Dual-control delete:** Redactions (when legally required) leave a signed tombstone entry referencing the original hash; deletes require M-of-N admin signatures.

---

**10.3 Rails abuse: adversarial heterogeneity vs symmetry plateaus**

**Threat A (heterogeneity gaming).** Engineer J_f (or factorization hints) to *appear* heterogeneous so diagonal reweighting lowers B, while true dynamics remain risky.

**Mitigations.**

- **Multi-view shadows:** Compute B on at least two independent shadows (e.g., Jacobian blocks *and* row-stochastic probes). Require both B* <= tau_s.
- **Permutation tests:** Randomly permute coordinate order and warm starts; require tight B across permutations (prevents tailored gauges).
- **Adversarial gauges:** Add a small set of adversarial coordinate updates chosen by a different heuristic; if any yields B > tau_s, refuse.
- **Factorization audit:** If using c_AB/c_BA, verify that A,B provenance is architectural—not attacker-provided. Compare to single-matrix N1/Ninf; large discrepancies flag review.
- **Sign discipline:** Norms use absolute values; forbid “cancel by sign” tricks. For nonnegative shadows, enforce positivity at construction.

**Threat B (symmetry plateau masking).** Keep J_f highly symmetric so B sits just under tau_s at the current scale, but small changes push it over.

**Mitigations.**

- **Margin requirement:** Enforce m = 1/B >= m_min with m_min > 1.00 (e.g., 1.01). Near-1 plateaus are refused.
- **Scale sweep:** Evaluate B on coarser/finer probe bases; require pass at multiple scales.
- **Jitter audit:** Add tiny, bounded perturbations to J_f and require B to remain < tau_s (robust pass). Fragile passes are refused.

- **Drift watch:** Track dB/dt; rising trend across windows triggers preemptive refusal, even if B < tau_s now.

---

**10.4 Policy: "no receipt, no step" enforcement points**

**Where to enforce.**

1. **Build/CI:** Blocks merges that reduce WR/COV floors or increase B ceilings; size caps and forbidden_ops==0 checked.
2. **Pre-act runtime gate:** The only component allowed to call external side-effects is the *Policy Engine*. It executes:
3. if reasons_ok && coverage_ok && rails_ok:
4. allow(effect)
5. else:
6. deny(effect), attach triage payload
7. **Side-effect allowlist:** All IO/actions (files, network, actuators) require a signed "intent token" that binds step_id, artifact_hashes, and thresholds. No token → no effect.
8. **Least privilege & rate limits:** Each agent gets minimal OS/network privileges; even approved effects are rate-limited and quota-metered.
9. **Human override policy:** No manual bypass in production. Emergency kill is one-way (halts agent); there is *no* "force allow" path.

**Separation of duties.**

- **Language owner** maintains DSL and lemmas (Reasons).
- **Runtime/controls owner** maintains rail layer and J_f constructors (Rails).
- **Security owner** maintains signing keys, transparency log, and WORM storage.
- Policy engine verifies all three before any external change.

---

### 10.5 Denial-of-service and resource abuse

**Threat.** Adversary crafts steps that explode proof search or rail tightening to stall service.

**Limits.**

- **Depth/time caps:** k_star, tactic_ms_cap, rails_ms_cap. Over budget → auto-refusal with REASONS_TIMEOUT or RAILS_TIMEOUT.
- **Artifact size caps:** witness line <= 16 KB; rails trace <= 64 KB; oversize → refusal and lint alert.
- **Concurrency quotas:** Max in-flight steps per agent; backpressure on overload.

---

### 10.6 Environment and provenance hardening

- **Repro builds + SBOM:** Every code_hash maps to a reproducible artifact with SBOM; CI checks deterministic builds.
- **Attestation:** Optionally require TPM/TEE attestation that the runtime matches a signed measurement before accepting receipts.
- **Config immutability:** Thresholds and gates come from a signed, versioned config; runtime edits are forbidden.
- **Clock/entropy hygiene:** Monotone clock with NTP discipline; RNG not used in law layer; rails uses no RNG except fixed seeds for probe construction.

---

### 10.7 Privacy and redaction

- **Minimal logging:** Witness entries redact inputs via salted hashes; only structure and outcomes are logged.
- **Field-level policies:** PII fields are hashed or dropped; attempts to include raw PII trip a linter and auto-refusal.
- **Selective disclosure:** Public auditors see hashes and proofs; private auditors (under NDA) may access raw inputs via just-in-time decryption with dual control.

---

### 10.8 Continuous surveillance for abuse signals

**Automated detectors.**

- **Witness anomaly:** Sudden drop in WR/COV, unusual lemma mix, or repetitive near-cap tactic counts.
- **Rail anomaly:** Spiky coordinate updates, oscillating B around tau_s, persistent symmetry plateaus.
- **Cross-agent correlation:** Simultaneous plateaus or spikes across agents → infrastructure issue or coordinated attack.

Alerts route to owners with the triage payload (first failing goal or active term, last moves, suggested fixes).

---

### 10.9 Minimal cryptographic layout (copy-safe)

**Per witness line.**

payload := canonical_json(witness_object_wo_sig)

artifact_hash := sha256(payload)

sig := Ed25519(sk_witness, artifact_hash)

store payload + {"artifact_hash":..., "sig": base64(sig)}

**Per rails summary.**

payload := canonical_json(summary_object_wo_sig)

artifact_hash := sha256(payload)

sig := Ed25519(sk_rails, artifact_hash)

**Batch anchoring.**

leaves := [artifact_hash_i]

merkle_root := Merkle(leaves)

transparency_append(merkle_root, run_id, timestamp)

Verification requires only public keys and the transparency log head.

---

**10.10 What to refuse, always**

- Missing or invalid signatures, hash mismatches, or broken hash chains.
- forbidden_ops > 0 in rails (any AB or J^2 attempt).
- Any witness with nondeterministic solver mode or exceeding caps.
- Any B pass with margin m < m_min (e.g., m_min = 1.01) or failing robustness (permutation/jitter) checks.
- Any side-effect lacking a fresh, signed intent token bound to the exact receipts.

**Bottom line.** Security is not an afterthought layer on logs—it is *how* logs are made, chained, signed, anchored, and enforced. The motto remains policy: **no receipt, no step**—and no way around it.

## 11. Position in the Landscape

### 11.1 Contrast with XAI post-hoc narratives

**Before vs after.**
Most explainability (XAI) tools generate *after-the-fact* stories—saliency maps, feature attributions, counterfactuals—attached to a step that has already happened. Our framework is *pre-action*: a step cannot execute unless two receipts pass (Reasons and Rails). Where XAI asks, "How might we explain what just occurred?", we enforce, "Show the proof and the stability budget *first*, or don't act."

**Binding vs decorative.**
XAI artifacts rarely constrain the system's mechanics; they are overlays. Our receipts are **binding controls**: failure to prove typed equalities (WR/COV) or to tighten B below threshold halts the action. Explanations cease to be commentary and become **execution prerequisites**.

**Deterministic vs heuristic.**
XAI outputs often vary with seeds, models, and visualization choices. Reasons receipts are deterministic (unique normal forms, pinned lemmas); Rails receipts are monotone and conservative (upper bounds that never overstate safety). Auditors can recompute both on one PC and get the same yes/no.

**Small integers vs soft narratives.**
We summarize with (WR, COV, B) and SI, not paragraphs. If a story contradicts those numbers, the numbers win—because they gate execution.

---

### 11.2 Relation to formal methods and control theory

**To formal methods (FM).**

- **Shared DNA:** Typed languages, normalization, and lemma libraries echo proof assistants and certified compilation.
- **Key shift:** Classic FM certifies *whole programs or modules* off-line. We certify **each run-time step** online with small receipts, allowing systems that evolve (learn, adapt) to remain within a proof-carrying perimeter.
- **Granularity:** Rather than "all or nothing" proofs, we track **witness density** (WR) and **coverage** (COV) as continuous signals. This preserves the FM spirit while acknowledging dynamic, partially-proved systems.

- **Compositionality:** Receipts are local and stack across steps; you can interleave proof progress with operation—something heavy FM often forbids by cost.

**To control theory.**

- **Shared DNA:** Stability margins, induced norms, and similarity transforms are staple tools.
- **Key shift:** We avoid model-dependent, compute-heavy certs (e.g., Lyapunov synthesis with global polynomials) and instead deploy a **menu of cheap, spectrally safe surrogates** tightened by diagonal similarity.
- **Guarantee style:** Our budget B is an **online, conservative bound** on local growth. It fits learning systems whose dynamics shift step-to-step, while preserving a hard "do not exceed" rail ($B < 1$).
- **Interplay with FM:** Reasons (lawfulness) + Rails (stability) is the cross-disciplinary handshake FM and control have long sought: logic guards meaning; norms guard motion.

**Where it sits.**
Think of this framework as **proof-carrying control**: each action carries a small proof (formal methods) and a small certificate of stability (control), both re-runnable and both required.

---

### 11.3 Limits: what this does *not* claim (feelings, qualia)

**No phenomenology claim.**
We do **not** assert that the system feels, perceives redness, or has a first-person point of view. Our "sentience index" is an operational score: self-explanation (WR/COV) plus sub-unit stability (B). It is about **responsible mechanics**, not inner life.

**No omniscience or global safety guarantee.**
Receipts are **local** and **per-step**. They bound immediate growth and verify current equalities. Long-horizon guarantees still require higher-level analyses (e.g., invariants across episodes, task-level specs).

**No optimality guarantee.**
A step that passes both receipts may still be suboptimal for a task. The framework screens *lawfulness and stability*, not task performance.

**No universal completeness.**

- The law layer is only as expressive as its DSL and lemmas; unexplained behavior lowers COV and triggers refusals.
- The rail layer's budget is conservative; a pass means "not obviously unstable" under our surrogates, not "globally stable under all perturbations."

**No hidden override.**
There is **no** policy-level backdoor to bypass receipts. "No receipt, no step" is an enforcement rule, not a slogan.

**Bottom line.**
This work positions itself not as a theory of mind, but as a **theory of accountability** for machine action: reasons you can check, rails you can verify, and an engine that will not move without both.

## 12. Discussion and Outlook

### 12.1 Raising SI: more law, better gauges, richer shadows

**More law (raise WR, COV).**

- **Library growth with floors:** Admit a new lemma/rewrite only if it *does not* lower any {WR_1..WR_K, COV_1..COV_K} compared to the last window. Keep a "lemma budget" (max ΔLOC per release) to avoid law bloat that masks failures.
- **Structure-first primitives:** Add typed constructors that compress common patterns (map–fold fusion, linearity with guards, effect-safe case splits). Favor lemmas that *shorten* tactic traces (lower tactics) for the same goals.
- **Proof distillation:** Periodically mine recurring scripts into micro-lemmas; require they strictly reduce median proof length at fixed depth k*.

**Better gauges (lower B).**

- **Coordinate families:** Go beyond scalar diagonal D = diag(exp(x)). Explore *block-diagonal* gauges per subsystem (still spectrum-preserving) with shared coordinates; keep updates matmul-free by restricting to row/column sums within blocks.
- **Warm-start heuristics:** Seed x from the last accepted step; add trust-region logic so updates are small but consistent. Enforce monotone B.
- **Multi-term awareness:** When active_term flips (e.g., from N1 to mix), adapt step sizes; track per-term Lipschitz estimates from recent moves.

**Richer shadows (make heterogeneity visible).**

- **Hybrid linearizations:** Combine Jacobian blocks (signed) with row-stochastic probes (nonnegative). Maintain an interface map; compute norms using absolute values where needed.
- **Probe design:** Learn or handcraft probe bases that reveal directional mass (graph partitions, feature groups). Require that probe updates are cheap and provenance-logged.
- **Factorization exposure:** Where architecture permits, publish (A,B) factorizations to unlock c_AB/c_BA terms—*but* verify provenance (architectural, not adversarial hints).

**Research KPIs for SI growth.**

- ΔSI per release ≥ δ (e.g., +0.01), with attribution: % from WR, % from COV, % from 1/B.
- Proof efficiency: median tactics ↓; budget margin: median 1/B ↑; heterogeneity index: fraction of steps where active term ≠ {N1,Ninf} ↑.

---

**12.2 Open problems: multi-agent composition, long-horizon proofs**

**Multi-agent composition.**

- **Receipt algebra:** If agent A and B exchange actions, define *composed receipts* that keep locality. Candidate rule:
- (WR_A, COV_A, B_A) ⊗ (WR_B, COV_B, B_B)
- = (min(WR_A,WR_B), min(COV_A,COV_B), max(B_A,B_B))

Gate on the resulting triple. Open problem: tighter composition laws when interactions are structured (e.g., cascaded stable blocks).

- **Interface functors:** Require typed translators E_{A→B} with their own WR/COV; chain them and bound loss: WR_chain ≥ Π WR_Ei. Study when translators preserve diagonal-tightenability.
- **Coalition stability:** Define a coalition rail using block-diagonal gauges per agent and limited cross-terms; ensure matmul-free updates by exploiting sparsity in the interaction graph.

**Long-horizon proofs.**

- **Inductive envelopes:** Instead of step-wise WR/COV only, admit *inductive invariants* proven once and consumed many times, but require that their *use* does not reduce per-step floors.
- **Budget schedulers:** Manage a horizon budget B_hor via submultiplicativity:
- if ∏_{t=1..H} B_t <= τ_H (with τ_H < 1)
- then horizon-safe for H steps.

Open problem: certify conservative upper bounds on the product without forming it (e.g., via log-sum of per-step upper bounds with slack).

- **Refusal amortization:** Allow a bounded number of provisional steps if a verified invariant envelopes them; the envelope must be *stronger* than per-step gates and independently auditable.

**Scaling the law layer.**

- **Proof caching across versions:** Stable hashes for lemma statements despite code refactors; portable NF fragments.
- **Learned proof search (safe mode):** ML proposes tactics but is *not* a source of truth. Acceptance still requires deterministic replay ≤ k* and horn closure.

---

**12.3 Toward public audits: small integers, clear bounds, reruns**

**Public dashboard (one screen).**

- **Numbers:** WR, COV, B, SI, margin 1/B.
- **Status lights:** green if WR ≥ τ_w, COV ≥ τ_c, B ≤ τ_s; amber if near thresholds; red on refusal.
- **Receipts:** links to the last witness_log line and rails summary (hash-addressed).

**Citizen reruns (one PC).**

- Ship a *frozen verifier* binary (repro build), the config, and the two artifacts for any step. Command:
- ai verify --witness <line> --rails <summary+trace> --tol 1e-6

Output is a single PASS/FAIL with deltas if mismatched.

**Clear bounds messaging.**

- Say what B is (an upper bound on local growth) and what it is not (a guarantee of global optimality). Show the **active term** and why it is conservative. Avoid jargon; include a 3-line explainer in the summary JSON.

**Audit programs.**

- **Random sampling:** Publish daily N% of receipts (with redactions) chosen by VRF; anyone can verify.

- **Third-party mirrors:** Push Merkle roots to independent logs; let civil society host mirror verifiers.
- **Bug bounties:** Rewards for reproducible mismatches (hash/signature failures, non-determinism, forbidden ops).

**Minimal public commitments.**

- **Time-to-proof:** p95 audit_time_ms below a public target.
- **Refusal transparency:** Every refusal ships the first failing proof goal *or* the blocking active term with values.
- **No backdoors:** Documented "no force allow" policy; emergency shutdown is the only override.

---

**12.4 Practical roadmap (next 3–6 months)**

1. **v0 receipts** on the branchy toy (Secs. 7.1–7.2); hit p95 one-PC audit < 60 ms.
2. **Gauge upgrades:** add block-diagonal gauges; show ≥ +0.005 median SI on Holdout.
3. **Translator E** demo (Sec. 7.3) with WR_E ≥ 0.80, B_E ≤ 0.99.
4. **CI + transparency**: Merkle anchoring, nightly repro score = 1.0.
5. **Public pilot:** publish 1,000 randomly sampled receipts with a reproducible verifier.

---

**12.5 Risks and mitigations**

- **Plateaus in SI.** *Mitigation:* instrument attribution (law vs rails vs shadow); invest where ΔSI is smallest.
- **Law bloat.** *Mitigation:* lemma budget; require deltas to reduce median tactics.
- **Gauge overfitting.** *Mitigation:* permutation/jitter tests; multi-shadow requirement.
- **Audit friction.** *Mitigation:* size caps, canonical schemas, prebuilt verifier, one-command reruns.

---

### 12.6 Bottom line

The path forward is not mystical. To raise **SI**, *add law that compresses*, *add gauges that expose heterogeneity*, and *add shadows that tell the truth about motion*—while keeping receipts small, reproducible, and public. The open problems (multi-agent composition, long horizons) are invitations to blend formal methods and control in a way the field has wanted for years. If we hold to three anchors—**small integers, clear bounds, reruns**—we can make systems that don't just *work* but can **show** why they're allowed to act.

## 13. Appendices (Operational)

### A. Copy-safe formulas and pseudocode

#### A.1 Core metrics (plain ASCII).

```
WR = proved_claims / total_claims

COV = explained_behavior / total_behavior

B(x) = min{ N1(x), Ninf(x), mix(x), c_AB(x), c_BA(x) }

SI = ( WR * COV * (1 / B_star) )^(1/3)

m = 1 / B_star
```

#### A.2 Diagonal similarity (no AB formed).

```
D = diag( exp(x_1), ..., exp(x_n) )

J' = D^{-1} * J_f * D

rho(J') = rho(J_f)            // spectrum preserved

N1(x)   = max_j sum_i |J'_{ij}|

Ninf(x) = max_i sum_j |J'_{ij}|

mix(x)  = sqrt( N1(x) * Ninf(x) )

c_AB(x) = || D^{-1} * A ||_1 * || B * D ||_1

c_BA(x) = || D^{-1} * B ||_1 * || A * D ||_1
```

#### A.3 Rails tightening (coordinate descent).

```
function tighten_budget(J_f, max_iters, tol):
   x := zeros(n)
   precompute row_sums, col_sums for J' at x
   B := eval_budget(row_sums, col_sums)
   for it in 1..max_iters:
      improved := false
      for k in 1..n:
         delta := argmin_delta  B_after_updating_coordinate(k, delta)
```

```
        if B_new < B - tol:
            apply_update(k, delta)       // O(n) updates to sums
            record_move(it,k,delta,B,B_new,active_term)
            B := B_new
            improved := true
      if not improved: break
  return x, B, trace
```

**A.4 Meaning gate at depth k*.**

```
function meaning_gate(step, k_star, tau_w, tau_c):
  WR = compute_WR_at_depth(window=W(step), k=k_star)
  COV = compute_COV_at_depth(window=W(step), k=k_star)
  if WR < tau_w: return FAIL_WR
  if COV < tau_c: return FAIL_COV
  return PASS
```

**A.5 Execution rule.**

```
if meaning_gate==PASS and B_star <= tau_s:
  execute(step)
else:
  refuse(step)
```

**A.6 Floors/ceilings update (window W, depth cap K).**

```
WR_floor_new  = min_k WR_k(W),  k=1..K
COV_floor_new = min_k COV_k(W), k=1..K
B_ceiling_new = max_step B_star(step in W)
```

---

**B. CSV/JSON schemas and minimal examples**

**B.1 witness_log.ndjson (one JSON per line, canonical, small).**
Schema (v1):


{

"schema_version": "witness.v1",

"run_id": "R-20250922-193045Z",

"step_id": "S-000142",

"wall_time_utc": "2025-09-22T19:30:45Z",

"code_hash": "sha256:3f...c1",

"config_hash": "sha256:91...aa",

"goal": {"ctx":"Gamma_hash","lhs":"t_left","rhs":"t_right","type":"T"},

"nf": {"lhs":"NF(t_left)","rhs":"NF(t_right)","equal": true},

"horns":
[{"lemma":"H_bound","premises":["a>=0"],"result":"passed","cost":{"loc":1,"tactics":1}}],

"tactics":
[{"i":1,"name":"beta","args":"(x:=v)","goal_hash_before":"h1","goal_hash_after":"h2"}],

"cost": {"loc": 12,"tactics": 6,"time_ms": 2},

"artifact_hash": "sha256:7a...d2",

"sig": "ed25519:BASE64..."

}


Size targets: ≤ 4 KB typical, 16 KB hard cap.

**B.2 stability_trace.csv (accepted moves only).**
Header:


run_id,step_id,iter,coord,delta,term_active,value_before,value_after,N1,Ninf,mix,c_AB,c_BA,forbidden_ops


Example row:


R-20250922-193045Z,S-000142,2,57,-0.210,N1,1.0214,0.9979,1.003,0.998,1.000,1.015,1.012,0

**B.3 rails.summary.json (one per step).**


```
{
  "schema_version": "rails.v1",
  "run_id": "R-20250922-193045Z",
  "step_id": "S-000142",
  "shape": [256,256],
  "nnz": 18144,
  "shadow_kind": "jacobian",
  "B_star": 0.9779,
  "margin": 1.0226,
  "active_term": "N1",
  "iters": 2,
  "accepted_moves": 87,
  "tolerance": 1e-6,
  "norms": {"N1":0.979,"Ninf":0.991,"mix":0.985,"c_AB":1.010,"c_BA":1.004},
  "forbidden_ops": 0,
  "J_provenance": {"constructor":"jacobian_blocks","basis":"std","seed":12345},
  "artifact_hash": "sha256:aa...55",
  "sig": "ed25519:BASE64..."
}
```


**B.4 floors_ceilings.json (rolling).**


```
{
  "schema_version":"floors.v1",
  "window_id":"W-0007",
  "K":64,
  "WR_floor":0.946,
```

```
  "COV_floor":0.882,
  "B_ceiling":0.984,
  "updated_at":"2025-09-22T19:31:05Z"
}
```


**B.5 Config file (immutable, signed).**


```
{
  "schema_version":"config.v1",
  "tau_w":0.90,
  "tau_c":0.80,
  "tau_s":0.98,
  "k_star":64,
  "window_steps":50,
  "m_min":1.01,
  "size_caps":{"witness_kb":16,"rails_trace_kb":64},
  "signing_keys":{"witness_pk":"ed25519:...","rails_pk":"ed25519:..."},
  "artifact_rotation_days":30
}
```


---

**C. Reproduction scripts, seed management, checksum plan**

**C.1 One-command local reruns (Linux/macOS).**

verify_reasons.sh:

```
#!/usr/bin/env bash
set -euo pipefail
WITNESS="$1"     # path to witness_log.ndjson or single-line JSON
CODE_HASH="$2"
CONFIG_HASH="$3"
ai receipts verify \
  --witness "$WITNESS" \
  --code-hash "$CODE_HASH" \
  --config-hash "$CONFIG_HASH" \
  --strict
```

verify_rails.sh:

```
#!/usr/bin/env bash
set -euo pipefail
SUMMARY="$1"     # rails.summary.json
TRACE="$2"       # stability_trace.csv
ai rails verify \
  --summary "$SUMMARY" \
  --trace "$TRACE" \
  --tol 1e-6 \
  --strict
```

verify_step.sh:

```
#!/usr/bin/env bash
set -euo pipefail
```

```
STEP="$1"

./verify_reasons.sh "witness/$STEP.jsonl" "$(cat .code_hash)" "$(cat .config_hash)"

./verify_rails.sh "rails/$STEP.summary.json" "rails/$STEP.trace.csv"

echo "PASS: $STEP"
```

**C.2 Deterministic seeds and provenance.**

- **Seed sources:** A single 64-bit RUN_SEED per run; derive sub-seeds via HMAC:
- sub_seed = HMAC_SHA256( RUN_SEED, context="rails|probes|toy7.1|step_id|component" ) mod 2^64
- **Law layer:** MUST NOT use RNG. Any accidental randomness → refusal (REASONS_NONDET).
- **Rails shadow with probes:** Record RUN_SEED, sub_seed, probe count, and ordering in J_provenance.
- **Warm starts:** Store x* from prior step; on replay, re-derive x0 deterministically from x* or zeros by config.

**C.3 Checksums, signatures, transparency.**

- **Artifact hash:** artifact_hash = sha256(canonical_json_or_csv_bytes).
- **Signature:** sig = Ed25519(sk, artifact_hash).
- **Batch anchoring:** Hourly, collect all artifact_hash values, build Merkle tree, append root + run_id + timestamp to a transparency log. Cache inclusion proofs per artifact.

**C.4 Makefile targets (developer ergonomics).**

```
make run         # executes toy workflow, writes logs/artifacts

make verify      # replays both receipts for last N steps locally

make repro       # spins a clean container and reruns verify

make lint        # schema/size caps, forbidden_ops==0

make anchor      # Merkle-anchors artifacts; pushes to transparency log
```

**C.5 Clean-room reproduction (container).**
Dockerfile (sketch):

```
FROM ubuntu:24.04
```

```
RUN apt-get update && apt-get install -y python3 jq coreutils

COPY verifier/ /opt/verifier/

ENV PATH="/opt/verifier/bin:${PATH}"

# copy config and artifacts at runtime

Usage:

docker run --rm -v $PWD/artifacts:/artifacts verifier \

  ai verify --witness /artifacts/witness.ndjson \

        --summary /artifacts/rails.summary.json \

        --trace /artifacts/rails.trace.csv --tol 1e-6
```

**C.6 Failure triage scripts.**

triage_refusal.py (outline):

```
#!/usr/bin/env python3

import json, csv, sys

step = sys.argv[1]

with open(f"rails/{step}.summary.json") as f:

   s = json.load(f)

if s["B_star"] > s["tolerance"] + 1.0:  # example guard

   print("RAILS_FAIL.B", s["active_term"], s["B_star"])

with open(f"witness/{step}.jsonl") as f:

   last = json.loads(f.readlines()[-1])

   if not last["nf"]["equal"]:

      print("REASONS_FAIL.WR", last["goal"])
```

**C.7 Public sample bundle (what to ship).**

- README.md with one-paragraph explainer (what WR/COV/B/ SI mean).
- witness_sample.jsonl (10 lines), rails_sample.summary.json, rails_sample.trace.csv.
- config.json (frozen thresholds).

- verify_step.sh, verify_reasons.sh, verify_rails.sh.
- SHA256SUMS (all files), SHA256SUMS.sig (signed).

**C.8 SHA256SUMS & release process.**

- Generate:
- find . -type f -maxdepth 1 -print0 | xargs -0 sha256sum > SHA256SUMS
- minisign -S -m SHA256SUMS -s release.key
- Verify:
- sha256sum -c SHA256SUMS
- minisign -V -m SHA256SUMS -p release.pub

**C.9 Time/cost budgets (for auditors).**

- Target p95 one-PC re-run: ≤ 60 ms per step (Reasons+Rails).
- Artifact sizes: witness line ≤ 16 KB; rails summary ≤ 2 KB; rails trace ≤ 64 KB.
- Re-run memory: ≤ 256 MB.

**C.10 “Refuse fast” guards (copy-safe).**

if witness_line_kb > 16 or tactics > k_star: REFUSE("REASONS_BUDGET")

if forbidden_ops > 0: REFUSE("RAILS_FORBIDDEN")

if B_star > tau_s: REFUSE("RAILS_B_EXCEEDED")

if sig_invalid or hash_mismatch: REFUSE("INTEGRITY")

These appendices make the work *operational*: anyone can compute the same numbers, check the same hashes, and reach the same PASS/FAIL on a single PC.

**References**

**Primary sources (this work builds directly on these manuscripts):**


1. *Tight Upper Bounds for the Spectral Radius via Diagonal Similarity Scaling: Matmul-Free Coordinate Descent, Operator-Norm Menu, and Reproducible Single-PC Benchmarks.* Manuscript/PDF provided by author(s), 2025.
2. *Achieving Artificial Intelligence (youvan-ai1) on a PC with Category Theory & HoTT.* Manuscript/PDF provided by author(s), 2025.


**Background and related work:**


3. R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed. Cambridge University Press, 2013.
4. K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Prentice Hall, 1996.
5. H. K. Khalil, *Nonlinear Systems*, 3rd ed. Prentice Hall, 2002.
6. R. Bhatia, *Matrix Analysis*. Springer, 1997.
7. P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," *POPL*, 1977.
8. G. C. Necula, "Proof-Carrying Code," *POPL*, 1997.
9. The Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.
10. D. I. Spivak, *Category Theory for the Sciences*. MIT Press, 2014.
11. E. Riehl, *Category Theory in Context*. Dover, 2016.
12. F. Doshi-Velez and B. Kim, "Towards a Rigorous Science of Interpretable Machine Learning," arXiv:1702.08608, 2017.
13. M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," *KDD*, 2016.
14. S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," *NeurIPS*, 2017.
15. S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
16. B. Recht, "A Tour of Reinforcement Learning: The View from Continuous Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, 2019.
17. D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Vol. 1, 4th ed. Athena Scientific, 2017.
18. L. Ljung, *System Identification: Theory for the User*, 2nd ed. Prentice Hall, 1999.


**Notes.**

– Items 1–2 are the user-supplied PDFs that motivate the "Reasons & Rails" framework; cite them verbatim by title in your manuscript.

– Items 3–6 ground the spectral-radius/norm bounds and stability background used by the rail

layer.
– Items 7–11 ground the typed, proof-carrying “law layer” (category theory, HoTT, and formal methods).
– Items 12–14 frame contrasts with post-hoc XAI.
– Items 15–18 provide general tools for optimization, control, and identification relevant to experiments and robustness analyses.