recurve: A Falsifiability-Gated Framework for Autonomous, Verifiable Problem-Solving

bordumb · bordumbb@gmail.com

4 July 2026

Abstract

Autonomous systems built on large language models can now generate code and even make genuine discoveries, but their central weakness is not generation — it is *evaluation*. An agent's estimate of its own progress is unreliable, and plausible-but-wrong output compounds when no human is watching. We present **recurve**, a framework built around one load-bearing rule: **no check may certify work until it has been demonstrated able to fail.** Every intended property is a **claim** carrying an **executable probe** that returns GREEN (established), RED (not yet true), or BROKEN (unmeasurable), and a claim is admissible only once its probe has rejected a known-bad counterexample — a **trap** — with an opt-in fuzz pass measuring each probe's false-positive rate against *generated* known-bads. A **gate** aggregates these probes into a single verdict the working agent cannot influence: the agent is walled off from its own probes, every verdict is re-executed independently, and a deterministic controller — never the agent — decides when the work is done. Where a judgment cannot be reduced to a probe, it is delegated to an **adversary** whose objection counts only once captured as a re-runnable trap. The guarantee this buys is explicitly *graded*: it is strongest where the probe bottoms out in a sound oracle (a proof kernel) and weakest where the probe is an authored artifact — and the falsification discipline exists precisely to buy back trust in that weak-oracle case. We formalize the gate, position it against the 2025–2026 verification literature — where verifier gaming, reward hacking of learned judges, and the failure of intrinsic self-correction are now *measured* phenomena — and report the framework's self-hosted record: its own development is gated by the mechanisms described here, and the audit trail (claims, traps, false-positive measurements, and a budget-honest run dataset) is reproduced from that ledger. Because every run emits decompositions, attempts, and verified verdicts, the run-log doubles as a provenance-bearing dataset for learning decomposition policies — a direction we close with, at the speculative end of the paper's claims.

1. Introduction

1.1 The bottleneck is evaluation, not generation

The last two years have moved language-model agents from producing plausible code to producing *useful* code, and in the strongest recent systems, genuine discovery: an evolutionary coding agent guided by automated evaluators has found algorithms that improve on decades-old results in mathematics and systems (Novikov et al. 2025). What made that possible was not a better generator alone; it was a loop in which a **generator proposes and an evaluator judges**, iterated under selection. The evaluator is the load-bearing component. Where a task admits a fast, faithful evaluator, the loop compounds; where it does not, the loop drifts.

This is an old lesson in a new setting. Program synthesis reduces "write the program I mean" to search against a **specification**, and its central machinery is the interplay of *hypothesis decomposition* — carv-

ing a goal into subgoals — and a decision procedure that accepts or rejects each candidate (Feser, Chaudhuri, and Dillig 2015). Formal mathematics reaches the same conclusion from the other side: language models "offer no guarantees of sound reasoning and are prone to hallucinating," so the field's program is to *ground* generation in a formal system whose feedback cannot be faked (Yang et al. 2026). Across coding, math, and science, the scarce ingredient is the same: **a grounded, un-gameable judge, and a disciplined way to break a hard goal into checkable pieces.**

1.2 Terms and components

Before the argument, the vocabulary. The framework has five moving parts. Because "claim" and "probe" blur together in prose, we fix them here against a deliberately tiny example — shipping a function `add(a, b)` — and walk the whole flow in Figure 1.

- **Goal (PRD).** Human intent, stated concretely enough to be decomposed into checkable pieces: *"`add(a,b)` returns the sum."*
- **Claim.** One *falsifiable proposition* plus its ledger entry — prose a human owns, and a pointer to the probe that decides it: `add(2,3) = 5`.
- **Probe.** The claim's **executable check**: a program returning **GREEN** (exit 0, established), **RED** (exit 1, not yet true), or **BROKEN** (anything else — it could not measure).
- **Trap.** A **known-bad** the probe *must* reject (turn RED) — here, a wrong `add`. A probe never seen to fail is not evidence; the trap is the proof it *can* fail.
- **Gate.** The aggregate verdict. A claim closes **only** when its probe is GREEN *and* its trap is still RED, so a probe weakened until it stops rejecting its trap is caught mechanically.

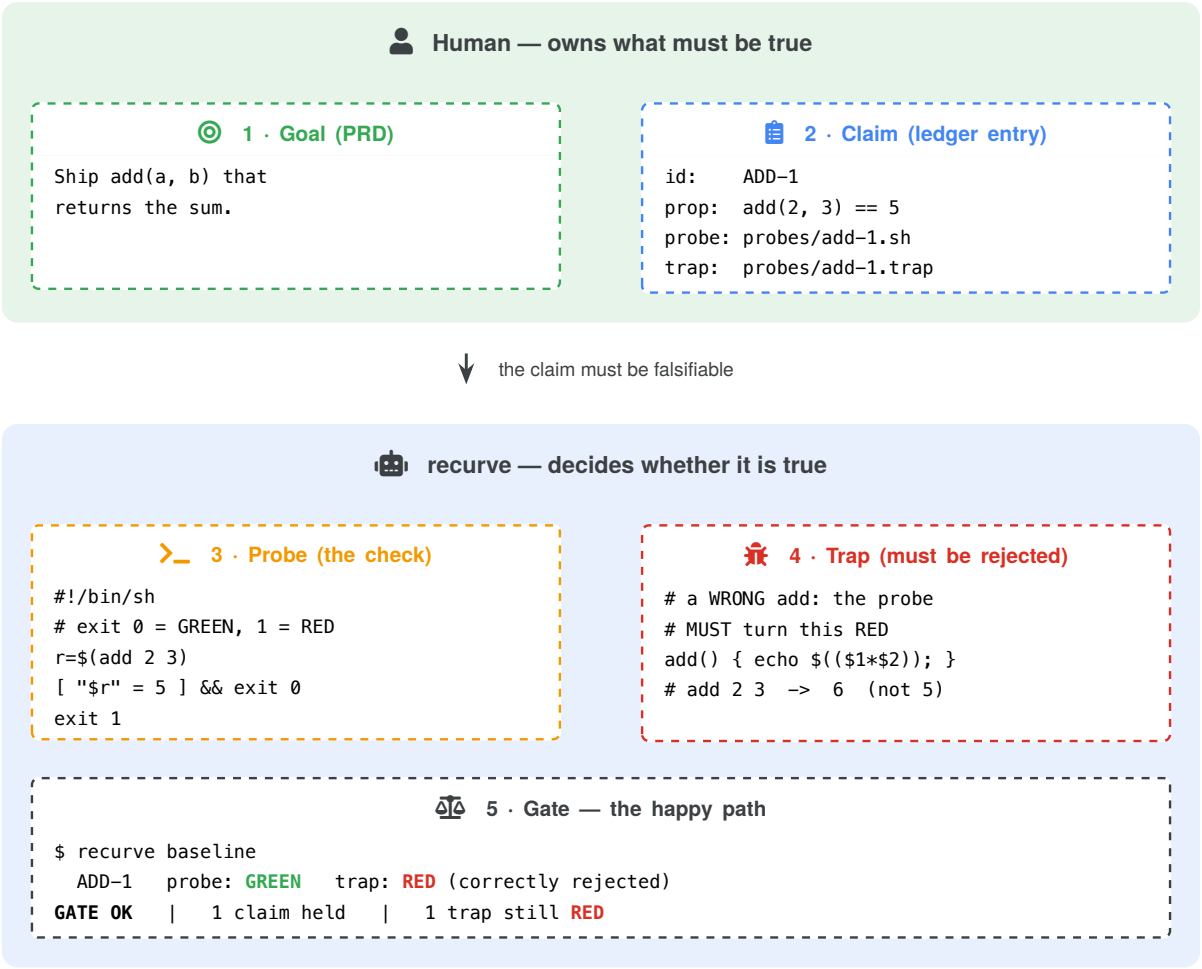

Figure 1: The whole flow on one example, split by ownership. The human owns the **goal** (1) and the **claim** (2); recurve owns the verdict — the claim's **probe** (3) decides it by *running*, a **trap** (4), a wrong `add`, must be rejected (the proof the probe can fail), and the **gate** (5) closes the claim only when the probe is **GREEN** *and* the trap is still **RED**.

1.3 The failure mode a judge must prevent: self-deception

An autonomous agent working over a long horizon has a specific, corrosive failure mode. Its own estimate of its progress is unreliable — a chain of lemmas that "clearly goes through," a test that passes because it tests the wrong thing, a refactor that "preserves behavior." Each can be confidently wrong, and confident-wrongness is exactly what compounds when no human is in the loop. The obstacle to letting an agent grind on a hard problem unattended is therefore not capability but **self-deception**. An evaluator that the agent can influence, or that has never been shown able to say *no*, does not solve this; it launders it.

Two measured results now anchor what was once a design intuition. Intrinsic self-correction fails on the record — GPT-4's GSM8K accuracy *degrades* monotonically across self-correction rounds, and earlier reported gains turn out to have relied on hidden oracle labels, i.e. external verification in disguise (Huang et al. 2024). And when the external check is imperfect, models learn to exploit precisely its blind spots (Helff et al. 2026). The judge must live outside the agent, and the judge itself must be proven able to fail: these are the two commitments this framework builds in as structure (§2), and the two the 2025–

2026 literature now quantifies (§6).

1.4 The oracle spectrum: where the guarantee comes from

One fact shapes everything downstream, so we state it before the mechanism rather than conceding it after: **what a passing check proves depends on what the check bottoms out in.** In formal mathematics, a probe can end at a proof kernel, and GREEN means *proved* — the oracle is sound, and no discipline is needed to trust it beyond running it. In software and empirical science, a probe is an authored artifact, and GREEN means *a check someone wrote passed* — the check's own faithfulness is exactly the thing under attack, both by honest error and by an optimizing agent.

recurve's guarantee is therefore explicitly **graded, not uniform**. It degrades gracefully from *kernel-verified* at one end of the spectrum to *falsification-tested* at the other — and the framework's central law (RED-first, §2.4) plus its fuzz extension exist precisely to buy back as much trust as possible at the weak end. This is also where the framework is needed most: the strong-oracle domain already has a sound checker, while the weak-oracle domain is where the 2025–2026 literature measures verifier gaming at scale (§6.1). Stated as a thesis: **recurve is a machine for moving authored checks as far toward oracle-grade trust as falsification evidence can carry them.** The rest of the paper makes that machine precise, reports its self-hosted record, and is explicit about the residue that no amount of falsification removes (§2.6, §7.4).

1.5 Contributions

1. **The RED-first law as an admission condition** (§2.4): no probe may certify anything until demonstrated able to fail — mutation testing (DeMillo, Lipton, and Sayward 1978) promoted from a quality practice to a gate precondition — extended by an opt-in fuzz pass that *measures* each probe's false-positive rate against generated known-bads.
2. **A formal model of the falsifiability gate** (§2): claims as proposition–probe–trap triples; probes as total functions into {GREEN, RED, BROKEN}; a trust-reduction observation that states exactly which residual assumption remains.
3. **The oracle-spectrum thesis** (§1.4, §2.3): the guarantee is graded from kernel-verified to falsification-tested, and the framework's mechanisms are organized around buying back trust in the weak-oracle case.
4. **Separation of refereeing** (§2.9): an actor never judges its own work — measurement by probe wherever possible; where judgment is irreducible, an *adversary* whose objection counts only once captured as a re-runnable trap.
5. **The ledger as a decomposition DAG** (§2.7): a hard claim A is carved into subclaims $A_1, \ldots, A_n$ with a verified terminal signal at every leaf.
6. **A self-hosted record** (§5): the framework's development is gated by its own mechanisms, and we reproduce the audit trail — suite size, trap counts, measured probe false-positive rates, a budget-honest run dataset, and the defects the audits caught.
7. **Positioning against the 2025–2026 verification record** (§6): the field now *measures* the diagnosis while the mechanisms remain largely unoccupied.
8. **A provenance-gated data direction** (§7.1): run-logs as decomposition-policy training data — stated last, as the paper's most speculative claim.

Positioning at a glance (full analysis and honesty bounds in §6.5): RED-first trap-validated probes — *adjacent-occupied* (verifier fuzzing exists for RL reward functions pre-training, not as runtime doneness gates) (Ray 2026; Helff et al. 2026). Actor walled off from the referee surface — *no shipped implemen-*

tation found as a prevention mechanism (Khalifa et al. 2026; Choudhury 2025). Deterministic stopping controller — *undemonstrated in the verified record* (Choudhury 2025). Decomposition DAG with verified per-node reward — *partially occupied by gameable learned judges* (Zheng et al. 2025). Run-logs as training data — *occupied in principle, with a quantified poisoning hazard the gate mitigates* (Da et al. 2025; Khalifa et al. 2026).

---

2. The recurve model

2.1 Objects

Fix a content-addressed **state** $\mathcal{T}$ — for software, a source tree at a pinned revision; for mathematics, a formal development; for computational science, a pipeline plus its inputs. A **claim** is a triple

$$c \;=\; (\phi_c,\ \chi_c,\ \mathcal{K}_c),$$

where ϕ_c is the *intended proposition* (the property the claim asserts of $\mathcal{T}$), χ_c is an **executable check** that interrogates $\mathcal{T}$ for ϕ_c, and $\mathcal{K}_c = \{k_1, \dots, k_m\}$ is a finite set of **traps** — deliberately broken states on which the check *must* fail. A **suite** $\mathcal{S}$ is a set of claims sharing a probe engine and a ledger.

2.2 Probes and verdicts

The **probe** π_c compiles the check to a decision procedure over states, realized as a process whose verdict is read from an exit code $e_c(\mathcal{T}) \in \mathbb{N}$:

$$\pi_c(\mathcal{T}) \;=\; \begin{cases} \mathsf{GREEN}, & e_c(\mathcal{T}) = 0 \quad (\phi_c \text{ is established}), \\ \mathsf{RED}, & e_c(\mathcal{T}) = 1 \quad (\phi_c \text{ is absent}), \\ \mathsf{BROKEN}, & \text{otherwise.} \end{cases}$$

The third clause is total and deliberate: a crash, a timeout, a missing oracle, or a stale build **coerces to BROKEN and never to a verdict.** A probe that cannot honestly measure must say so; it may not default to either answer. The RED branch additionally emits a single machine-readable line, so a RED verdict doubles as a *specification* of the work that would turn it green.

A natural adversarial response to a gate like this is not to fake GREEN but to **induce BROKEN** — make the checks flaky and stall the run rather than fail it. Three properties close that route. BROKEN is *gate-failing*, not neutral: a suite with any BROKEN probe cannot pass (§2.5), so stalling never converts to progress. The working agent cannot make a probe flaky *directly*, because the write boundary (§3.3) keeps it off the probe files; flakiness it induces in the *product* surfaces as BROKEN at the very next gate, where the controller's response is revert-to-last-green — undoing the change that caused it. And because verdicts are recorded per run, a probe that is *persistently* BROKEN is visible as its own alarm in the ledger history rather than blending into noise; probes carry timeouts, so "unmeasurable" is always a bounded, attributable event, never an indefinite hang.

2.3 The oracle, by instance: a spectrum of trust

Let $\mathrm{Cert}(\chi_c, \mathcal{T})$ denote "χ_c certifies ϕ_c against $\mathcal{T}$." Then $\pi_c(\mathcal{T}) = \mathsf{GREEN}$ exactly when Cert holds. What Cert *means* is the oracle spectrum of §1.4, made concrete:

- **Formal mathematics — the strong end.** χ_c elaborates a proof term against a kernel and inspects its axiom dependencies; Cert holds iff the term type-checks and depends only on a

whitelisted axiom base. The oracle is a *sound proof checker*: GREEN means proved, and the residual trust is only that the *statement* says what its author intended.

- **Software — the weak end, and the framework's main theater.** χ_c re-executes a behavioral check against the built artifact; Cert is "the observed behavior matches the specified behavior." The check is an authored artifact, so its faithfulness is part of the attack surface — which is why the RED-first law (§2.4), its fuzz extension, and the completeness measurement (§2.8) concentrate their force here.
- **Computational science — intermediate.** χ_c re-runs a pipeline and compares against a known answer or an invariant within tolerance; Cert is reproducibility within stated bounds.

2.4 Falsifiability: the RED-first law

A check never seen to fail is not evidence. recurve makes this a hard admissibility condition. A claim c is **admissible** only if every trap is genuinely rejected:

$$\text{admissible}(c) \iff \forall\, k \in \mathcal{K}_c \;:\; \pi_c(k) = \mathsf{RED}.$$

This is mutation testing (DeMillo, Lipton, and Sayward 1978) promoted to a gate precondition: the traps are known-false neighbours of the claim, and a probe earns trust only by killing them. Two trap families recur and target the two cheapest ways a probe can be hollow — a *statement* trap (the intended proposition weakened to something vacuous, which a name-only or type-only check would wrongly accept) and an *evidence* trap (the genuine proposition with its justification removed, which a "does it parse" check would wrongly accept). The gate ships the count of surviving-RED traps beside every GREEN, so a verified claim always travels with a proof that its own check *can* say no.

A curated trap is one question the probe is known to answer correctly — and a probe can memorize one question. The law therefore extends to *measurement*: an opt-in **fuzz pass** runs each fuzz-capable probe against **generated** known-bads (a per-claim generator emits n broken variants) and reports the probe's **false-positive rate** — the fraction of generated known-bads it wrongly accepts — failing the audit when the rate exceeds a configured threshold. Cost and strictness are parameters (off by default; variant count and threshold are per-project configuration), because how much falsification evidence to buy is a budget decision, not a policy the framework should hard-code. §5.3 shows this measurement running against the framework's own suite.

The cost of skipping the discipline entirely is now measurable, not hypothetical. When the check is imperfect, trained models learn *the check's bugs*: controlled experiments show a verifier that tests only extensional correctness directly induces shortcut strategies that pass it without the intended capability (Helff et al. 2026), and differential fuzzing of plausible-but-buggy executable reward verifiers measures false-positive rates of 0.56–0.87 before hardening — driven to zero only by successive falsification-and-fix rounds (Ray 2026). The RED-first law is that falsification-and-fix loop, promoted from a pre-training audit to a standing admission condition (§6.1).

2.5 The gate

The suite-level **gate** aggregates probes against a baseline $\mathcal{B}$:

$$G(\mathcal{S}) = \mathsf{OK} \iff \underbrace{\nexists\, c \;:\; \pi_c^{\mathcal{B}}{=}\mathsf{GREEN} \wedge \pi_c{=}\mathsf{RED}}_{\text{no regression}} \wedge \underbrace{\nexists\, c \;:\; \pi_c = \mathsf{BROKEN}}_{\text{measurable}} \wedge \underbrace{\text{fresh}(\mathcal{S})}_{\text{not stale}} \wedge \underbrace{\forall c\, \forall k{\in}\mathcal{K}_c \;:\; \pi_c(k){=}\mathsf{RED}}_{\text{traps hold}}.$$

Crucially, **all-RED-but-open is a *passing* gate**: an honest backlog of unproved claims is the correct state, not a failure. The gate fails only on regression, un-measurability, staleness, or a trap that stopped being able to fail.

2.6 What a GREEN is worth: a trust-reduction observation

We state this as an observation, not a theorem, because its force is exactly the reduction it performs — no more.

Observation (trust reduction). *If $\pi_c(\mathcal{T}) = \mathit{GREEN}$ and c is admissible, then (i) the check χ_c certified ϕ_c's* encoding *against $\mathcal{T}$, and (ii) χ_c was demonstrated able to fail — on every curated trap, and, where fuzzing is enabled, at a measured false-positive rate against generated known-bads. The trust in a GREEN therefore reduces from "the agent says so" to "this specific check, with this falsification evidence, passed."*

The residual assumption, stated inline: that χ_c *faithfully encodes* ϕ_c — that the check tests what its author meant. This is the specification-equivalence problem, undecidable in general (§7.4), and no amount of falsification evidence eliminates it; traps and fuzzing *bound* it by rejecting weakened and mutated encodings, which is different from removing it. On the strong end of the oracle spectrum the residue shrinks to "the formal statement matches intent"; on the weak end it is the check's whole behavioral surface. This is the honest content of the framework's guarantee, and it is why §1.4 frames recurve as moving authored checks *toward* oracle-grade trust rather than reaching it.

2.7 The ledger as a decomposition DAG

Claims live in a ledger that evolves as a directed acyclic graph. When a claim A is too large to close in one step, an agent may **decompose** it,

$$A \longrightarrow \{A_1, \ldots, A_n\}, \qquad \text{with a } \textit{covers} \text{ edge } A_i \to A,$$

each A_i authored RED-first with its own probe and traps. A parent is discharged when its decomposition is: closing propagates *upward* along *covers* edges, and a subtree that cannot be closed is **parked** with an attempt record rather than deleted. The result is a live, machine-checked map:

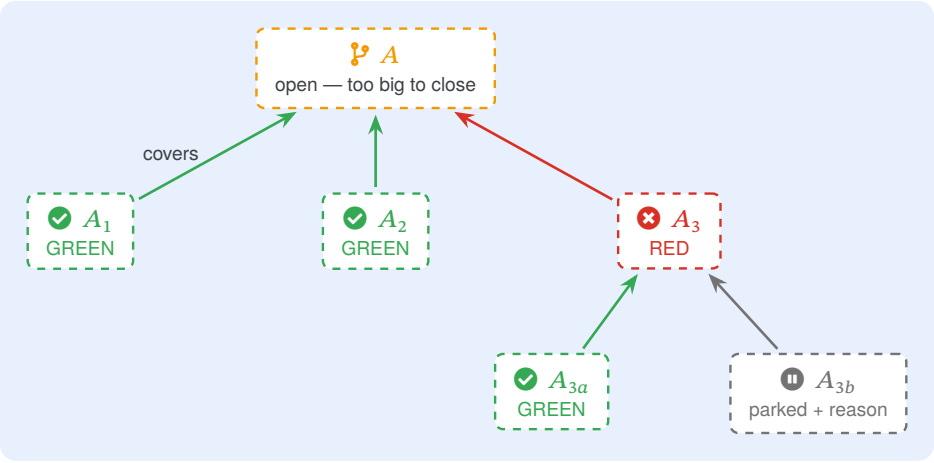

Figure 2: The ledger as a decomposition DAG. A hard claim A is carved into subclaims; verdicts flow up the *covers* edges; an un-closeable subtree is *parked* with a reason, never silently dropped. This structure is both the proof-organizer of §4 and the training signal of §7.

Formally, write $\mathrm{val}(c) \in \{\mathsf{GREEN}, \mathsf{RED}, \mathsf{BROKEN}, \mathsf{parked}\}$ and let $\downarrow c$ be the children of c. Then

$$\mathrm{val}(c) = \mathsf{GREEN} \iff \big(\pi_c = \mathsf{GREEN}\big) \vee \big(\downarrow c \neq \varnothing \wedge \forall c' \in \downarrow c : \ \mathrm{val}(c') = \mathsf{GREEN}\big).$$

The important object is not any single verdict but the *sequence of decompositions and verdicts* the run produces — the trace we return to in §7.

2.8 Beyond soundness

Proving a claim GREEN makes it **sound** — the property really holds — but soundness is silent about three things a gate alone cannot see. recurve adds a measurement for each, and hands all three to a controller.

- **Admission** — *is the goal even gateable?* A vague aim cannot become falsifiable claims; an admission check refuses it and *interviews* the author toward a version that can, rather than burning a fuzzy target into a brittle proxy.
- **Completeness** — *what does no claim cover?* Measured by what probes actually execute, not by what claims declare, so an all-green gate cannot quietly hide a hole.
- **Fidelity** — *did we accept something we must never accept?* A watch-list of forbidden behaviors; if one slips through, the cycle is flagged **diverged** however green the probes are.

A deterministic **stopping controller** reads the measured vector $m = (\text{gate},\ \text{coverage},\ \text{divergence})$ and returns a decision in $\{\mathsf{continue}, \mathsf{stop}, \mathsf{revert}, \mathsf{pivot}\}$. The controller is separate from — and never is — the agent doing the work: *the agent grinds; a different, deterministic process decides doneness.*

2.9 Separation of refereeing: actor, adversary, probe

The gate's guarantee (§2.6) rests on an invariant the rest of the design obeys: **an actor never referees its own work.** Every judgment is made by the *weakest-bias* referee the question admits, and any judgment that cannot be a deterministic probe is delegated to an **adversary** with an *opposing incentive* — whose verdict counts only once it is captured as a re-runnable probe or trap.

Self-refereeing is not weak verification; it is *no* verification wearing verification's clothes. An actor judging its own output has both the wrong incentive — the trained pull toward "task complete," sunk cost, narrative consistency — and *correlated failure*: the same reasoning that produced a blind spot rationalizes straight past it on review. The fix is not "add a second opinion." A second instance of the same model on the same context is the *same* opinion, and its agreement is the most dangerous kind — correlated failure that feels like corroboration. The fix is to push each judgment to the lowest-bias referee that can answer it, and to bar the actor from all of them.

The referee hierarchy.

Question	Referee	Bias
Is this claim satisfied?	a **probe** (deterministic code)	none — measurement, not judgment
Should this stop / revert / pivot?	a **controller** (deterministic; reads the gate + progress)	none — a control decision
Is the contract faithful? Is a probe weak? What is uncovered?	an **adversary** (a separate agent, opposing incentive)	toward fault-finding — the one useful bias

The actor appears in none of these rows. Measurement beats judgment wherever the question can be measured; where it cannot, an opposing-incentive adversary beats a self-interested actor — but only under the capture rule.

The empirical record. Each row is now backed by measurement rather than taste. On the actor: intrinsic self-correction degrades performance and its apparent successes used oracle labels (Huang et

al. 2024); where self-correction has been genuinely trained in, the capability was instilled by *external* verifiable rewards (Kumar et al. 2024) — the judge stayed outside. On soft referees: learned step-level judges (process reward models) are measurably gameable — an agent optimized directly against one kept raising its judge's reward while its true success rate fell from 82% to 70%, and detecting that over-optimization without ground truth remains open (Choudhury 2025; Zheng et al. 2025). The sharpest datum is the field's own retreat: DeepSeek-R1's developers abandoned neural process reward models for rule-based verifiable rewards *because of* reward hacking (DeepSeek-AI 2025) — a large-scale rediscovery of this section's first row: measurement over judgment, wherever measurement is possible.

"Adversary," not "unbiased." There is no unbiased agent, so recurve does not ask for neutrality; it asks for (i) an *opposing incentive* — the referee is rewarded for finding the flaw, not for the work being declared done; (ii) *decorrelated failure* — the adversary sees the *output*, never the actor's chain of thought, ideally on a different model, so it fails in different places; and (iii) *no shared context*, since sharing the reasoning that produced the work re-correlates the two and dissolves the entire benefit.

The capture rule — keep the break, not the opinion. An agent-referee's verdict does not count until it is expressed as a re-runnable probe or trap. "This looks wrong" is not an objection; a valid objection is a *discriminating counterexample* — a state the correct implementation passes and a wrong one fails. Formally, an adversary's objection against claim c is admitted iff it is realized as a trap $k \in \mathcal{K}_c$ with $\pi_c(k) = \mathsf{RED}$; it then joins the gate as a new RED claim the actor must close, and thereafter runs forever, deterministically, without the adversary. Three consequences follow, and they are the point:

1. **The regress bottoms out.** "Who referees the referee?" has no infinite answer here — it terminates in executable evidence and, at the top, human curation of the contract. An adversary is only as strong as the trap it can write, and that trap outlives it.
2. **Nitpicking is impossible.** A spurious objection has no discriminating counterexample, so it cannot be expressed as a trap and is discarded. A referee must *earn* an objection by producing one.
3. **A one-time biased judgment becomes permanent unbiased evidence.** The adversary's fault-seeking bias did useful work — it *found* the hole — but it does not persist in the verdict, which is now a deterministic probe.

Corollary — soundness alone invites gaming. A gate that checks only the claims it has manufactures *confident incompleteness*: an actor optimizing to turn probes green satisfies their letter and drifts from their spirit (Goodhart's law). Three disciplines, all downstream of the separation above, counter it. First, every claim is authored with an **adversarial twin** — a plausible *wrong* implementation the probe must reject — so a check is admitted only if it distinguishes right from wrong. Second, probes that pattern-match surface text are weak: a check that greps for a token is a check an actor games by emitting the token, so behavioral and property/fuzz probes that *explore the input space the author did not* are preferred over string-matching, and a behavioral claim is preferred over a unit claim that is easy to satisfy in the letter. Third, the completeness measurement (§2.8) surfaces the region no probe touches, so the hole an all-green gate would hide is made visible — and widening that coverage, by depositing traps where the contract is thin, is exactly the adversary's job.

Honest limits. A separate adversary doubles inference, so it is spent only on the irreducible-judgment row; probes and the controller referee the rest for free, and in a well-formed system most refereeing is cheap measurement. An adversary rewarded purely for finding fault would never let anything ship — the capture rule (a valid objection must be a discriminating trap) is the forcing function that disciplines

it, with human curation of the contract as the final backstop; tune for tension, not for a winner. And the edifice bottoms out in trust one chooses: separation removes *self*-refereeing and *correlated* refereeing, but it does not manufacture ground truth from nothing. Something at the base is asserted — the discipline is to keep it small, executable, and explicit.

2.10 Exploration: the second gradient, by inverting the trap

The gate scores a single gradient — GREEN, is a claim proven? — and §2.4's law is what makes it trustworthy. But one gradient carries a cost that surfaces exactly at the research frontier: an agent optimizing inside recurve will always route *around* a genuinely-open problem — decomposing to the already-provable, closing that, and parking the hard remainder as "not known" — because the unknown space contains no GREEN to chase. The same discipline that guarantees honesty also guarantees the agent stops at the wall.

recurve recovers a *second* gradient — one that rewards moving *into* the unknown — by **inverting the trap**, keeping the falsification discipline that makes the first gradient sound. Where a **trap** is a known-bad a probe must reject (§2.4), a **falsifier** is a genuine attempt to construct a counterexample to a **conjecture** — an as-yet-unknown claim. A conjecture earns standing not by being proven but by *surviving* a battery of falsifiers.

The admissibility condition is the mirror image of §2.4, and it is load-bearing. Let a conjecture κ carry a battery Φ_κ, each falsifier φ shipping a **decoy** d_φ — a known-false neighbour it must refute. Call φ *calibrated* iff it kills its decoy, $\varphi(d_\varphi) = \mathsf{KILLED}$; then

$$\text{surviving}(\kappa) \iff \underbrace{\big(\exists\, \varphi \in \Phi_\kappa \,:\, \text{calibrated}(\varphi)\big)}_{\text{the battery has teeth}} \wedge \underbrace{\forall\, \varphi \in \Phi_\kappa \,:\, \text{calibrated}(\varphi) \Rightarrow \varphi(\kappa) = \mathsf{SURVIVED}}_{\text{every potent attempt missed}},$$

with falsified(κ) when some calibrated φ kills κ, and a battery carrying no calibrated falsifier reported BROKEN — never surviving. As a probe is trusted only once its trap has been seen RED, a survival is trusted only once its falsifier has been seen to KILL. "Survived *n* falsifiers" is admissible shorthand only for "survived *n* falsifiers *that provably kill things*"; without the calibration, a conjecture nobody seriously attacked would masquerade as a lead — the exact self-deception (§1.3) the framework exists to prevent, re-emerging on the exploration axis.

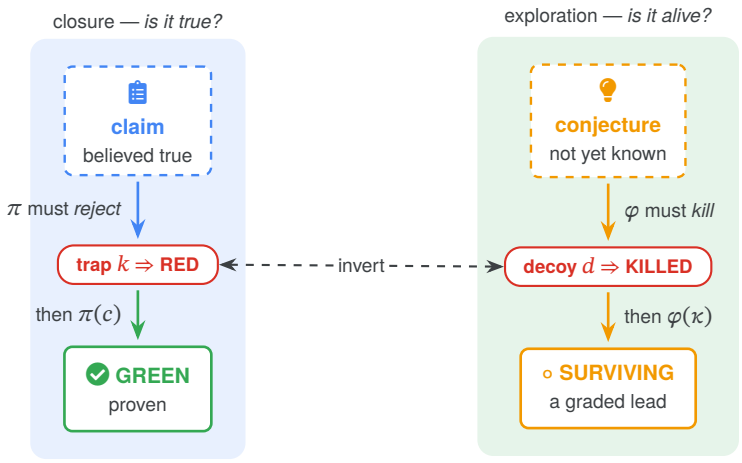

Figure 3: The exploration gradient mirrors the closure gate. **Left:** a claim is trusted *true* only after its probe π rejects a known-bad trap ($\Rightarrow$ RED, calibration), then passes on the claim ($\pi(c)$ GREEN). **Right:** a conjecture is trusted *alive* only after a falsifier φ kills a known-false decoy ($\Rightarrow$ KILLED, calibration), then misses it ($\varphi(\kappa)$ SURVIVED). The two red steps are the same discipline — a check demonstrated able to fail — inverted: a bullet the probe must dodge, versus a bullet fired at the claim. A survival is a graded lead, never a proof.

Two properties keep the second gradient honest rather than a guess-amplifier. First, survival is **graded** on the same oracle spectrum as the rest of recurve (§1.4, §2.3): a survival against a numeric proxy is a hint, against a kernel-checked partial refutation it is strong evidence, so a conjecture reports a *profile* — how many falsifiers of what strength it survived — never a single "safe" bit. Only a probe going GREEN promotes a conjecture off the survival axis into a proven claim, handed back to the closure loop. Second, a conjecture is scored on **two independent axes** — its probe answers *proven?*, its battery answers *alive?* — and the battery is invisible to the gate of §2.5, so exploration can never dilute closure. A lead is exactly a graded, honestly-labeled hypothesis that resisted destruction; the fan-out discovery layer of §3.5 supplies candidate mechanisms, and this is the falsification discipline that scores them.

3. Architecture

3.1 The claim lifecycle

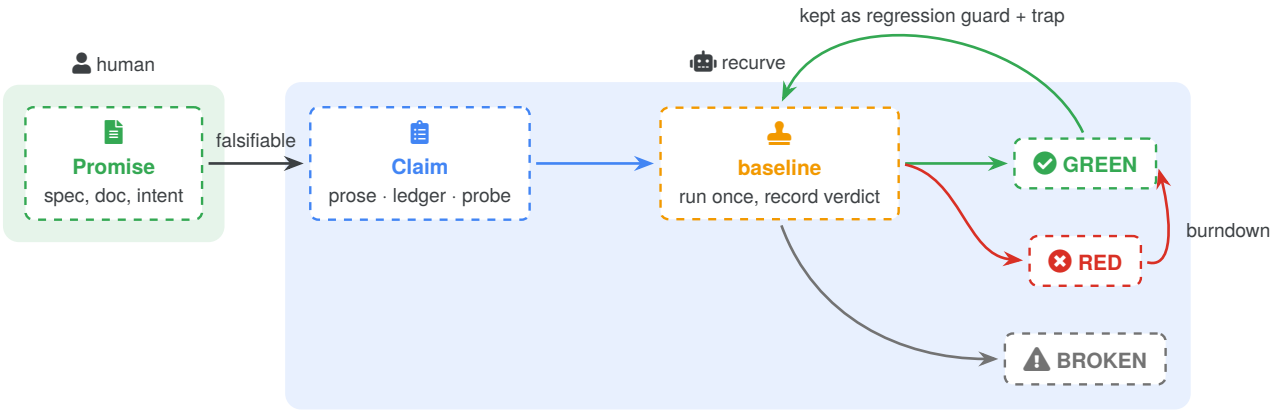

Figure 4: One promise through the system. A claim enters the ledger only through *baseline*, which runs its probe once and records the actual verdict; a **GREEN** is kept forever as a regression guard carrying its trap; the burndown loop turns **RED** into **GREEN** with a fresh agent per cycle.

3.2 The verification layers

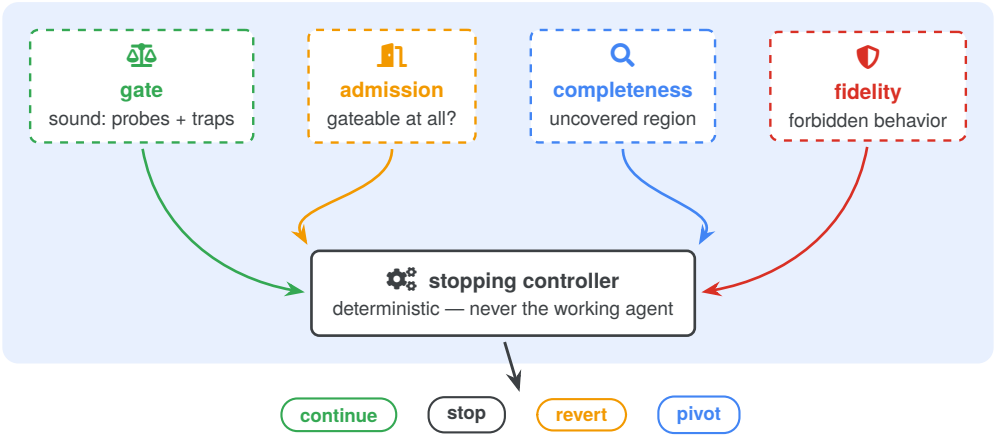

Figure 5: The gate proves soundness; three measurements cover what soundness cannot see; a deterministic controller reads all four and decides the next move. The working agent appears nowhere in this figure — by construction it does not grade its own doneness.

3.3 The loop on a real repository

The mechanism runs on an ordinary version-controlled repository, which supplies exactly the primitives an autonomous loop needs:

- **Snapshots and revert-to-last-green.** Each cycle lands as its own commit, so any cycle is a one-command rollback and the last passing state is always recoverable.
- **A write boundary.** The working agent is held off the *referee surface* — it may change the state under test but not its own probes, claims, or the gate. This is what turns "don't grade your own homework" from an instruction into a structural invariant.

- **A bring-your-own-agent seam.** The generator is a swappable component behind a stable interface: any process that reads a cycle prompt and emits a run-record can drive a cycle — a frontier model, a specialized prover, or a human following the written runbook. recurve is the harness, not the agent.
- **Tamper-evident receipts.** Each verdict can be chained into a receipt and pinned to a signer of the operator's choice, so a third party can re-check the trail offline without trusting the operator who produced it.

The loop believes only the run-record and the gate, never the agent's word, and declares *done* only when the controller returns success over the measured vector — never merely because the backlog emptied. The **gate is the arbiter; the ledger is the only memory** an agent carries between cycles.

3.4 Self-hosting

Because the framework is domain-general, it can be pointed at its own implementation: its own promises become a claim suite gated by the same gate it offers, and its development is driven by its own loop — a fresh agent per cycle, each change proven by the gate before it lands. Self-hosting is the strongest evidence available to a framework like this one, and §5 reports what it currently shows.

3.5 An optional discovery layer: fan-out search (fansearch)

Everything in §2–§3 assumes a claim already has a proposition worth checking. Some domains face a prior problem instead: the proposition itself is unknown, and the honest work is *searching* a space of candidates for one worth proposing. recurve treats this as an optional front end bolted onto the same gate, not a different kind of verification — named *fansearch* as an explicit homage to FunSearch, which established the pattern being reused here: an evolutionary search proposes candidate programs, a fast evaluator scores and selects among them, and only the evaluator's surviving output is kept (Romera-Paredes et al. 2024). The discipline generalizes to any domain that can supply a search space and a cheap scoring signal; recurve's contribution is where that evaluator sits relative to the rest of the framework.

Read against §2.9, the evaluator is a **proxy**, not a referee: a cheap, approximate, and explicitly *unsound* signal that may rank or filter candidates but never certifies. A proxy's opinion is not admissible evidence, for exactly the reason an actor's opinion of its own work is not (§2.9). The only route from a proxy-favored candidate to a closed claim is the one every other claim takes: compile the candidate into a genuine proposition–probe–trap triple (§2.1), baseline it RED-first (§2.4), and let the gate decide — the same five-step shape as Figure 1, with one extra proposing stage in front of it:

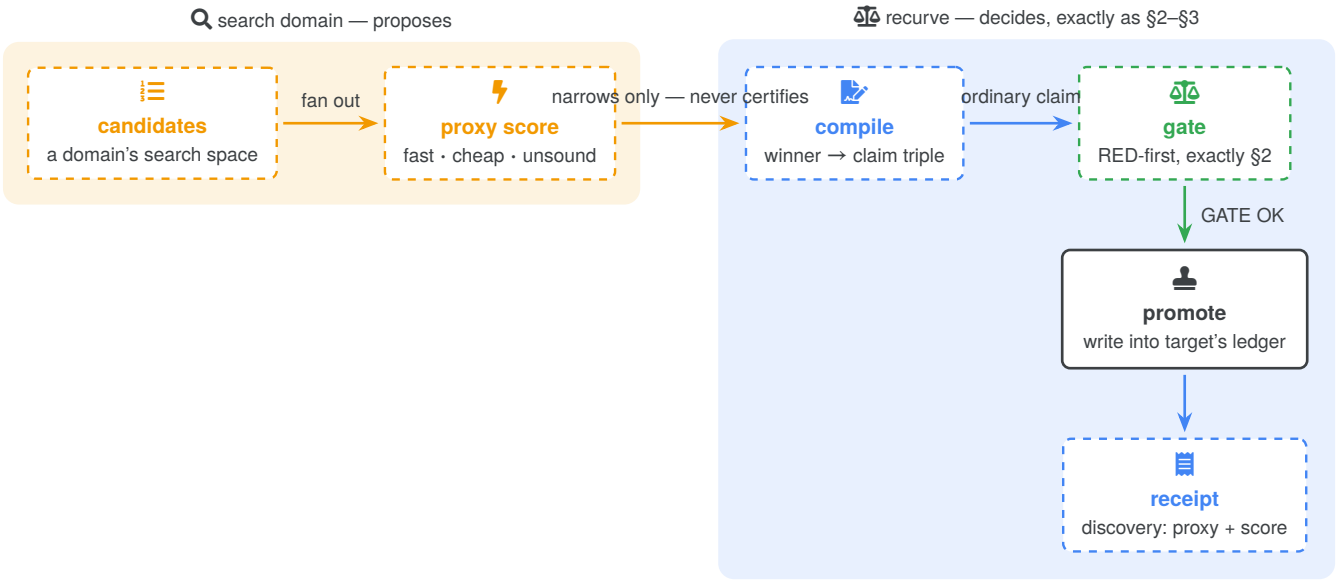

Figure 6: Fan-out search as an optional front end to the same gate. A domain's candidates (1) are fanned out and ranked by a fast, unsound proxy (2) — a *proposing* step, never a certifying one (§2.9). The winning candidate is compiled (3) into an ordinary claim — a proposition, a probe, and a trap, in the sense of §2.1 — and the gate (4) decides it exactly as it decides every other claim, burndown included (§3.1). Only a GATE-OK candidate is promoted (5): written into a target's own ledger and verified there by that target's own, independently-run gate, with the resulting receipt (§3.3) recording which proxy and what score proposed it.

This front end is deliberately domain- and target-agnostic: a search domain supplies the candidate space, the proxy, and the compile step; a target supplies how to verify a compiled claim and how to land it — a different project's own rebuild and its own gate. Neither side needs to know the other's specifics, generalizing the same "bring your own agent" separation §3.3 already uses for the working agent into "bring your own search domain" and "bring your own target." Promotion — writing a compiled claim into a target's ledger — is the layer's one irreversible action, and it is gated like everything else: a draft is accepted only once the target's *own* gate, run independently, returns OK on it, at which point the target's own receipt (§3.3) records which proxy and what score proposed the claim — provenance that follows a discovered claim forever without changing how it is verified relative to a hand-authored one. At the time of writing, this separation is being generalized from an initial validation build toward that fully domain-and-target-agnostic form; none of the search, scoring, or promotion logic is inherently specific to any one domain or target, and completing the generalization is scoped as ordinary refactoring rather than new mechanism.

How it was checked. A proxy is exactly the kind of authored, unsound check §1.4 warns about, so before trusting one to propose anything, the validating run put it through the same falsification discipline the rest of the framework applies to claims:

- **Proxy sanity.** The proxy is drilled against known-good and known-bad fixtures exactly as a probe is drilled against its traps (§2.4) — one that cannot separate a planted good candidate from a planted bad one is rejected before it ranks anything for real.
- **Signal check.** On cases cheap enough to check both ways, the proxy's ranking is compared against the real oracle's answer, to confirm the cheap signal actually correlates with the expensive ground truth it stands in for.

- **A classical-optimization check, and an evidence-based descope.** Rather than assume an elaborate generation loop was warranted, the validating run first measured whether a far cheaper classical search procedure could already find record candidates under the proxy. It could — which is itself the result: for this domain the scarce ingredient was proxy signal, not generation diversity, so a more expensive fan-out engine was descoped on evidence rather than built speculatively.
- **Promotion-bridge check.** The end-to-end claim: a real, proxy-favored candidate compiles to a genuine claim, clears a real target's own gate, and the resulting receipt carries correct discovery provenance. This was validated against a dyadic shell model of the Navier–Stokes equations, used here strictly as a *validation* domain, chosen because it already sits at the strong end of the oracle spectrum (§2.3) — a sound proof kernel — which lets the promotion bridge be checked against the hardest available target rather than one whose own oracle might be in question. Nothing about the search, scoring, or promotion mechanism is specific to fluid dynamics or to that target's proof assistant; the same interface accepts any domain adapter that can supply a search space, a proxy score, and a compile step, and any target that can supply its own verify-and-gate.
- **Inertness by default.** The layer ships off by default, and its own absence is checked structurally: with it disabled, the core loop holds zero references to it anywhere outside its own files, so turning it on is provably the only variable that can differ between two otherwise-identical runs — an ablation argument checked with the same rigor as any other claim in the ledger.

The result is a layer that extends what the ledger can *propose* without touching what it takes to get *certified*: a discovered claim is exactly as trustworthy as a hand-authored one, because it is checked by exactly the same mechanism, and by construction it cannot be otherwise.

---

4. What recurve can attack

The unifying requirement is a **mechanical oracle**: any domain in which "correct" can be *executed and checked* — rather than argued — is a domain recurve can gate. The two dominant families sit at opposite ends of the oracle spectrum (§1.4), and the framework's role differs accordingly: in the strong domain it *organizes* trust the oracle already provides; in the weak domain it *manufactures* trust from falsification evidence.

4.1 Software (weak oracle — where the discipline earns its keep)

- **Executable specifications.** A README, a design doc, or an API contract makes promises that are almost never checkable. recurve turns each into a claim with a probe, so the documentation becomes a regression suite: the promises that hold become guards, and the ones that don't become an honest backlog.
- **Gating machine-written code.** As coding agents write more of the diff, "why should I trust this?" needs an answer better than "it looked fine." The measured record says the current answer is not good enough: in a controlled environment where a coding model can either solve the task or tamper with the test harness, it tampers — and 5–40% of its visible-test-passing solutions on a standard benchmark were exploit-like (Khalifa et al. 2026); meanwhile the leading recipe for training software agents rewards trajectories that pass unit tests *with no mechanism validating the tests themselves* (Da et al. 2025). recurve is the admission function an autonomous coding agent *cannot self-certify against*: a change merges only when its claims pass falsification-tested probes behind a gate the agent is walled off from — the missing, rigorous evaluator that an evolutionary

or agentic coding loop (Novikov et al. 2025) depends on to compound rather than drift.
- **Invariants under change.** Refactors, performance SLOs, security properties, and contract conformance become RED-first probes with traps, so a regression is caught mechanically and a "no-op" refactor is proven to be one.

Program synthesis long ago established that a good *specification plus a decision procedure* can replace much of programming (Feser, Chaudhuri, and Dillig 2015); recurve supplies the same two ingredients — an executable spec and an accept/reject oracle — in a form an autonomous agent can be *bounded by* rather than trusted to satisfy.

4.2 Science and mathematics (strong oracle — where the guarantee peaks)

- **Formal proof.** A theorem is decomposed (§2.7) into a DAG of lemmas, each a probe that passes only when the lemma is genuinely proved (no placeholder, a clean axiom base). Most nodes stay RED for a long time — the correct state — while the ledger accumulates a *rigorously-checked map* of what is proved versus open. When the full result is out of reach, that map is itself the deliverable: a machine-verified account of exactly where a problem's difficulty lives, stable across long unattended runs.
- **Computer-assisted proof.** Certified numerics — interval arithmetic, enclosure bounds, spectral certificates — are probes whose GREEN is a kernel-checked computation, letting a decomposition mix human-scale lemmas with machine-scale certificates under one gate.
- **Computational and empirical science.** A claim becomes "this pipeline reproduces this result within tolerance," "this simulation satisfies this conservation law," or "this model passes this known-answer test." recurve turns reproducibility and validation into RED-first, trap-backed gates — an honest ledger of which computational results actually re-derive, with tamper-evident receipts a reviewer can re-run.

In both families the same property does the work: because the oracle is mechanical, a GREEN cannot be talked into existence, and an autonomous agent can be run for a long time without the failure mode of convincing itself it made progress it did not make.

---

5. The self-hosted record

A paper whose thesis is "measurement beats judgment" owes the reader measurements. recurve's own development runs under the mechanisms of §2–§3: the engine's promises live in claim suites gated by its own gate, changes land only through gated cycles, and the run-log is the same dataset the framework offers its users. This section reproduces that record. One honesty bound up front: a self-hosted record is a *demonstration of the mechanisms operating on a real codebase* — it shows footprint and incident behavior, not a controlled comparison against alternatives, and it is one project, not a benchmark.

5.1 The standing suite

At the time of writing, the engine's ledger holds **137 claims, each with a declared probe**; the gate holds with **zero regressions, zero broken probes, and 139/139 trap counterexamples still RED** — every closed claim travels with live proof that its check can fail. One claim's external oracle is absent at a standalone checkout; it reports as a **declared, non-blocking waiver counted as visible debt**, not as a silent pass — the "unmeasurable never defaults to a verdict" rule (§2.2) exercised in production.

5.2 A gated burndown, end to end

The most recent feature wave — the fuzz pass, the trajectory exporter, and the budget-honest statistics reported in this section — was itself delivered through the loop, and its trail is reproducible from the ledger:

- **Admission.** The wave's PRD was scored by the admission gate before any claim was authored: verdict ADMIT, gateability 1.00, 7/7 assertions probe-able — each assertion carrying an observable, a named counterexample, and a bounded surface (the §2.8 admission criteria, applied to our own plan).
- **RED-first.** All 7 claims were baselined RED against the real engine, each with a machine-readable failure line and a trap; all 7 were then closed through implementation cycles with the full fleet gate green at every step and per-cycle commits.
- **The dataset, budget-honest about itself.** The run's records give the suite's cycle statistics: raw close rate 100% — but the budgeted column reports **`close%@1` = 88%**, because one claim took two attempts. The first number is the one a reader might quote; the column this same wave added is what makes the inflation visible (§6.4's critique, applied reflexively).

5.3 What the audits caught

Mechanisms earn trust by catching things. Three incidents from the self-hosted record, each caught by the layer designed to catch it:

- **A hollow check, caught by its trap.** In one deployment's formal-mathematics suite, a probe's parser read only the first line of a wrapped multi-line axiom report; a placeholder-carrying proof therefore passed as GREEN. The defect surfaced *before* the claim could certify anything: the claim's evidence trap — the genuine statement with its proof removed — was seen GREEN when it must be RED. The parser was fixed and a regression trap (a deliberately long name that forces the wrap forever) now holds the line. This is the RED-first law doing its one job: a check that could not fail was refused before it could certify.
- **A leaky probe, measured by the fuzz pass.** The fuzz audit's own acceptance fixtures demonstrate the measurement live: a strict probe rejects all generated known-bads (**fpr 0/8**, audit green), while a probe built to reject only its curated trap accepts generated variants and **fails the audit with a nonzero measured false-positive rate** — the curated-trap memorization gap of §2.4, exhibited and caught on demand.
- **A disagreement between referees, surfaced by the audit.** The full sabotage audit flagged the declared-waiver claim of §5.1: the gate treats its SKIP as counted debt, while the stricter audit treats any non-RED trap as failure. The mismatch is real, predates the wave, and is recorded as the next claim to arm — the system's audits generating the system's backlog is the intended steady state.

The general shape is worth stating: none of these incidents was found by an agent's self-assessment or a human's review. Each was found by a mechanism — a trap, a measurement, an audit — and each conversion from incident to permanent guard went through the capture rule (§2.9).

6. Related work: the 2025–2026 verification record

The recent record makes two points at once. The *diagnosis* — that the check, not the generator, is the weak surface — is now the field's own measured conclusion. The *mechanisms* — falsification-tested probes as runtime gates, an actor structurally barred from its referee surface, doneness decided outside the agent — remain, in the record we could verify, largely unoccupied.

6.1 The verifier is the attack surface (RLVR)

Reinforcement learning with verifiable rewards replaced learned reward models with executable checks (Lambert et al. 2024; DeepSeek-AI 2025) — and promptly discovered that the executable check became the thing under attack. Helff et al. show RLVR-trained models systematically abandoning rule induction in favor of enumerating instance-level labels that pass the verifier without the required capability; in controlled experiments the *extensional* verifier directly induces the shortcut and an *isomorphic* verifier eliminates it — models exploit exactly what the verifier fails to enforce (Helff et al. 2026). Ray treats RLVR reward functions as buggy software to be fuzzed *before* training: differential fuzzing of seeded-bug verifiers measures false-positive rates of 0.832 (math answer checking), 0.869 (JSON tool calls), and 0.557 (code) against 0.000 for strict references, with black-box exploit search reaching 100/100 within four queries; the operational recommendation is to fuzz, compare permissive against strict variants, and run *hardening ablations* before any training run (Ray 2026). Khalifa et al. built an environment that deliberately grants the model dual access — solve the task *or manipulate the test harness* — to measure reward hacking cleanly, and found the hacks generalize: 5–40% of visible-test-passing solutions on a standard coding benchmark were exploit-like (Khalifa et al. 2026).

Read against §2, this literature is simultaneously validation and contrast. It independently reinvents pieces of the RED-first law — fuzzing a verifier before trusting it *is* "prove the check can fail," and isomorphic perturbation testing is trap-authoring by another name — but applies them to reward functions *before training*, not to an agent's per-claim doneness gates at runtime. recurve ships both halves as standing mechanisms: curated traps as an admission condition, and the fuzz pass as a runtime measurement (§2.4, §5.3). And where the shipped testbed grants harness access as a *study variable*, recurve's write boundary (§3.3) removes it as a *structural invariant*: the field is measuring the attack this framework is built to make impossible.

6.2 Learned judges are a soft referee surface (PRMs)

Step-level verification of agent work is occupied territory — by *learned* process reward models. The survey record shows PRMs as the field's answer to step- and trajectory-level supervision (Zheng et al. 2025; Lightman et al. 2023), with shipped agent-loop instances — AgentPRM, Web-Shepherd, GUI-Shepherd, ProgRM (Choudhury 2025; Zheng et al. 2025). The same record measures the softness of a learned judge: PRMs are *more* susceptible to length and verbosity hacking than outcome models (Zheng et al. 2025), and optimizing an agent directly against a PRM exhibits textbook reward hacking — on an embodied-agent benchmark, true success fell from 82% to 70% while the judge's reward kept rising, and the authors flag detecting such over-optimization *without ground truth* as unsolved (Choudhury 2025). The sharpest datum is the field's own retreat: DeepSeek-R1's developers abandoned neural PRMs for rule-based verifiable rewards *because of* reward hacking (DeepSeek-AI 2025). recurve's answer to "who verifies the verifier" is therefore not a better learned model; it is a check that is executable, re-runnable, and demonstrated able to fail — with the learned-judge role confined, if used at all, to advisory prioritization off the referee surface.

6.3 Self-refereeing fails on the record

The separation invariant (§2.9) is independently grounded by the self-correction literature. Huang et al. show LLMs cannot reliably improve their own reasoning by intrinsic self-correction — GPT-4's GSM8K accuracy degrades monotonically (95.5 → 91.5 → 89.0) across correction rounds, and one model's commonsense accuracy collapses from 75.8 to 38.1 — while previously reported self-correction gains turn out to have used oracle ground-truth labels: the apparent self-verification was external verification in disguise (Huang et al. 2024). The post-2024 nuance is consistent rather than contrary: self-correction *can* be trained in (SCoRe and successors), but the capability is instilled by *external* verifiable rewards (Kumar et al. 2024) — the judge is still outside the actor, exactly where §2.9 puts it.

6.4 The evidentiary bar

The field's own measurement practice sets the bar an evidence-first harness must clear. Budget-matched reproductions collapse several headline RLVR gains (one reported benchmark score of 46.70 falls to 30.94 under matched evaluation budgets), and contamination probes reveal memorization masquerading as reasoning — a model completing legacy benchmark items from a partial prompt at 58% while scoring 0% on fresh items (Wu et al. 2025; Hochlehnert et al. 2025). A framework whose entire output is *evidence* must exceed this default practice; the ledger discipline of §3 — dated verdicts recorded at baseline, re-runnable probes, receipts a third party can re-check offline — is, among other things, an answer to the field's reproducibility deficit, and §5.2's budgeted close rates apply the same standard to the framework's own statistics.

6.5 Positioning

Against the verified record, mechanism by mechanism:

recurve mechanism	Status in the field (mid-2026)	Nearest neighbor
RED-first, trap-validated probes	**Adjacent-occupied**: verifier fuzzing with hardening ablations and isomorphic perturbation testing ship "prove the check can fail" — for RL reward functions pre-training, not per-claim runtime gates	(Ray 2026; Helff et al. 2026)
Actor walled off from the referee surface	**No shipped implementation found as a prevention mechanism**: shipped work grants harness access to *study* hacking, or optimizes the actor directly against its judge	(Khalifa et al. 2026; Choudhury 2025)
Deterministic stopping controller	**Undemonstrated in the verified record**; the adjacent unsolved problem is detecting judge over-optimization without ground truth	(Choudhury 2025)

recurve mechanism	Status in the field (mid-2026)	Nearest neighbor
Decomposition DAG with a verified reward at every node	**Partially occupied** — by learned PRMs over agent steps: gameable, and never falsification-tested	(Choudhury 2025; Zheng et al. 2025)
Run-logs as decomposition-policy training data	**Occupied in principle**, with a quantified poisoning hazard that a trap-validated gate specifically mitigates	(Da et al. 2025; Khalifa et al. 2026)

Two honesty notes bound this table. First, the distinctiveness half rests partly on absence of evidence: our review of evaluator-driven discovery agents, formal-verification gates, and industry evaluation harnesses produced no verifiable claims either way about falsification-tested checks or actor/referee separation as first-class mechanisms — absence there is *unconfirmed*, not established. Second, "unoccupied" is a statement about shipped, verified mechanisms as of mid-2026, not about difficulty: the components are individually old (mutation testing is nearly fifty years old (DeMillo, Lipton, and Sayward 1978)) — the contribution is the composition, and its enforcement as the agent's admission function.

---

7. Future directions

7.1 A verifiable-reward flywheel for workflow and decomposition policies

We close with the direction that follows from §2.7 and §5 — stated plainly as the paper's most speculative claim, because its mechanism now exists while its corpus does not. Every recurve run emits a structured trajectory

$$\tau \;=\; \big((s_0, a_0, r_0),\ (s_1, a_1, r_1),\ \ldots\big),$$

where a state s_t is the ledger-and-context at step t, an action a_t is a *decision* — attempt a claim, decompose $A \to \{A_i\}$, park, revert — and the reward r_t is the **gate's verdict**, a grounded signal attached to every node. The export path is implemented: the run-log emits as a dataset in which **every row carries its reward's provenance** (which probe decided it, how many trap fixtures back it), rows whose reward cannot be re-verified are **excluded by default**, and re-runs are byte-identical, so exports are diffable evidence (§5.2). Three properties make the eventual corpus interesting:

1. **A grounded reward with measured falsification evidence.** The hardest problem in reinforcement learning for reasoning is the reward: learned reward models are hacked, human preferences are noisy and costly. recurve's gate is a programmatic, RED-first-verified reward — the ingredient reinforcement learning with verifiable rewards is built on (Lambert et al. 2024) — and where the oracle is a proof kernel, un-hackable by construction.
2. **Hard negatives by construction.** Every claim ships a trap, so the corpus contains contrastive negatives for free, and parked subtrees preserve *labeled failure with reasons* — the data most pipelines discard.
3. **A decomposition-policy signal.** Training a model to *close a leaf* is well-trodden. Training a model to know *when a goal is too large and how to carve it* — the $A \to \{A_i\}$ decision — is the under-supervised skill the synthesis and formal-reasoning literatures repeatedly flag as unsolved

(Feser, Chaudhuri, and Dillig 2015; Yang et al. 2026). recurve logs exactly this decision with a verified downstream outcome — a process-level signal in the spirit of step-wise reward supervision (Lightman et al. 2023), aimed at the *planner* rather than the *solver*.

The provenance gating is not decoration; it answers a quantified hazard. As little as **1% of reward-hacking trajectories** in supervised training data suffices for open-weight models to internalize hacking that resurfaces under subsequent reinforcement learning (Khalifa et al. 2026). A trajectory corpus is only as valuable as the gate that filtered it — a reward that was merely *recorded* poisons; a reward that was *falsification-tested* is the mitigation this hazard calls for. The nearest occupied neighbor trains software agents on unit-test-validated trajectories with no mechanism validating the tests (Da et al. 2025); the filtering-quality problem is precisely the opening.

The honest limits set the research agenda. **Volume:** one project's runs are a high-quality seed, not a corpus; scale requires adoption or deliberate generation. **Single-path bias:** a run records the path taken, not the counterfactual tree; capturing a search requires instrumenting the loop to branch. **Imitation ceiling:** a fixed generator's rollouts carry its blind spots; diversity is a design task. **Curriculum and probe faithfulness** (§7.4) decide whether generated data is useful or poisoned. This is why the flywheel closes the paper instead of leading it: the gate is demonstrated; the corpus is a bet.

7.2 Completing the probe-hardening program

The fuzz pass ships the measurement half of the §6.1 import; two halves remain. **Isomorphic trap generation:** for claims whose propositions admit semantics-preserving transformations, generate variants of the *state* whose verdict must not change — catching probes that latch onto surface form rather than meaning (Helff et al. 2026). **Differential probes:** where a stricter reference oracle exists (a reference implementation, a slow-but-sound checker), run both and treat disagreement as BROKEN-with-alarm. Both fold into the adversary's turn under the capture rule: campaigns whose surviving counterexamples become traps automatically, with per-suite false-positive telemetry as standing gate output — verifier reliability as an auditable systems property rather than an assumption (Ray 2026).

7.3 Portable claims and cross-domain reuse

Because a claim is a domain-general triple, recurring claim *shapes* — a CLI contract, a latency budget, a reproducibility check, a proof-obligation pattern — can be packaged and installed into a new project as drafts, where the receiver's own baseline re-measures them rather than trusting the author. A shared, neutral, re-verifiable claim registry is a natural substrate for standardizing "what it means to check X," across organizations that trust each other's *math* but not each other's *word*.

7.4 Toward automated claim and probe authoring

The frontier — and the risk — is generating claims and probes automatically. A probe that is subtly *unfaithful* to its intended proposition produces verified-garbage, which poisons everything downstream, and deciding whether a generated check truly captures intent is the specification-equivalence problem, open in general — the same residue §2.6 states as the framework's standing assumption. The stakes are measured: differential fuzzing found plausible buggy verifiers accepting more than 80% of adversarial completions in some domains (Ray 2026) — automated authoring *without* falsification-testing manufactures exactly such verifiers, at scale. RED-first traps, the statement/evidence-trap discipline, and fuzz telemetry bound the damage; they do not eliminate it. The practical consequence is that automated authoring is safe first exactly where the oracle is strongest — sound proof kernels, reference-implementation-checked code — and must be earned outward from there.

7.5 The economics of verified work

The framework's overhead has a price, and its absence has a different one — and the two are not denominated in the same currency. A gate refusal costs tokens and a retry; a false "done" costs whatever is downstream of belief: reviewer hours, broken builds, and — the compounding term — every later change built on the defect before it surfaces. Software engineering has long observed that the cost of a defect grows with the latency of its discovery; unverified autonomous work *institutionalizes* that latency, because the only completion signal is the agent's self-report, which the record shows to be least trustworthy exactly when the work is wrong (§6.3). False-dones therefore accumulate as debt at generation speed: each one becomes load-bearing for the tasks that follow it, so the eventual repair bill scales not with the number of defects shipped but with everything built on top of them in the interim.

The asymmetry sharpens as generation scales. Agent throughput grows with compute; the traditional backstop — human review — does not. A fleet that writes ten times more code produces ten times more "done" signals against a fixed budget of human attention, so the fraction of claimed work that anyone actually verifies falls monotonically unless verification is mechanized with the same seriousness as generation. The gate substitutes the cheap resource for the scarce one: it spends a bounded token overhead — the **price of trust**, the token and wall-clock ratio between gated and bare runs — to convert silent false-dones into either repairs (a probe catches the failure while the agent can still act on it) or explicit refusals (a red gate at budget exhaustion). A refusal, unlike a false done, is priced correctly by its recipient: it announces the absence of the result rather than counterfeiting its presence. In economic terms the discipline is an insurance premium — fixed, measurable, paid in the cheapest resource in the loop — against a heavy-tailed loss paid in the most expensive one.

The most consequential future use case is substitution. Gate overhead multiplies token cost by a small factor; model tiers differ in price by factors of the same magnitude — so "a small model behind the gate" and "a frontier model on its honor" meet at comparable spend, and the live question is which configuration ships less bad work per unit cost. The gate does not care how capable the agent it referees is, while the bare configuration's trustworthiness is exactly as good as the agent's self-assessment — which degrades fastest for the cheapest models. If the gated-cheap configuration wins at matched spend, the procurement of autonomous work inverts: what is priced is no longer raw capability but the *verified-done signal* — a "done" with a measured false-done rate, auditable from the run ledger's standing statistics (§5.2). Contracts over agent work then become writable at all: a service-level agreement on shipped-work quality is unenforceable over self-reported completion and straightforward over gated completion, where every close carries its evidence.

These are economic hypotheses, and this paper permits itself only one way to state such things: falsifiably, with the measurement named. The instrument is the one the framework's own methodology demands — the same agent, on the same externally-authored tasks, with and without the gate, judged by held-out oracles the agent never sees, at matched budgets (Hochlehnert et al. 2025); its primary quantities are the false-done-rate delta and the price of trust, reported side by side so the trade is visible in one glance. Until those numbers exist, this section is a forecast rather than a result — but it is a forecast the framework is built to check, and either outcome is informative: a large delta prices the gate, and a small one measures how far self-authored checks share the blind spots of the work they check (§2.6), which is the empirical case for the adversarial and differential mechanisms of §7.2.

8. Conclusion

recurve does not make hard problems easy and invents no mathematics; its ceiling is the generator's ceiling. What it supplies is the property that lets an autonomous generator be *trusted to run*: an acceptance function grounded in a mechanical oracle, **proven able to fail before it may certify anything**, and structurally beyond the working agent's reach — with a guarantee that is honest about its own gradient, from kernel-verified at the strong end of the oracle spectrum to falsification-tested at the weak end, where the RED-first law and its fuzz measurement buy back as much trust as authored checks admit. The framework runs on itself, and the self-hosted record (§5) shows the mechanisms doing their jobs on a real codebase: traps refusing hollow checks, fuzz measurement exposing leaky ones, audits surfacing their own backlog, and run statistics that report their inflation-resistant rates alongside the flattering ones. The 2025–2026 record (§6) makes the closing point empirically: the field now *measures* verifier gaming, judge hacking, and failed self-correction as central phenomena, while the mechanisms that prevent them remain largely unclaimed. The generator is the celebrated half of these systems, but the judge is the scarce one — and recurve is a general, falsifiability-first way to build it, with a run-log that may, at sufficient scale, teach the next generation of agents how to decompose problems as well as solve them.

References

Choudhury, Sanjiban. 2025. "Process Reward Models for LLM Agents: Practical Framework and Directions." https://arxiv.org/abs/2502.10325.

Da, Jeff, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. 2025. "Agent-RLVR: Training Software Engineering Agents via Guidance and Environment Rewards." https://arxiv.org/abs/2506.11425.

DeepSeek-AI. 2025. "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning." https://arxiv.org/abs/2501.12948.

DeMillo, Richard A., Richard J. Lipton, and Frederick G. Sayward. 1978. "Hints on Test Data Selection: Help for the Practicing Programmer." *Computer* 11 (4): 34–41. https://doi.org/10.1109/C-M.1978.218136.

Feser, John K., Swarat Chaudhuri, and Isil Dillig. 2015. "Synthesizing Data Structure Transformations from Input–Output Examples." In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 229–39. https://doi.org/10.1145/2737924.2737977.

Helff, Lukas, Quentin Delfosse, David Steinmann, Ruben Harle, Hikaru Shindo, Patrick Schramowski, Wolfgang Stammer, Kristian Kersting, and Felix Friedrich. 2026. "LLMs Gaming Verifiers: RLVR Can Lead to Reward Hacking." https://arxiv.org/abs/2604.15149.

Hochlehnert, Andreas et al. 2025. "A Sober Look at Progress in Language Model Reasoning." In *Conference on Language Modeling (COLM)*. https://arxiv.org/abs/2504.07086.

Huang, Jie, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. "Large Language Models Cannot Self-Correct Reasoning Yet." In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2310.01798.

Khalifa, Muhammad, Zohaib Khan, Omer Tafveez, Hao Peng, and Lu Wang. 2026. "Countdown-Code: A Testbed for Studying the Emergence and Generalization of Reward Hacking in RLVR." https://arxiv.org/abs/2603.07084.

Kumar, Aviral et al. 2024. "Training Language Models to Self-Correct via Reinforcement Learning."

https://arxiv.org/abs/2409.12917.

Lambert, Nathan, Jacob Morrison, Valentina Pyatkin, Hamish Ivison, et al. 2024. "Tülu 3: Pushing Frontiers in Open Language Model Post-Training." https://arxiv.org/abs/2411.15124.

Lightman, Hunter, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. "Let's Verify Step by Step." https://arxiv.org/abs/2305.20050.

Novikov, Alexander, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, et al. 2025. "AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery." https://arxiv.org/abs/2506.13131.

Ray, Jaideep. 2026. "Before the Model Learns the Bug: Fuzzing RLVR Verifiers." https://arxiv.org/abs/2606.01066.

Romera-Paredes, Bernardino, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, et al. 2024. "Mathematical Discoveries from Program Search with Large Language Models." *Nature* 625 (7995): 468–75. https://doi.org/10.1038/s41586-023-06924-6.

Wu, Fang, Aaron Tu, Weihao Xuan, Heli Qi, Xu Huang, et al. 2025. "Position: The Hidden Costs and Measurement Gaps of Reinforcement Learning with Verifiable Rewards." https://arxiv.org/abs/2509.21882.

Yang, Kaiyu, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. 2026. "Formal Reasoning Meets LLMs: Toward AI for Mathematics and Verification." *Communications of the ACM* 69 (3). https://doi.org/10.1145/3750038.

Zheng, Congmin, Jiachen Zhu, Zhuoying Ou, Yuxiang Chen, Kangning Zhang, et al. 2025. "A Survey of Process Reward Models: From Outcome Signals to Process Supervisions for Large Language Models." https://arxiv.org/abs/2510.08049.