SAMPLE — CHAPTER 3

# The Improvement Loop

Agents that get better in production

Datt Goswami

agenticfrontier.dev · v2026.07 · sample

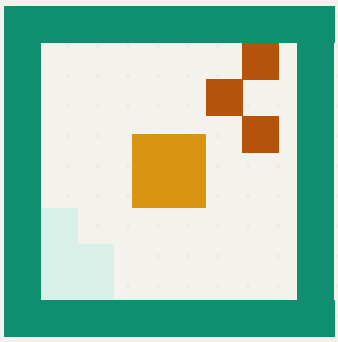
# Contents

The full book: 12 chapters in 4 parts, plus front matter and appendices. This sample carries Chapter 3 in full.

CHAPTER 3 · PART II · THE SUBSTRATES · SAMPLE

# Harnesses That Build Themselves

Harness design is model-specific and iterable, so it is becoming something agents do to themselves — weakness mining in, human authorship out.



## §1 · The harness engineer does not scale

Every agent team owns a file nobody is proud of: the harness config that made one model behave. It took a week of trace-reading to write, it works for exactly the model it was tuned on, and the next model release will invalidate half of it. The uncomfortable structural fact was stated plainly by a Shanghai AI Laboratory team this June: agent performance is jointly shaped by base model and harness, different models exhibit distinct behaviors, so effective harness design is *inherently model-specific* — and yet harnesses are still largely engineered by human experts, "a paradigm that scales poorly as modern LLMs become increasingly diverse and rapidly evolving" (Zhang et al., 2026).

Model-specific work that must be redone per model, against a model-release cadence measured in weeks, is a job description with an expiry date. Not because the work is trivial — the manual craft is real and measurably strong. The Allen Institute's TMax line makes that point from the other side: a deliberately

*simple* recipe and harness, with careful RL training, produces terminal agents that dominate the open-recipe Pareto frontier below 32B parameters on Terminal-Bench 2.0 (Ivison, Yin, et al., 2026). Human-designed simplicity remains the baseline to beat — which is exactly what makes the mid-2026 crop of harness-automation papers worth reading closely: they are starting to beat it, from several directions at once.

> [← B1] **The Harness Is the Product** defined harness anatomy and made the case that the harness term carries the steeper capability gradient. This article is about who — or what — climbs that gradient now. The human-run version of this loop, receipts included, is [← B13] **Closing the Loop**; today's subject is the version that runs without the human.

KEY TAKEAWAY 1

Harness design is model-specific by nature and human-authored by habit — a combination that cannot survive the current model-release cadence. The strongest manual baselines (simple recipes, carefully trained) still win benchmarks, but the automation wave is now measured against exactly those baselines.

## §2 · The Self-Harness loop

The paradigm paper of the wave is Self-Harness (Zhang et al., 2026): an LLM-based agent improves *its own operating harness*, with no human engineers and no stronger external agent in the loop. The construction is an iterative three-stage cycle (see Figure 3.2):

- **Weakness Mining** — identify model-specific failure patterns from execution traces;
- **Harness Proposal** — generate diverse yet minimal harness modifications tied to those failures;

- **Proposal Validation** — accept a candidate edit only after regression testing.

Each stage carries a discipline the naive version of this idea lacks. The mining stage starts from *traces*, not intuitions — the failure pattern must appear in execution data. The proposal stage is constrained to minimal edits tied to mined failures — no wholesale rewrites, no generic instruction-stuffing. And the validation stage is fail-closed: an edit that does not survive regression testing does not ship. The authors' qualitative analyses stress the first point — the loop "does not simply add generic instructions" but turns model-specific weaknesses into concrete, executable harness changes.

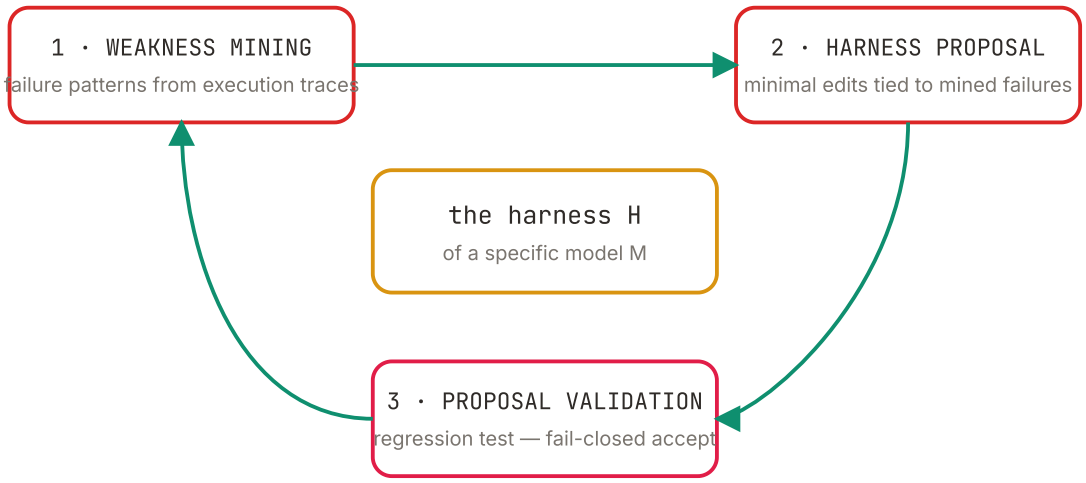



Figure 3.2: The Self-Harness loop of Zhang et al. (2026). Traces in, minimal tied edits proposed, regression-tested edits out — a fail-closed improvement cycle whose update surface is the harness itself.

The instantiation is the credibility test. On Terminal-Bench-2.0, starting from a deliberately minimal harness, Self-Harness ran for three base models from different families — MiniMax M2.5, Qwen3.5-35B-A3B, and GLM-5 — and improved held-out pass rates for all three: 40.5% to 61.9%, 23.8% to 38.1%, and 42.9% to 57.1%, respectively (see Figure 3.3). Two properties of that result matter more than its size. It is *held-out* — the loop's regression discipline generalized past the tasks it mined. And it is *per-model* — three different harnesses emerged

for three different models, which is the whole argument for automating the job: the artifact that works is not one harness but a harness-per-model, and only a loop produces those at model-release cadence.

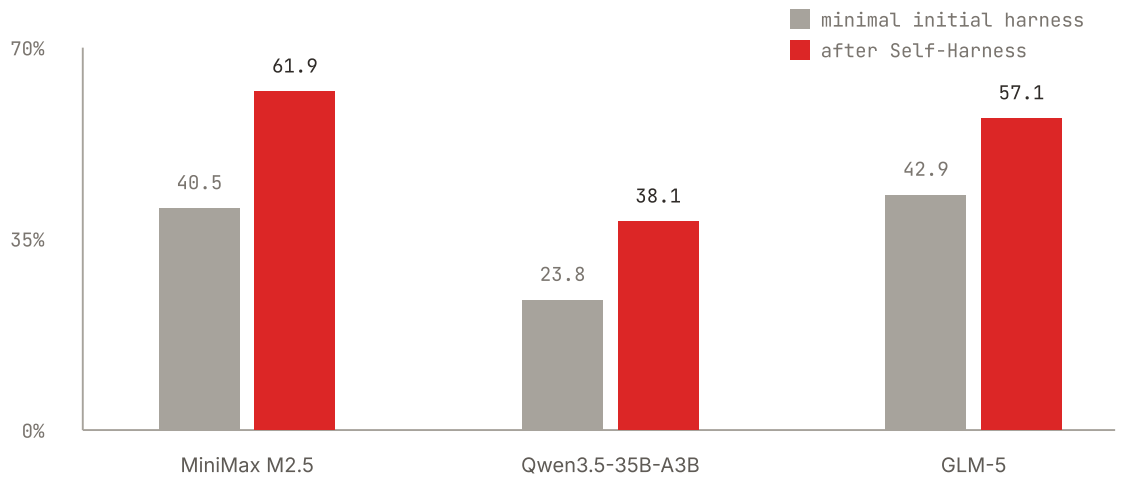



Figure 3.3: Held-out pass rates on Terminal-Bench-2.0 before and after Self-Harness, per base model (values from Zhang et al., 2026). Three models, three different mined harnesses, one loop.

KEY TAKEAWAY 2

Self-Harness operationalizes harness self-improvement as Weakness Mining → Harness Proposal → Proposal Validation, fail-closed at the validation stage. On held-out Terminal-Bench-2.0 tasks it lifted three different base models — evidence that the loop, not any single harness artifact, is the thing worth owning.

## §3 · The automation wave

Self-Harness is the purest statement of the paradigm, but the wave is wider, and each entry automates a different slice of the harness engineer's week.

**Compiling the harness from a document.** HORIZON treats the harness itself as a build artifact: a Markdown harness is compiled into a project pack — domain knowledge, an executable evaluator, an acceptance predicate, a git and runtime policy — and a hands-free loop evolves repository-level hardware designs against it (Yu et al., 2026). The relocation of authorship is the point: the human writes a *specification document*; the compilation step, the state management, and the improvement loop are mechanical. Where D1 read HORIZON as a loop instance, read it here as a statement about interface: the harness engineer's deliverable moved one level up the stack.

**Optimizing the pipeline in place.** FAPO — from Cisco's Foundation AI group — hands a working multi-step LLM pipeline to an optimizing agent (concretely, Claude Code operating inside a standardized codebase) and lets it evaluate, inspect intermediate steps, diagnose failures, propose scoped changes, and validate variants against a score function (Kassianik, Saglam, et al., 2026). The design is conservative in exactly the way §2's validation stage is: prompt edits first, and only when attribution identifies a structural bottleneck does it modify chain structure within a permitted scope. The numbers are the strongest in this article's corpus for pipeline-level automation: across six benchmarks and three task models, FAPO beats the GEPA baseline in 15 of 18 model–benchmark comparisons, with a mean gain of +14.1 pp — and in the six HoVer and IFBench comparisons where the prompt-first search escalated to structural changes, it wins all six with a mean gain of +33.8 pp. The escalation result deserves the emphasis: the largest wins came precisely when the optimizer was allowed to touch the harness's *structure*, not just its words.

**Meta-optimizing the builder.** Autodata closes the recursion: the agent that builds training and evaluation data is itself trained to be better at building it, and the meta-optimized builder delivers a larger uplift than the base method (Kulikov et al., 2026). One rung of self-reference, applied to the tooling layer — the same shape Self-Harness applies to the harness and FAPO applies to the pipeline.

KEY TAKEAWAY 3

The wave automates the harness engineer's job at three interfaces: specification (write a document, compile the harness), optimization (an agent edits the live pipeline — biggest gains when allowed to change structure, +33.8 pp mean in FAPO's escalation cases), and meta-level (the builder itself gets trained). The common discipline is scoped edits validated against an explicit score.

## §4 · Pieces that route and schedule themselves

Below the whole-harness loops, individual harness components are acquiring the same property.

**Routing.** Agent-as-a-Router starts from a diagnosis: static routers underperform because of *information deficit*, not model weakness — simply augmenting a vanilla LLM router with task-dimension performance statistics yields a 15.3% relative gain, beating a heuristic router built on the same priors (Zhou et al., 2026). Their ACRouter therefore reframes routing as a Context→Action→Feedback→Context loop that accumulates execution-grounded experience *during deployment*. The routing table stops being configuration and becomes memory — a harness component that learns its own policy from the traffic it routes.

**Scheduling against a compiler.** COMPILOT wires an off-the-shelf LLM, zero-shot, into a closed loop with a compiler: the model proposes loop-nest transformations, the compiler reports back legality and measured speedup or slowdown, and the model refines its strategy on that concrete feedback (Merouani et al., 2025). Across the PolyBench suite this yields geometric-mean speedups of 2.66× in a single run and 3.54× best-of-5, competitive with — and in many cases ahead of

— the state-of-the-art Pluto polyhedral optimizer. It is the smallest, cleanest demonstration in the corpus that the loop pattern needs no fine-tuning at all when the environment already supplies a verifier: the compiler's legality check plus a stopwatch is a complete Proposal-Validation stage that existed all along.

KEY TAKEAWAY 4

Harness components self-improve where their feedback is native: a router that banks execution statistics gains 15.3% relative over its static self, and a zero-shot LLM against a compiler's legality-plus-stopwatch verifier reaches 2.66–3.54× geomean speedups. Wherever a component's environment already verifies, the loop is nearly free.

## §5 · Skills meet harnesses

One boundary keeps this article honest. When the thing that accumulates is *procedural knowledge* — reusable techniques, organized and reloaded across tasks — the improvement has left the harness and entered the skill store Σ. HASTE marks the border case: a hierarchical multi-agent system whose orchestrator coordinates domain specialists and promotes learned knowledge between global, domain, and competition-specific tiers, so that later ML-engineering tasks start warm instead of cold (Kim et al., 2026). The harness contribution is the tiered *loading* discipline — their ablation shows scoped loading matters as much as having skills at all. What the skills themselves earn — the medal rates, the transfer economics, the warm-start arithmetic — is the next article's subject, and D4 owns those numbers.

[→ Chapter 4] **Skills Are Compound Interest** treats skills as capability capital: acquisition cost, transfer yield, depreciation under model churn. The harness loads the portfolio; the portfolio is its own story.

KEY TAKEAWAY 5

Where the accumulating artifact is procedural knowledge rather than harness structure, the loop belongs to the skill store — a different substrate with different economics. The harness's role there is disciplined loading, and the border runs exactly through systems like HASTE.

## §6 · The self-harness discipline

The papers in this article agree on more than a direction; they agree on a shape. Every working harness loop is fail-closed somewhere — Self-Harness at regression testing, FAPO at score-validated scoped edits, HORIZON at an executable acceptance predicate, COMPILOT at compiler legality. None of them ships an edit because it looks plausible. That is the discipline to copy, and it is also where the human re-enters: not as the author of each edit, but as the owner of the gate the edits must pass.

The failure modes this loop must guard are inherited from the series frame. A harness that edits itself against a static acceptance test will eventually optimize the test — the co-evolution problem [→ Chapter 10]. And a harness loop that harvests its own successful traces for the next round of mining inherits the collapse mechanics of self-training [→ Chapter 7]. The guardrail budget goes to the validation stage, because that is the only stage with veto power.

LOOP CARD · THE HARNESS SELF-MODIFICATION LOOP

**SIGNAL** — execution traces of the current harness-model pair, mined for model-specific failure patterns (Zhang et al., 2026).
**UPDATE** — the harness H: minimal, failure-tied edits (prompt first, structure only on attributed bottleneck, per Kassianik, Saglam, et al., 2026).
**GUARDRAIL** — fail-closed validation: regression tests on held-out tasks plus a human sign-off gate on any structural change; an edit without a passing regression run does not merge.
**CHECK** — held-out task pass rate per model, re-measured after every accepted edit batch; the loop is working if held-out (not mined-set) performance rises across model swaps.

KEY TAKEAWAY 6

Automate the authorship, keep the veto. Every credible harness loop in the 2026 corpus is fail-closed at validation — that stage, plus a human sign-off on structural edits, is where a production team should spend its entire caution budget.

## What comes next

The harness loop of this article produces edits — routing policies, context rules, structural changes. But the most valuable thing an agent accumulates over many tasks is not configuration; it is *competence*: procedures that worked, abstracted and banked for reuse. That store compounds like capital, survives model swaps like capital, and depreciates like capital. The balance sheet is next.

# References


1. Zhang, H., Zhang, S., Li, K., Zhang, C., Chen, Y., Zhang, Y., Bai, L., & Hu, S. (2026). **Self-Harness: Harnesses That Improve Themselves.** *arXiv preprint*. arXiv:2606.09498.

2. Ivison, H., Yin, J. O., Shao, R., Xiao, T., Lambert, N., & Hajishirzi, H. (2026). **TMax: A Simple Recipe for Terminal Agents.** *arXiv preprint*. arXiv:2606.23321.

3. Yu, C., Deng, C., Pinckney, N., & Khailany, B. (2026). **Agentic Hardware Design as Repository-Level Code Evolution.** *arXiv preprint*. arXiv:2606.28279.

4. Kassianik, P., Saglam, B., Zhao, H., Nelson, B., Vijay, S., Priyanshu, A., & Karbasi, A. (2026). **FAPO: Fully Automated Prompt Optimization of Multi-Step LLM Pipelines.** *arXiv preprint*. arXiv:2606.19605.

5. Kulikov, I., Whitehouse, C., Wu, T., Nie, Y., et al. (2026). **Autodata: An Agentic Data Scientist to Create High Quality Synthetic Data.** *arXiv preprint*. arXiv:2606.25996.

6. Zhou, P., Tang, Z., Ma, Y., Tang, J., Han, Y., Wan, Z., Meng, F., Wang, W., Zhuang, B., Zhao, W., & You, Y. (2026). **Agent-as-a-Router: Agentic Model Routing for Coding Tasks.** *arXiv preprint*. arXiv:2606.22902.

7. Merouani, M., Kara Bernou, I., & Baghdadi, R. (2025). **Agentic Auto-Scheduling: An Experimental Study of LLM-Guided Loop Optimization.** *Proceedings of the 34th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. arXiv:2511.00592.

8. Kim, Y., Talebirad, Y., & Zaïane, O. R. (2026). **Why Solve It Twice? Hierarchical Accumulation of Skills for Transfer-Efficient ML Engineering.** *arXiv preprint*. arXiv:2606.30911.

GET THE FULL BOOK

# The Improvement Loop

Shipping an agent is the start of the job, not the end of it. The uncomfortable question comes after: is it any better in month three than it was on day one — and if it is, what exactly improved? The model didn't retrain. Something else moved. This book is about that something else.

Twelve essays on the improvement loop — the engineered feedback cycle that makes deployed agents better with use. Part I draws the frame: proof that a self-improving stack already exists, and the line where automation ends and agency begins. Part II walks the substrates that actually improve — harnesses that build themselves, skills as compound interest, memory you can train, and the world-model turn. Part III is the loops themselves, with their failure modes: training on your own exhaust, the verification ceiling, judges on trial, and the Red Queen dynamic where evaluator and evaluated co-evolve. Part IV goes beneath: the serving loop that turns inference-time compute into capability, and the paradigm bet waiting after autoregression.

It is the operational sequel to *Agentic Engineering*: that volume is about building agents that ship; this one is about what happens to them after they ship. The two cross-reference each other, and each stands alone.

Every claim was traced to its source before publication. It is a **living book**: versioned like software, and every future edition is included with your copy.

The purchase includes the book as **PDF and EPUB** and **every future version**. Find it via **agenticfrontier.dev/book**.