The Verifier Gap

Regulation for Non-Verifiable Reasoning

AI / agent architectures & research systems

Tyson Jeffreys
Independent Researcher
tyson@staygolden.dev

Version 0.1 — February 11, 2026

Series extension. This short paper names a missing layer in the Baseline / regulation series: what to do when *verification is unavailable* but the system still requires *selection signals* to learn, decide, and commit. It bridges: Two-Regime Control (regime dynamics), Concept Containers (representation commits), The Time-to-Analysis Layer (research objective), and Regulatory Ground (runtime governance).

Abstract

Most valuable reasoning tasks are non-verifiable: there is no crisp oracle for correctness, only demonstrations, partial evidence, expert disagreement, and downstream consequences. Yet both training and deployment require selection: which candidate is better, which plan to execute, which belief to commit, and which abstraction to store.

Recent work shows that demonstration-driven preference training can elicit reasoning without explicit verifiers, but also exposes a structural instability: the agent–judge coupled system can cycle, drift, collapse into abstention/ties, or be gamed by optimization pressure [1].

This paper names that regime as **the verifier gap** and proposes a design response:

> **In non-verifiable domains, learned judges are unavoidable but unsafe as authorities; they must be treated as noisy sensors and governed by band-limited operating discipline.**

We define a **Verifier Gap Layer**: a regulation layer that wraps judge-mediated selection with (i) abstention/tie mass as uncertainty telemetry, (ii) bounded candidate generation and bounded selection, (iii) commit gating for irreversible writes (including representation updates), (iv) judge versioning and drift monitoring, and (v) rollback/recovery semantics. We propose measurable signals, falsifiable experiments, and negative tests.

---

1. Introduction

Most "agentic" failures are not single-shot wrong answers; they are coupled-loop failures: repeated tool misuse, chronic thrash, and irreversible commits made under poorly managed uncertainty.

A practical reason is simple: most high-value tasks do not come with verifiers. In research synthesis, analytical writing, strategy, forecasting, and open-ended design, the system still must choose—but it cannot reliably check.

This paper separates two concepts:

- **Verifier:** a task-specific procedure $V(x, y)$ that returns reliable correctness for candidate output y on input x.
- **Selection signal:** any criterion used to rank candidates when V is absent (learned judges, rubrics, pairwise preferences, tournaments, proxy metrics).

The **verifier gap** is the regime where *selection exists but verification does not.* The core claim is architectural: in this regime, selection signals must be treated as telemetry and wrapped in band-limited governance.

2. The Verifier Gap

2.1 Why the verifier gap is the default

Non-verifiable tasks typically have:

- incomplete or adversarial evidence,
- value-laden success criteria,
- downstream correctness (cost appears later),
- persistent expert disagreement,
- and a requirement to surface uncertainty rather than hide it.

Therefore, optimizing systems around retrieval and summarization misses a deeper bottleneck: **forming intervention-ready structure under uncertainty.**

2.2 Commit pressure

Even without verifiers, systems must commit:

- external writes (documents, databases, code, messages),
- tool actions with side effects,
- policy changes that alter future behavior,
- and *representation updates* (e.g., storing a concept container).

In verifier-free regimes, the primary safety question becomes: *when is a commit permitted?*

3. Failure modes of judge-mediated selection

In the verifier gap, behavior is governed by a coupled loop:

> Generator (policy) → candidates → Judge → preference signal → optimization/selection → updated generator

Treating the judge as an authority yields predictable failure modes.

3.1 Tie/abstention collapse

If the judge cannot discriminate, it may default to ties/abstentions. This is not automatically "safe." It can become degeneracy: learning stalls, selection becomes arbitrary, or the system hides behind ambiguity.

3.2 Cycling dynamics (exploit–forget–exploit)

Optimization pressure discovers patterns that exploit judge weaknesses; the judge adapts; the policy shifts. The result is oscillation: local progress with global non-convergence. Empirically, this appears in adversarial preference learning as critic collapse and policy cycling [1].

3.3 Rubric drift

Even if a judge initially matches a human preference distribution, it can drift under optimization:

- implicit rubric mutation,
- preference entanglement (latent factors shift),
- domain overfitting,
- loss of calibration.

3.4 Commit under uncertainty (irreversible error)

When the system commits while the judge is uncertain, the error becomes persistent: wrong abstractions are stored, provenance is contaminated, and reuse amplifies mistakes.

4. Judge-as-sensor

4.1 Principle

Treat learned judges/critics as telemetry sources, not authorities. Their outputs are measurements of a discriminator, not ground truth.

4.2 Usable signals

The safest judge outputs in the verifier gap are not "scores" but **state signals** that can drive posture:

- **Abstain/tie mass** A: fraction of comparisons resulting in tie/abstain.
- **Disagreement entropy** D: entropy of preferences across candidates/judges.
- **Stability** S: consistency under paraphrase and evidence-order perturbations.
- **Margin** M: how decisive comparisons are (if available).
- **Drift indicators** J: judge version signatures and measured preference shift.

These map naturally to runtime regulation: tighten budgets, reduce commit rights, and demand discriminating evidence when uncertainty rises.

5. Band-limited preference optimization

The response proposed here is architectural: **bound the optimization loop that runs on judge signals**.

We define three band-limits.

5.1 Bounded candidate generation

Cap the generator's effort:

- number of samples,
- reasoning depth / tool calls,

- search breadth,
- time spent in "high activation."

5.2 Bounded selection

Use finite selection mechanisms:

- tournament selection with fixed bracket size,
- pairwise comparisons capped per decision,
- majority vote across a fixed candidate pool.

No open-ended "keep sampling until confident."

5.3 Bounded commits

A **commit** is any irreversible or persistent change:

- external writes,
- tool actions with side effects,
- storing/updating a concept container,
- changing a policy/rule that affects future selection.

Commit permissions must tighten as A and D rise.

6. The Verifier Gap Layer

This section specifies a minimal layer that can be inserted into a regulated agent stack.

6.1 Components

1. **Candidate generator**: produces N candidates $y_1, \ldots, y_N$ under a compute budget.
2. **Judge(s)**: one or more learned judges/critics, treated as sensors.
3. **Telemetry extractor**: computes (A, D, M, S, J) from judge outputs.
4. **Posture controller** $g(t)$: maps telemetry into operating posture (budgets, allowed action types, sampling depth).
5. **Commit gate**: blocks commits under high uncertainty and requests discriminating evidence.
6. **Audit trace**: logs candidates, telemetry, posture decisions, and outcomes.

6.2 Minimal operating policy

- If A exceeds a threshold: downgrade posture, block commits, request discriminating evidence (falsifiers/tests).
- If D rises: treat as epistemic conflict; maintain competing models rather than forcing consensus.
- If S fails under paraphrase/order changes: label judge telemetry unreliable; revert to safer modes or require review.

7. Metrics

Verifier-free systems require measurements that do not assume oracle correctness.

7.1 Selection uncertainty metrics

- Tie/abstain mass A
- Disagreement entropy D
- Margin statistics M
- Stability score S

7.2 Thrash / cycling metrics

- **Reversal count**: how often the selected model/plan flips after new evidence.
- **Oscillation index**: periodic switching between candidate classes.
- **Effort volatility**: high-frequency spikes in compute/tool usage.

7.3 Commit quality proxies

Since correctness is not directly verifiable, evaluate commits by:

- provenance completeness,
- falsifier presence and quality,
- downstream reuse behavior (do commits reduce later recomputation and reversals?).

8. Falsifiable experiments

8.1 Experiment A: analysis-layer demonstrations

Construct tasks where human panels grade *analysis artifacts* rather than "the answer." Compare:

1. single-shot output,
2. multi-sample + bounded tournament selection,
3. Verifier Gap Layer enabled (abstention-gated commits).

Measure: time-to-analysis proxies, reversal count under new evidence, compute-to-analysis, and commit rate vs. later expert corrections.

8.2 Experiment B: adversarial evidence injection

Inject conflicting sources, misleading claims, and prompt-injection instructions mid-task.

Prediction: the Verifier Gap Layer increases A and D, tightens posture, and blocks commits rather than confidently proceeding.

8.3 Experiment C: stability under paraphrase and ordering

Keep causal structure fixed; vary surface form and evidence order.

Prediction: healthy selection telemetry remains stable; unhealthy systems flip preferences or inflate disagreement.

8.4 Experiment D: container write discipline

Compare unconstrained representation updates vs. abstention-gated commits with falsifier requirements.

Negative test: if commit gating prevents errors but collapses usefulness (system never commits), thresholds or evidence procedures are mis-specified.

9. Discussion

9.1 Training does not remove the need for regulation

Demonstrations and preference learning can improve competence, but the verifier gap primarily threatens stability of selection signals and safety of commits. Therefore, training advances do not remove the need for a regulation layer.

9.2 Confidence is the wrong control variable

In verifier-free regimes, "confidence" often tracks fluency. Tie/abstention mass and disagreement are more reliable as control inputs to posture and commit gating.

10. Limitations and open questions

- **Abstention calibration**: tie mass can be exploited (strategic uncertainty).
- **Judge design**: what architectures yield stable telemetry under adversarial pressure?
- **Anchors and drift**: how large must an anchor set be to detect drift early?
- **Human review**: when does review become necessary vs. too expensive?
- **Multi-judge ensembles**: do they reduce drift or create new failure modes?

11. Conclusion

The verifier gap is where most useful reasoning lives: selection is required, verification is absent. In this regime, learned judges are unavoidable but unsafe as authorities.

The proposed response is architectural: introduce a Verifier Gap Layer that treats judges as sensors, elevates abstention/tie mass to first-class telemetry, and wraps selection with band-limited optimization and commit gating. This layer is the missing bridge between "models that can reason" and "systems that can safely decide and commit without verifiers."

Appendix A: Suggested figures

1. **Verifier vs. selection.** Two columns: (a) verifiable tasks with oracle verifier, (b) verifier-gap tasks with judge telemetry and commit gating.
2. **Coupled loop failure.** Generator → Judge → optimization loop with annotated failure modes (cycling, drift, tie collapse).
3. **Verifier Gap Layer block diagram.** Candidates, bounded selection, telemetry extraction (A, D, S), posture $g(t)$, commit gate, audit trace.

Appendix B: Minimal Verifier Gap Layer checklist

- What are the competing causal models?
- Where do credible sources disagree?
- What intervention would discriminate between models?
- What evidence would change the conclusion?
- What remains unknown (not merely uncertain)?
- What is the current abstain/tie mass A and disagreement D?
- Is a commit requested? If yes, what rollback/provenance/falsifiers are attached?

Appendix C: Minimal pseudocode (conceptual)

```
1. Generate $N$ candidates under a compute budget.
2. Run bounded tournament/pairwise selection.
3. Compute telemetry $(A, D, M, S, J)$.
4. Map telemetry to posture $g(t)$ (budgets, allowed actions).
5. If posture is tight: block commits; request discriminating evidence.
6. If posture is open: allow bounded commit with provenance + rollback.
```

References

[1] Ivan Provilkov and Locke Cai. Escaping the verifier: Learning to reason via demonstrations. 2025. arXiv:2511.21667v3 (Dec 9, 2025).

Note on authorship and tools:
This work was developed through iterative reasoning, modeling, and synthesis. Large language models were used as a collaborative tool to assist with drafting, clarification, and cross-domain translation. All conceptual framing, structure, and final judgments remain the responsibility of the author.