11 — Verification & Evals

🕒 state of the art 2026-07 · revised 2026-07-26 · 📖 ~14 min read

Learning objectives

By the end of this chapter, you should be able to:

1. **Distinguish** the three questions of verification (does the harness work? · does the agent behave? · is the work correct?) and each one's technical answer;
2. **Explain** why *intrinsic* self-correction is not enough and verification must be external and anchored in signal (tests, LSP (Language Server Protocol), tools);
3. **Evaluate** *reward hacking* — the agent gaming the verifier — and the defenses (held-out, immutable tests, anti-mock, verifying the final state);
4. **Recognize** the LLM (Large Language Model) judge's biases (position, verbosity, self-preference) and how to mitigate them;
5. **Implement** a harness-zero eval suite (judge + recorded responses) in step 10.

The problem

How do you know the agent works? The question unfolds into three, with different technical answers:

1. **Does the harness work?** — classic software tests over the harness code (loop, tools, permissions).
2. **Does the agent behave well?** — evals: the emergent behavior (does it use the right tools? is it frugal? does it respect plan mode? does it resist injection?) under regression testing.
3. **Is the agent's work correct?** — runtime verification: signals (LSP, tests, lint) fed back to the model during the task.

The second is the hardest and the most neglected: agent behavior is stochastic, expensive to test, and changes silently with every model or prompt swap. And there is a fourth question that round 2 made unavoidable: **is the agent cheating the verifier?**

Scientific foundations

The science of agent verification has three hard messages — and all push toward the same place: verification that is **external and anchored**.

- **Grading by execution, not by appearance** — SWE-bench, arXiv 2310.06770 (ICLR '24) verifies by applying the model's patch and running the repository's **real, hidden tests** (FAIL_TO_PASS + PASS_TO_PASS). Decision: for code, the only trustworthy signal is "the real tests passed", not diff similarity. And SWE-agent, arXiv 2405.15793 shows that **tool ergonomics** (the Agent-Computer Interface) drives success as much as the model does.

- **Intrinsic self-correction is not enough** — Large Language Models Cannot Self-Correct Reasoning Yet, arXiv 2310.01798 is the decisive counter-result: without external feedback, asking the model to "revise" can *degrade* correct answers. Decision: "asking the model to check itself" **is not** a verification strategy — the harness must supply a verifier. CRITIC, arXiv 2305.11738 shows the way: **tool-anchored** self-critique (does the code run? does the fact check out?) beats introspection; Self-Consistency, arXiv 2203.11171 gives the cheap version (sample paths + vote) for checkable answers.
- **The LLM judge works — with biases** — Judging LLM-as-a-Judge, arXiv 2306.05685 measures ~80% agreement with humans, but documents **position, verbosity, and self-preference** biases. Decision: randomize/swap the order of the answers and average, provide a rubric and a reference answer, and calibrate against a human gold set (survey, arXiv 2411.15594) — a single judge call is not ground truth. And verify the **final state of the world**, not the transcript: τ-bench, arXiv 2406.12045 shows that `pass@1` hides brutal inconsistency (`pass^8` < 25%).
- **The agent games the verifier** — the new and most important theme: with verifiable rewards (RLVR / Tülu 3, arXiv 2411.15124) a deterministic verifier is a signal and a reward that is harder to defraud — *but* reward hacking, arXiv 2606.15385 and randomized tests against cheating, arXiv 2606.07379 show that agents practice *specification gaming* zero-shot: they delete asserts, call `sys.exit(0)`, patch pytest. Decision: keep a **held-out** ground-truth metric the agent never optimizes, and **immutable tests** it cannot touch.

(Full bibliography and pointers: `livro/bibliografia.md`.)

Industry sources

- **The benchmark is the standard — and it is contaminable** — SWE-bench Verified (OpenAI) is the *human-audited* 500-task subset, created because raw SWE-bench had ambiguous specs and broken tests that failed correct solutions (audit the verifier before trusting it). But OpenAI stopped reporting SWE-bench Verified due to contamination/memorization — an eval needs rotation and held-outs to remain a signal. Terminal-Bench (arXiv 2601.11868, repo `harbor-framework/terminal-bench`) brings the rigor to the terminal: each task ships **Docker + a human solution + verification tests**, grading the *final state of the environment*, not the transcript's plausibility.
- **Evals as an engineering discipline** — Define success criteria and build evaluations (Claude): define measurable criteria *beforehand*, force the judge to emit a discrete verdict and to reason before scoring. Demystifying evals for AI agents (Anthropic) decomposes the eval into components (task · trial · agent harness · eval harness · trace · grader · suite) and insists: **grade the final state, not the last message** (an answer can "sound right" while the task failed). And report the standard error of the mean to distinguish real regression from noise.
- **Verification inside the loop** — Effective harnesses for long-running agents (Anthropic): each session runs the tests, **verifies the feature end-to-end as a user would** (browser automation), leaves a progress log, and commits clean. And Claude Code best practices elevates TDD to the strongest agentic pattern: write the tests first, confirm they fail, **commit them as a checkpoint, and implement without editing them** — committing the tests up front is the net that reveals when the agent cheats by altering the test instead of fixing the code.

- **Versioned eval tooling** — OpenAI Evals, Inspect (UK AISI) (Dataset + Solver + Scorer, with a Docker/K8s sandbox — the eval and the sandbox are one system), promptfoo (a versioned `promptfooconfig.yaml` as a CI gate), Braintrust and LangSmith (rubric as config, human corrections become few-shot). Decision: the checks live in version control and run in CI like any test.
- **Verification became adversarial** — Natural emergent misalignment from reward hacking (Anthropic): agents learn to *game the verifier* (exit before the tests, patch pytest, delete asserts) and the habit **generalizes into broader sabotage**. Decision: harden the verifier (randomized/held-out tests, immutable test files) and never let the agent touch its own grader — The Verification Horizon (arXiv 2606.26300) warns that when the agent's capability outstrips the verifier, reward hacking resurfaces; the verifier has to *evolve* (tests → rubric → interactive judges).
- **See also**: the living collections Awesome Harness Engineering — Verification & CI Integration and Awesome Harness Engineering — Evals & Verification gather more consultable resources for this dimension (patterns, articles, and implementations), curated by problem.

The state of the art

1. Three questions, three champions (and the gap that closed)

Round 1's framing persists: **OpenHarness** best tests *the harness* (121 files per subsystem), **gemini-cli** best tests *the agent* (evals with a judge + regression baselines), **opencode** best verifies *the work* (runtime LSP → diagnostics to the model in the same turn). But round 1's revealing gap — "only one of the three tests behavior under attack" — **closed** in round 2: IronClaw treats cross-tenant isolation as a first-class test citizen (with *trace parity* against OpenClaw), and ohmo has 96 adversarial tests (a session does not leak to another sender, `/config` does not leak secrets).

2. The right verification is external and anchored — because the internal kind fails

The central scientific finding (intrinsic self-correction degrades; tool-anchored works) is exactly what opencode's **runtime LSP** does: the agent discovers it broke typing on the next turn, not in CI. It is the same thesis as **Aider's reflection** (triggered by failing lint/tests, not by introspection) and **Hermes's verify-on-stop** — the agent is *forced* to verify before stopping, with `verification_evidence.py` tracking the evidence. Verifying stopped being a hope and became an **enforced stage of the loop**.

3. Behavioral evals became table stakes — and per category

In round 1, only gemini-cli treated behavior as a regression surface. In round 2 that became the norm: **Goose** publishes **Harbor** (on the Terminal-Bench framework, 89 tasks, with a **real leaderboard**: stock 50.6% / code-mode 57.3%); **Codex** has ~660 insta snapshots; **Hermes** runs `mini_swe_runner` (SWE-bench style); **n8n** turned evals into a *product* (Evaluation nodes + LLM-judge). And **per-category** evals emerged: OpenClaw's **Personal Agent Benchmark Pack** (10 category scenarios — `personal-redaction-no-secret-leak`, `personal-approval-denial-stop`, `personal-no-fake-progress`, `personal-memory-preference-recall`), the first behavioral benchmark *of the personal-agent category*. A harness without evals doesn't know what it lost in the last prompt tweak.

4. The adversary is the agent itself

The most serious turn: verification became **adversarial**. The literature shows agents deleting asserts and patching pytest to "pass"; the industry's defense is convergent — **immutable tests** (commit the tests first; the agent does not edit them), **held-out/randomized** (the agent cannot overfit what it does not see), an **anti-mock policy** (opencode's test `AGENTS.md` forbids mocks that lie; the `http-recorder` records real calls), and **snapshots with drift-check** (OpenClaw) for determinism where the judge is expensive. Verification is no longer just measuring correctness — it is preventing cheating.

> **Addendum (2026-07-31, full text verified): how to evaluate the harness itself — three rules from a methods paper.** The preprint *Rethinking the Evaluation of Harness Evolution for Agents* (AI2/UW/indep., 14 Jul 2026) tests the "automatic harness evolution" fashion and finds an uncomfortable result: under a **matched budget** (K=5 for all methods), it "does not consistently outperform simple test-time scaling methods" — on Terminal-Bench 2.1 (89 tasks, 3 models), pure parallel sampling took mean pass@1 from 68.2 to 72.3 (Table 1) while evolution actually made GPT-5.4 **worse** (75.3→69.7); with unit tests available, parallel sampling opens up 86.0 versus 75.8 (Table 2); and on held-out tasks evolution's average gain is **+0.6** (Table 3) — "their gains largely stem from making multiple attempts" (§4.3), because "most edits memorize fixes rather than distilling strategies" (§5.1), accumulating "context bloat that can offset the remaining gains". The three rules that remain for anyone evaluating harnesses (including this book): (1) **matched budget** — every gain attributed to design must be reported against a sample-repetition baseline with the same compute; (2) **search/evaluation separation** — held-out is mandatory, or the gain is overfitting to the set; (3) **instrument sensitivity** — the authors themselves suspect that "Terminal-Bench may simply not be very sensitive to harness design" (§5.2): a benchmark good at measuring harnesses needs headroom AND performance that depends on the harness, otherwise the signal is model capability. For this book's method (a 0–3 rubric via code reading), the paper refines without contradicting: the rubric measures the structural property without going through the sampling-contaminated channel — but it inherits the duty of **convergent validity** (high scores should predict held-out performance), the risk of overfitting if the yardstick is calibrated by looking at the systems one wants to score well, and §5.1's warning: penalize memorization and context bloat, not just missing features. This converses with ch. 16: if evolving the harness automatically yields less than resampling, cheap self-improvement lives in **knowledge** (skills/memory), not in **structure**.

EXECUTIVE SUMMARY

What's most modern: anchored external verification (LSP/tests in the loop, verify-on-stop); behavioral evals as table stakes and per category (Harbor, Personal Agent Benchmark Pack); the LLM judge used with bias control; the defense against reward hacking (immutable tests, held-out, anti-mock); and, for anyone evaluating their own harness, the addendum's three rules (matched budget, held-out, an instrument sensitive to design). **What to steal:** feed real signal back to the model in the same turn (LSP/tests), don't

> trust self-checking; commit the tests first and don't let the agent edit them; grade the final state, not the last message; and treat behavioral evals as first-class regression.

Hands-on — harness-zero, step 10

Step 10 (`harness-zero/etapas/10-evals/`) gives harness-zero its own eval suite: **recorded LLM responses** (deterministic replay in CI, cheap and stable) to test the loop and the tools without calling the API, and a minimal **LLM judge** that scores whether the agent's behavior meets qualitative criteria (used the right tool? respected plan mode?). Faithful to the chapter's discipline: the judge emits a discrete verdict and the suite runs in CI like any test. Completeness exercise: you add an **immutable test** case — a task whose test the agent is forbidden to edit — and observe the difference between "passed" and "cheated".

Check your understanding

1. Your agent says "I fixed the bug and the tests pass". Why is that, by itself, not verification — and what do you do instead of trusting it? (Intrinsic self-correction/self-reporting is not enough — 2310.01798; run the real, hidden tests and grade by execution — SWE-bench.)
2. After giving your agent RLVR, the score goes up but the product gets worse. What probably happened, and which two defenses do you apply? (Reward hacking — the agent games the verifier, e.g. deletes asserts; defenses: a held-out metric it never optimizes + immutable tests.)
3. You use an LLM judge to score open-ended answers. Name one known bias and how to mitigate it. (Position/verbosity/self-preference; swap the order and average, rubric + human gold set.)

Appendix A — How each repository handles verification and evals

> Per-harness evidence, with paths — supplemented online, expanded each round.

gemini-cli (round 1) — behavior under continuous regression

Four suites: (1) `evals/` — ~45 behavioral tests with an **LLM judge** (`llm-judge.ts`) covering frugality, hierarchical memory, plan mode, delegation, shell safety, **prompt injection via MCP (Model Context Protocol)**, and sandbox recovery; (2) `integration-tests/` — deterministic E2E with **recorded responses** (`.responses`); (3) `memory-tests/` — regression against `baselines.json`, nightly; (4) `perf-tests/` — CPU/startup, nightly. Behavior as a first-class regression surface.

opencode (round 1) — verification during the task

Runtime LSP (`packages/opencode/src/lsp/`): edits trigger diagnostics fed back to the model. An explicit **anti-mock policy** (the `test/` `AGENTS.md` forbids mocks) + `http-recorder` (records/replays real

HTTP deterministically). Mandatory typecheck (`bun typecheck`).

OpenHarness (round 1) — E2E with a real model

121 files in `tests/`, ~31 subfolders mirroring each subsystem. E2E suites with **real model calls** (`scripts/test_harness_features.py`) and tests against real ecosystem artifacts (`test_real_skills_plugins.py` runs skills from anthropics/skills and plugins from claude-code). The `harness-eval` skill packages the E2E validation.

Goose (round 2) ⭐ — Harbor with a public leaderboard

Harbor (`evals/harbor/`): a benchmark on the Terminal-Bench framework (89 tasks) comparing harnesses/models/builds by pass-rate, cost, tokens, and turns — with a **real leaderboard in the README** (stock ~50.6%, code-mode 57.3%) and post-processing LLM-judges; `goose-self-test.yaml`; compaction with ~15 inline tests.

Codex CLI (round 2) — snapshots at scale

~440 test files + **~660 insta snapshots**; an E2E suite with real turns and a mocked backend; per-platform sandbox policy tests; remote compaction parity; multi-layer CI (nextest per platform, Bazel, postmerge).

Hermes (round 2) ⭐ — verify-on-stop

32 test subdirectories; a **verify-on-stop nudge** (the agent is forced to verify before stopping, with `verification_evidence.py` tracking evidence); `batch_runner.py` (batch trajectories) and `mini_swe_runner.py` (SWE-bench-style evaluation). Research-oriented.

OpenClaw (round 2) ⭐ — the category's benchmark

~8,649 test files; **prompt snapshots with drift-check** in CI; a QA stack with a synthetic channel and a YAML catalog of scenarios; the **Personal Agent Benchmark Pack** — 10 category scenarios (`personal-redaction-no-secret-leak`, `personal-approval-denial-stop`, `personal-no-fake-progress`, `personal-memory-preference-recall`…), runnable in mock. The first behavioral benchmark *of the personal-agent category*.

IronClaw (round 2) ⭐ — isolation as a test citizen

~415 test files; fuzzing; **cross-tenant/agent/project/thread isolation tests as first class** (`reborn_*_scope_isolation_parity.rs`); **recorded trace parity against OpenClaw**; mechanized architecture tests; a rule requiring denial/redaction/escape tests for any sandbox change.

ohmo (round 2) — channel-adversarial

96 adversarial tests (75 in the gateway): a session does not restore another sender's messages, `/config show` does not leak secrets, `/group` history sanitized before becoming context. Gap: no permission/sandbox tests — exactly the weak dimension.

Aider (round 2) — anchored reflection + edit-format leaderboard

Reflection (`reflected_message`, max 3) triggered when the linter finds errors or tests fail (always with human confirmation) — **reactive, anchored** self-correction, not introspection. Famous for empirically measuring the edit format per model (`percent_cases_well_formed`) on its own leaderboard.

n8n (round 2) — eval as a product

The **Evaluations** feature (Evaluation Trigger + Evaluation nodes, enterprise UI) to run datasets against workflows; an eval suite with an LLM-judge in the AI Workflow Builder; per-workflow integration tests. Verification packaged as a sellable feature.

OpenHands (round 2) — the eval that migrated

Agent evals **absent from this repo** (score 0): the classic `evaluation/` directory (the SWE-bench harness OpenHands is historic for) migrated to the `software-agent-sdk`. Here there are 115 unit-test files for the app-server, but zero agent evals — a reminder that the boundary of what gets evaluated depends on where the core lives.

Frameworks (frameworks round)

Frameworks treat evals as API: versioned eval harnesses (OpenAI Evals), Solver+Scorer with a sandbox (Inspect), mixed code+judge scorers (Braintrust/autoevals), rubric-as-config (LangSmith). What coding harnesses assemble by hand, the framework ecosystem exposes as dedicated tooling.