# HARNESS-ZERO: HARNESS DISTILLATION VIA AGENT-AS-HARNESS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Agent harnesses, the external systems that mediate model-environment interaction, can substantially improve agent performance, but their gains remain tied to the harness at deployment. Because the best harness varies across domains, instances, and models, a general-purpose agent must either settle for a suboptimal shared harness or route among an ever-growing set of specialized ones. We therefore study *agent harness distillation*: using a domain- or instance-optimized harness as training-time guidance and transferring the behaviors it induces into model weights, so that its gains survive under a single fixed target harness. The challenge is that the two harnesses differ in action space and available information, so guidance from the optimized harness cannot serve directly as supervision for the target one. We introduce HARNESS-ZERO, which enables harness distillation through *agent-as-harness*. Guided by the optimized harness, a *harnessing agent* corrects student responses before execution in the target harness's action space, turning harness guidance into training demonstrations. Fine-tuning on the resulting trajectories internalizes harness-induced behavior into the model, so the specialized harness can be removed at deployment. Our experiments spanning knowledge work, tool use, and science domains show that: ❶ For frontier LLMs using the same evolved harness, agent-as-harness outperforms code-as-harness. ❷ With the specialized harness removed at deployment, HARNESS-ZERO improves the base model's macro-average task success from 23.3% to 44.3%, even exceeding the 41.7% it reaches with that harness still attached. ❸ HARNESS-ZERO recovers harness-induced behaviors absent from the base model, with 82.3% average recovery across 28 patterns in the three domains.

## 1 INTRODUCTION

An LLM agent's capabilities depend on both its model and its ***harness***: the external system that organizes tool use, manages context and state, and controls interaction with the environment (Weng, 2026; Ning et al., 2026b). Harness engineering has become a central lever for improving agent performance. Coding-agent harnesses combine shell access, file systems for persistent memory, subagents, and background jobs (Weng, 2026; Yang et al., 2024). Research-agent harnesses organize workflows for hypothesis generation, experimentation, and evidence collection (Lu et al., 2026). Context management and experience reuse further support continual learning and long-horizon execution (Ye et al., 2026; Ma et al., 2026; Karten et al., 2026b;a; Yan et al., 2026; Ye et al., 2024). Recent methods such as Meta-Harness automate this engineering process by optimizing harness code (Lee et al., 2026; Lin et al., 2026a; Zhang et al., 2026b).

Harness optimization, however, improves the agent's external scaffolding rather than the model itself, so its gains remain tied to that harness at deployment. Because the best harness varies across domains, instances, and base models (Zhang et al., 2026b; Luo et al., 2026b; Liu, 2026), a general-purpose agent must choose between a shared harness and a collection of specialized ones. A shared harness forgoes some specialized gains (Luo et al., 2026b; Liu, 2026; Yao et al., 2026). Maintaining many harnesses instead requires routing and incurs recurring costs in context, model calls, tool calls, and orchestration (Zhang et al., 2026c; Lin et al., 2026b; Chang et al., 2026; Zhang et al., 2025a). Neither choice moves the discovered harness improvements into the model.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

**HARNESS-ZERO**

To address these limitations, we introduce **HARNESS-ZERO**, a framework for ***agent harness distillation***. It uses an optimized harness as training-time guidance to distill its induced behaviors into model parameters. This turns gains discovered through harness optimization into model capabilities available under a fixed target harness.

The distillation target in HARNESS-ZERO spans specialized tool use expressed in the student's native action space, behavioral patterns enforced by middleware code, and accumulated knowledge and reusable experience supplied by skills and memory. The source and target harnesses can differ in both action space and available information, making trajectories collected under the source harness unsuitable for direct imitation under the target harness.

Our solution to this challenge is ***agent-as-harness***, which wraps the student agent with a ***harnessing agent*** at its response boundary. We denote the student's fixed operating harness as the target harness $h$, the evolved student-side harness (via, e.g., meta-harness (Lee et al., 2026)) as $h^\star$, and its adaptation for the harnessing agent as the private reference harness $\mathcal{K}$. As one example of this adaptation, a middleware rule in $h^\star$ that blocks risky actions becomes a review-time warning in $\mathcal{K}$, activated when the student proposes such an action.

During data collection, the harnessing agent uses $\mathcal{K}$ to review each proposed response before it is executed or added to the trajectory. It passes sound proposals unchanged and otherwise makes the smallest coherent correction, expressed as a complete response in the student's action space. The accepted response is executed through $h$, and the resulting observation is appended to the student's trajectory. The harnessing agent cannot inspect the student sandbox or consult hidden task solutions, and its private review discussion remains outside the student-visible trajectory. HARNESS-ZERO applies supervised fine-tuning (SFT) to these reviewed rollouts to internalize the demonstrated behaviors in model parameters.

**Agent-as-harness**

Agent-as-harness builds an adaptive mapping between harnesses: an agent reads the guidance of one harness and re-expresses it, response by response, as executable corrections in the action space of another. Any optimized harness can therefore supply training demonstrations for the target harness, regardless of how the two differ in actions or available information.

We evaluate HARNESS-ZERO across three domains: spreadsheet-based knowledge work (SpreadsheetBench Verified), multi-application tool use (AppWorld), and scientific reasoning (USPTO Retrosynthesis). We instantiate $h$ as a fixed, minimal mini-SWE-agent (SWE-agent Team, 2025) with one Bash `execute` tool. We first evaluate agent-as-harness on frontier models without training and find that it outperforms code-as-harness (81.1% vs. 78.1%, averaged across six benchmark–model settings). We then evaluate harness distillation on Qwen3.5-9B and remove $h^\star$, $\mathcal{K}$, and the harnessing agent at deployment. Under $h$ alone, HARNESS-ZERO raises macro-average performance from 23.3% to 44.3%, exceeding the 41.7% obtained by the base model with $h^\star$ still attached. Controlled ablations further show that harness-guided review produces substantially better distilled performance than alternative supervision sources, including direct trajectories from a stronger model, trajectories generated under $h^\star$, review without $\mathcal{K}$, and review given only the task answer (30% vs. 3–15%). Behavioral analysis also finds that the distilled model recovers most behaviors induced by $h^\star$ but absent from the base model (82.3% recovery on average across 28 patterns).

**Contributions.** ❶ We formulate *agent harness distillation* and present HARNESS-ZERO, which transfers behaviors induced by an optimized harness into model parameters for deployment under a fixed target harness. ❷ We introduce *agent-as-harness*, which translates guidance from an optimized harness into executable supervision at the student's response boundary, enabling imitation learning across different harness action spaces. ❸ Experimental results show that agent-as-harness can outperform code-as-harness, that HARNESS-ZERO retains the gains of optimized harnesses after they are removed, and that the distilled model recovers harness-induced behaviors.

## 2 RELATED WORK

**Code-as-harness and its optimization.** Harnesses mediate the interaction between LLMs and their environments through tools, context management, control flow, and persistent state (Weng, 2026; Ning et al., 2026b; Zhou et al., 2026). Modern coding agents such as Claude Code (Anthropic, 2026), Codex (OpenAI, 2026a), Kimi Code (Moonshot AI, 2026), Pi (Earendil Works, 2026), and OpenCode (Anomaly, 2026) embody different design philosophies within this space. Natural-Language Agent Harnesses represent run-level policies as editable documents, which an agent interprets into actions (Pan et al., 2026). Automatic, training-free optimization has expanded from prompts (Khattab et al., 2023; Fernando et al., 2023; Agrawal et al., 2025) and context (Zhang et al., 2026c; Ye et al., 2026) to workflows (Hu et al., 2025; Zhang et al., 2025b) and harnesses (Lee et al., 2026; Lin et al., 2026a; Zhang et al., 2026b). Zhang et al. (2026a) train a model to generate per-task harnesses just in time. Harness design can also support test-time strong-to-weak transfer, in which a stronger model builds an inference-time harness for a fixed weaker model (Qian et al., 2026). Complementing this line of work, HARNESS-ZERO introduces agent-as-harness, which can outperform code-as-harness for frontier models and supports distilling optimized harness behavior into model weights.

**Co-evolution of model and harness.** Several approaches combine harness optimization with parameter updates. One line alternates between the two, using revised harnesses to generate data for the next model update and updated models to motivate further harness search (Hebbar et al., 2026; Chen et al., 2026c; Kim et al., 2026; Karten et al., 2026b). Other methods optimize the two jointly, treating model–harness compatibility as the objective and adapting the agent harness together with the policy trained from its trajectories (Chen et al., 2026a;b; Luo et al., 2026a; Mao et al., 2026). These studies show that harness and weight updates can reinforce each other, but the resulting gains may remain coupled to the harness. A controlled coding-agent study supports this concern (Le et al., 2026): changing the evaluation harness affected performance more than the training method, and training with feedback collected across harnesses did not improve transfer to a held-out minimal ReAct harness. HARNESS-ZERO instead uses optimized harnesses to guide a temporary harnessing agent and distills the resulting behavior into model weights, internalizing harness gains without retaining or routing the harness collection at deployment.

**Distillation from privileged guidance.** EvoHarness-RL provides early evidence of harness internalization on ALFWorld, where its trained agent learns to manage external state and makes fewer, more selective harness calls (Ning et al., 2026a). OPHSD distills privileged outputs produced by sequential draft–verify and plan–solve LLM workflows into a standalone model (Zhao et al., 2026). Together, these studies provide preliminary evidence that models can absorb harness-induced behavior into their parameters. However, each addresses only an individual mechanism rather than a complete tool-using agent harness; EvoHarness-RL retains the external workspace at deployment, and OPHSD does not involve an interactive agent loop. HARNESS-ZERO introduces a general method for distilling complete tool-using agent harnesses into a model that runs under a minimal target harness at deployment.

## 3 HARNESS-ZERO

HARNESS-ZERO trains a student model to reproduce behaviors induced by an evolved harness. The method has three stages (Figure 1). First (§ 3.1), we evolve a student-side harness on training tasks and adapt it into a private reference harness for a separate *harnessing agent*. Second (§ 3.2), the harnessing agent wraps the student's response loop for training trajectory collection. Third (§ 3.3), we apply SFT to reviewed trajectories jointly produced by the student and the harnessing agent. At deployment, the distilled student retains the evolved harness's gains under the target harness alone.

### 3.1 HARNESS EVOLUTION AND ADAPTATION

The first stage builds the domain-specific guidance used during training trajectory collection. We denote the fixed target harness by $h$, the evolved student-side harness by $h^\star$, and the private reference harness adapted from it by $\mathcal{K}$. Given training tasks $\mathcal{D}_{\text{train}}$, we write the two steps as

$$h^\star = \text{Evolve}(\mathcal{D}_{\text{train}}), \qquad \mathcal{K} = \text{Adapt}(h^\star; h). \tag{1}$$

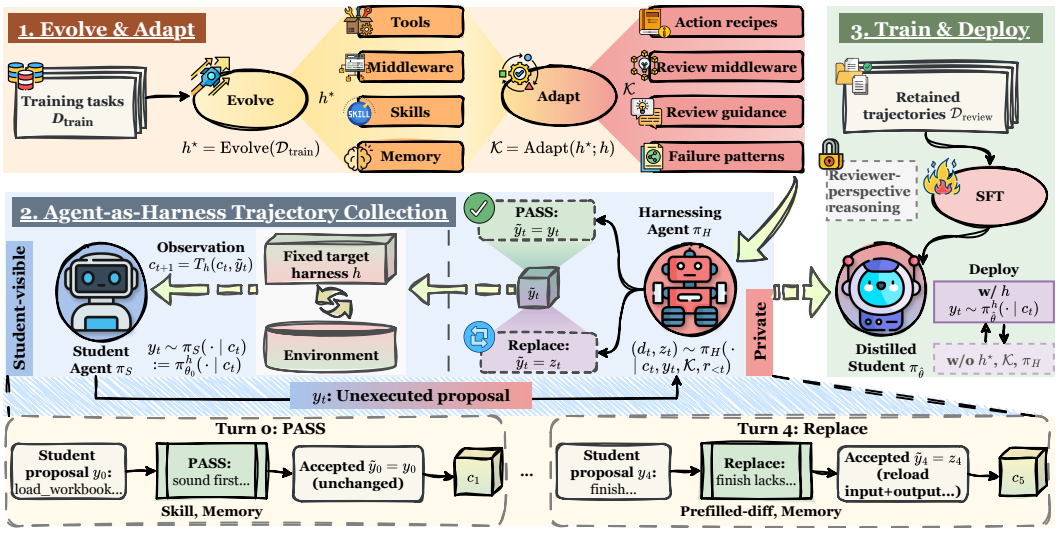


Figure 1: Overview of HARNESS-ZERO. **(1) Evolve and adapt.** We evolve a student-side harness $h^\star$ on training tasks, and adapt it into a private reference harness $\mathcal{K}$ relative to the fixed target harness $h$. **(2) Agent-as-harness trajectory collection.** The student $\pi_S$ runs under $h$ and proposes response $y_t$ at each turn. The harnessing agent $\pi_H$ reviews it using $\mathcal{K}$ and corrects it ($\tilde{y}_t$) when needed. $\tilde{y}_t$ is executed through $h$ and enters the student's context, while the review remains private. **(3) Train and deploy.** We apply SFT to the reviewed trajectories $\mathcal{D}_{\text{review}}$, masking any inadvertent reviewer-perspective reasoning from the loss. The distilled student is deployed under $h$ alone.

**Harness evolution.** We evolve $h^\star$ on training tasks. This step can use existing automated harness-optimization methods (Lee et al., 2026; Lin et al., 2026a; Zhang et al., 2026b). In our implementation, a simple skill-guided evolution loop analyzes recurring failures and updates the domain harness. The resulting $h^\star$ follows the DeepAgents abstraction (LangChain, 2025), which includes tools, middleware, skills, and memory. § A describes the three-round procedure and presents the evolution skill.

**Harness adaptation.** The evolved student-side harness $h^\star$ is designed to act directly around the student. Tools extend its action space, middleware modifies or blocks its execution loop, and skills and memory instruct the student. We adapt these components into the reference harness $\mathcal{K}$ used by the harnessing agent; the process can be automated by an agent. The adaptation is relative to $h$, because any correction the harnessing agent constructs from $\mathcal{K}$ must ultimately be executed by the student in the target harness's action space. In general, the adaptation preserves the harness components' intended behavior while changing their audience and enforcement point. Tools become specifications for constructing student-native equivalents; student-side middleware becomes review middleware that privately alerts the harnessing agent when the corresponding condition is met; and skills and memory become diagnostic criteria and intervention guidance. The adaptation may also produce a domain prompt appended to the harnessing agent's system prompt, stating the domain's review policy. § B gives detailed examples of such adaptation.

### 3.2 AGENT-AS-HARNESS

Let $\pi_\theta^h$ denote the response policy induced when a model with parameters $\theta$ operates under $h$, and let $T_h(c, y)$ denote the transition function that executes response $y$ through $h$ from context $c$ and returns the next student-visible context.

**Code-as-harness**, after harness evolution, runs the base student with parameters $\theta_0$ directly under $h^\star$ (Ning et al., 2026b):

$$y_t \sim \pi_{\theta_0}^{h^\star}(\cdot \mid c_t), \qquad c_{t+1} = T_{h^\star}(c_t, y_t). \tag{2}$$

**Agent-as-harness** instead runs the student under $h$, with a harnessing agent wrapping it at its response boundary. The harnessing agent intercepts each proposed response before execution and

either passes it or replaces it. It uses $\mathcal{K}$ for private guidance on when to intervene and how to construct a replacement; $\mathcal{K}$ neither acts on the environment nor enters the student-visible context. The target harness $h$ defines the student's action space and executes the accepted response. The student and harnessing agent may use the same underlying model, with different instructions and context for their respective roles.

At turn $t$, let $c_t$ denote the student's visible context: the task, previous accepted responses, and resulting environment observations. The student policy $\pi_S$ proposes a response $y_t$, which the harnessing policy $\pi_H$ reviews using $c_t$, the guidance in $\mathcal{K}$, and its private history $r_{<t}$:

$$\begin{aligned} y_t &\sim \pi_S(\cdot \mid c_t) := \pi^h_{\theta_0}(\cdot \mid c_t), \\ (d_t, z_t) &\sim \pi_H(\cdot \mid c_t, y_t, \mathcal{K}, r_{<t}), \\ \tilde{y}_t &= \begin{cases} y_t, & d_t = \textsc{pass}, \\ z_t, & d_t = \textsc{replace}, \end{cases} \\ c_{t+1} &= T_h(c_t, \tilde{y}_t), \end{aligned} \tag{3}$$

where $d_t$ is the review decision, $z_t$ is a complete replacement response valid under $h$, and $r_{<t}$ contains earlier review exchanges and the harnessing agent's prior file-system reads from $\mathcal{K}$. A single harnessing-agent session spans the entire student rollout. At each review, it receives the student-visible events added since the previous review and the current unexecuted proposal. Only the accepted response $\tilde{y}_t$ enters the student-visible trajectory. Any actions it contains are then executed through $h$, and their observations become part of $c_{t+1}$. The rejected proposal and private review remain outside this trajectory. This process realizes source-harness guidance as a target-harness trajectory incrementally, with each correction conditioned on the student's current interaction history. § C specifies the review process and gives the harnessing agent's system prompt.

**Code-as-harness vs. agent-as-harness**

**Code-as-harness** (Eq. (2)) runs the student directly under the evolved $h^\star$, which shapes and executes $y_t$.

**Agent-as-harness** (Eq. (3)) runs the student under $h$, while a harnessing agent reviews the student's proposal $y_t$ using $\mathcal{K}$, whose guidance is adapted from $h^\star$. The accepted response $\tilde{y}_t$ is then executed through $h$.

**Intervention policy and constraints.** Guidance from $\mathcal{K}$ steers rollouts toward behaviors and states that an unaided student under $h$ may not reach. To facilitate SFT, the harnessing agent minimizes changes to the student's proposals. It passes sound proposals unchanged. When intervention is necessary, it makes the smallest coherent correction needed to follow $\mathcal{K}$'s guidance and preserves the rest of the proposal whenever possible. Each replacement is a complete response valid under $h$ that continues from the current student-visible state.

The harnessing agent's privileged access is limited to reading $\mathcal{K}$. It cannot inspect hidden solutions or verifier feedback, nor can it access environment state outside $c_t$. Any additional task evidence must therefore be obtained by proposing an action available under $h$. For example, it can replace a premature completion with code that checks the student's work. Executing the code through $h$ adds both the check and its result to the student-visible trajectory, grounding the intervention in student-observable evidence. Together, these intervention and grounding constraints make the reviewed trajectories directly usable to fine-tune the student for operation under $h$.

### 3.3 Learning from reviewed trajectories

We perform imitation learning under the target harness by applying SFT to the accepted responses in reviewed rollouts (Eq. (3)), including both unchanged student proposals and harness-guided replacements. Replacements should be self-contained responses written from the student's perspective. Because they are generated within the private review context, they may inadvertently include reviewer-perspective reasoning about the student's proposal or the review decision. We mask such reasoning from the loss; § D details data collection and the filtering rule. Let $\mathcal{D}_{\text{review}}$ denote the

retained trajectories, we optimize:

$$\hat{\theta} = \arg\min_{\theta} - \sum_{\tau \in \mathcal{D}_{\text{review}}} \sum_{t=1}^{T_\tau} \log \pi_\theta^h(\tilde{y}_t \mid c_t). \tag{4}$$

Here, $T_\tau$ is the number of accepted response turns in trajectory $\tau$. At deployment, the distilled policy acts directly under the same target harness:

$$y_t \sim \pi_{\hat{\theta}}^h(\cdot \mid c_t), \qquad c_{t+1} = T_h(c_t, y_t). \tag{5}$$

The deployed system is therefore $(\pi_{\hat{\theta}}, h)$, without $h^\star$, $\mathcal{K}$, or the harnessing agent. The training objective is for $\pi_{\hat{\theta}}$ to reproduce under $h$ the behavior patterns induced by $h^\star$.

# 4 Experiments

We first compare agent-as-harness with code-as-harness at inference time, then test whether Harness-Zero can distill an optimized harness into model weights.

## 4.1 Experimental setup

**Tasks and splits.** We evaluate three task domains. (1) **SpreadsheetBench Verified** contains 400 real-world spreadsheet-manipulation tasks (Ma et al., 2024); we use 300 for harness evolution and training data collection and hold out 100 for evaluation. (2) **AppWorld** evaluates interactive tool use across simulated applications (Trivedi et al., 2024); we merge its official train and development sets into a 147-task training split and hold out the 168 `test_normal` tasks, grouped into 56 three-task scenarios. (3) **USPTO Retrosynthesis** covers single-step precursor prediction from the USPTO reaction corpus (Lowe, 2012; Jin et al., 2017); we use a 500-task training split balanced across its ten reaction classes and a disjoint 100-task test split. All three run in the Harbor framework (Harbor Framework Team, 2026; Shi et al., 2026). We report single-run task success (pass@1, %) for SpreadsheetBench and USPTO and scenario goal completion (SGC, %) for AppWorld.

**Models and training.** The target harness $h$ is a minimal mini-SWE-agent-style harness (SWE-agent Team, 2025) with a fixed system prompt and a single Bash execution tool. The training-free experiments use GPT-5.6 Sol (OpenAI, 2026b) and DeepSeek-V4-Pro (DeepSeek-AI, 2026). The distillation experiments use Qwen3.5-9B (Qwen Team, 2026) as the base model and GPT-5.6 Sol as the harnessing agent; harness evolution uses Kimi K3 (Kimi Team, 2026) under Kimi Code. Reasoning is enabled for all models, with reasoning effort set to *high* when applicable. After rollout collection and filtering, the training data comprise 487 rollouts for SpreadsheetBench, 282 for AppWorld, and 500 for USPTO. We perform LoRA SFT on Qwen3.5-9B for two epochs using the Tinker recipe (Thinking Machines Lab, 2025). § E gives the student prompt, the model access routes, and the full training configuration.

## 4.2 Evaluating agent-as-harness

We first compare agent-as-harness against code-as-harness at inference time, with no parameter updates. Table 1 varies two factors: whether the evolved harness is available, and whether it reaches the student as code wrapped around it or as a harnessing agent reviewing its responses. The two code-as-harness conditions use no harnessing agent: mini-SWE-agent runs the student under $h$ alone, and meta-harness mounts $h^\star$ on top of it. The two agent-as-harness conditions keep the student under $h$ and add a harnessing agent that consults either an empty reference harness (w/o evolved) or $\mathcal{K}$ adapted from $h^\star$ (w/ evolved). In both, the same model plays student and harnessing agent, so the gains cannot come from a stronger supervising model. On the frontier models we reuse the $h^\star$ evolved on Qwen3.5-9B.

**Obs.❶ With evolved harness, agent-as-harness outperforms code-as-harness on average.** Across the six settings in Table 1, agent-as-harness with the adapted $\mathcal{K}$ averages 81.1%, against 78.1% for meta-harness and 68.6% for mini-SWE-agent. With an empty $\mathcal{K}$ it averages only 69.2%, so review alone explains little of the gain. Beyond these benchmark numbers, agent-as-harness also offers better adaptability across model updates. A code harness encodes assumptions about how a

Table 1: Evaluating four harness settings. Best per setting in bold, second best underlined.

| Benchmark | Model | Code-as-Harness | | Agent-as-Harness | |
|---|---|---|---|---|---|
| | | mini-SWE-agent ($h$) | meta-harness ($h^\star$) | w/o evolved ($h, \varnothing$) | w/ evolved ($h, \mathcal{K}$) |
| SpreadsheetBench | DeepSeek-V4-Pro | 76.0 | 77.0 ↑1.0 | <u>78.0</u> ↑2.0 | **86.0** ↑10.0 |
| | GPT-5.6 Sol | 83.0 | <u>84.0</u> ↑1.0 | 83.0 ↑0.0 | **88.0** ↑5.0 |
| AppWorld | DeepSeek-V4-Pro | 75.0 | **91.1** ↑16.1 | 76.8 ↑1.8 | <u>89.3</u> ↑14.3 |
| | GPT-5.6 Sol | **96.4** | <u>94.6</u> ↓1.8 | <u>94.6</u> ↓1.8 | **96.4** ↑0.0 |
| USPTO | DeepSeek-V4-Pro | 31.0 | <u>56.0</u> ↑25.0 | 30.0 ↓1.0 | **60.0** ↑29.0 |
| | GPT-5.6 Sol | 50.0 | <u>66.0</u> ↑16.0 | 53.0 ↑3.0 | **67.0** ↑17.0 |
| **Average** | | 68.6 | <u>78.1</u> ↑9.5 | 69.2 ↑0.6 | **81.1** ↑12.5 |
| **Relative improvement over $h$** | | — | <u>+22.5%</u> | +1.0% | **+27.6%** |

Table 2: Harness distillation results across three benchmarks.

| Setting | Benchmark | | | Avg. |
|---|---|---|---|---|
| | SpreadsheetBench | AppWorld | USPTO | |
| *Base model* | | | | |
| mini-SWE-agent ($h$) | 31.0 | 26.8 | 12.0 | 23.3 |
| meta-harness ($h^\star$) | <u>39.0</u> ↑8.0 | <u>48.2</u> ↑21.4 | **38.0** ↑26.0 | <u>41.7</u> ↑18.4 |
| DeepAgents | 35.0 ↑4.0 | 19.6 ↓7.2 | 7.0 ↓5.0 | 20.5 ↓2.8 |
| Claude Code | 31.0 ↑0.0 | 10.7 ↓16.1 | 6.0 ↓6.0 | 15.9 ↓7.4 |
| *Distilled model* | | | | |
| **HARNESS-ZERO ($h$)** | **44.0** ↑13.0 | **58.9** ↑32.1 | <u>30.0</u> ↑18.0 | **44.3** ↑21.0 |

model should act, and as capabilities change those assumptions go stale, forcing the harness to be re-adapted for each new model (Qian et al., 2026; Liu, 2026). Agent-as-harness moves that adaptation into inference: the harnessing agent interprets $\mathcal{K}$ against the current trajectory and decides when and how to intervene, so the guidance stays reusable and a stronger harnessing model directly improves how it is applied. While this approach requires a sufficiently capable harnessing agent (§ F), we expect its advantage over fixed code harnesses to widen as foundation models continue to improve.

### 4.3 EVALUATING AGENT HARNESS DISTILLATION

We next test whether the behavior induced by the optimized harness $h^\star$ can be retained after distillation, when $h^\star$, reference harness $\mathcal{K}$, and harnessing agent are removed. Table 2 compares the base model under $h$, the base model with $h^\star$ mounted, and the distilled model under $h$. As reference points, we also run the base model under two general-purpose harnesses: DeepAgents (LangChain, 2025), the abstraction on which $h^\star$ is built (§ 3.1), and Claude Code (Anthropic, 2026).

**Obs.❷ Distillation raises the base model's macro-average performance by 21.0 points and surpasses $h^\star$.** HARNESS-ZERO raises the macro average from 23.3% to 44.3%, a 21.0-point absolute gain and a 90.1% relative improvement. It also exceeds the 41.7% macro average of the base model equipped with $h^\star$. Neither general-purpose harness benefits the base model. On one hand, a 9B model handles their larger, generic tool suites and extended context poorly. On the other hand, domains such as USPTO and AppWorld demand domain-specific tooling and constraints present in $h^\star$ (such as molecular validation tools for USPTO), rendering generic tools beyond basic bash largely unhelpful and distracting.

**Obs.❸ Procedural harness behavior is easier to internalize than deep domain knowledge.** On SpreadsheetBench and AppWorld, HARNESS-ZERO under $h$ alone surpasses $h^\star$. Their harnesses mainly encode recurring procedures for state inspection, targeted changes, and verification, which reviewed trajectories can demonstrate directly. On USPTO, HARNESS-ZERO improves over the base model (30.0% vs. 12.0%) but trails $h^\star$ (38.0%). There, $h^\star$ also supplies reaction priors, candidate-generation logic, and executable SMILES validation; transferring this knowledge and functionality may require broader pretraining or mid-training coverage, or more distillation trajectories.

### 4.4 COMPARING ALTERNATIVE DISTILLATION SIGNALS

To ablate the agent-as-harness recipe and understand what makes it effective, we compare against alternative training trajectory sources on USPTO. We vary the rollout generator, the use of $h^\star$, and the private context available to the harnessing agent. The three direct-distillation baselines collect rollouts from the teacher (GPT-5.6 Sol, used in the harnessing agent of HARNESS-ZERO) under $h$, the teacher with $h^\star$ mounted, and the base student with $h^\star$ mounted. The two harnessing-agent controls keep the base student under $h$ and give the harnessing agent either an empty $\mathcal{K}$ or the oracle answer.

All conditions start from Qwen3.5-9B, use the same 500-task collection split and training recipe, and evaluate the 2-epoch checkpoint under $h$ on the test set. Trajectories are retained after the same structural and privacy checks. For each condition, Table 3 reports the source's success rate on the 500 collection tasks and the test pass@1 of the resulting student under $h$.

Table 3: Comparing alternative distillation signals.

| Trajectory source | Collection success | Test pass@1 under $h$ |
|---|---|---|
| Base (untrained) | — | 12.0 |
| *Direct distillation* | | |
| Teacher rollout | 52.0 | 12.0 ↑0.0 |
| Teacher under $h^\star$ | 62.0 | 3.0 ↓9.0 |
| Student under $h^\star$ | 39.4 | 12.0 ↑0.0 |
| *Agent-as-harness* | | |
| Review with empty $\mathcal{K}$ | 44.2 | 11.0 ↓1.0 |
| Review with oracle answer | 98.6 | <u>15.0</u> ↑3.0 |
| **HARNESS-ZERO ($\mathcal{K}$)** | 59.4 | **30.0** ↑18.0 |

**Obs.❹ Effective supervision comes from harness-guided review, not from stronger demonstrations alone.** Directly fine-tuning on GPT-5.6 Sol trajectories leaves the student at the 12% base result, even though that source succeeds on 52.0% of the collection tasks. Likewise, trajectories collected by the base student under $h^\star$ yield 12% after SFT. This comparison isolates the importance of starting from the student's trajectory and translating harness guidance into corrections compatible with its current state and target action space.

**Obs.❺ Procedural guidance provides better distillation supervision than answer access or generic review.** Review with an empty $\mathcal{K}$ reaches only 11%. Providing oracle answers raises collection success to 98.6%, yet the distilled model reaches only 15%. By comparison, $\mathcal{K}$ contains no task answers and reaches a lower collection success of 59.4%, yet the student distilled from its trajectories reaches 30% pass@1. Collection success therefore does not predict distillation value. Oracle access encourages non-generalizable shortcut corrections. By contrast, $\mathcal{K}$ instills reusable procedural behavior (e.g., reasoning from reaction templates, proposing candidates, and systematically validating reactant sets) that the student can execute independently under $h$.

**Obs.❻ Executing $h^\star$ during collection does not make its behavior transferable.** Mounting $h^\star$ improves GPT-5.6 Sol's collection success from 52.0% to 62.0%, but the resulting distilled student falls from 12% to 3%. The student-generated counterpart also succeeds on 39.4% of collection tasks but returns to 12% after SFT. The failure is consistent with action-space mismatch. The model distilled from teacher trajectories under $h^\star$ extensively attempts unavailable harness-tool calls, and many trials thus exhaust the turn limit. HARNESS-ZERO avoids this mismatch by expressing each correction through $h$.

### 4.5 BEHAVIORAL INTERNALIZATION

To evaluate harness distillation at a finer granularity, we measure how well the distilled model internalizes the behavioral patterns encoded in $h^\star$. We first translate patterns of enabled tools, middleware, memory, and skills in $h^\star$ into trajectory detectors. For each detector, we compare the base model's trajectories under $h^\star$ and $h$ on the same test task. We select tasks where the pattern appears only under $h^\star$ and keep a pattern only when at least 10 tasks meet this criterion. This yields 18 harness-exclusive patterns on SpreadsheetBench, 6 on USPTO, and 4 on AppWorld; § G details the mining procedure and explains every pattern. For each pattern, its *recovery rate* is the fraction of the selected tasks on which HARNESS-ZERO exhibits the same behavior.

**Obs.❼ HARNESS-ZERO internalizes behaviors contributed by $h^\star$.** Averaged over the 28 patterns in Table 4, HARNESS-ZERO recovers 82.3% of the harness-exclusive behaviors. Recovery spans memory, skill, tool, and middleware sources, covering both model-visible guidance and exe-

Table 4: Recovery of 28 harness-exclusive behaviors across three domains.

| Source | Pattern | Recovery |
|---|---|---|
| | **SpreadsheetBench** (18 patterns) | |
| Memory | Edit existing workbook | 31/34 (91%) |
| | Assert workbook structure | 20/27 (74%) |
| | Avoid fragile coordinates | 19/19 (100%) |
| | Save requested workbook | 14/14 (100%) |
| Skill | Inspect before editing | 32/32 (100%) |
| | Inspect target/examples | 14/18 (78%) |
| | Use `solution.py` | 53/76 (70%) |
| | Execute final script | 52/75 (69%) |
| | Read cached formulas | 28/43 (65%) |
| | Normalize matching keys | 6/10 (60%) |
| | Use closed target ranges | 30/32 (94%) |
| | Reload saved workbook | 17/17 (100%) |
| Middleware | Protect pre-filled cells | 24/26 (92%) |
| | Write computed literals | 19/21 (90%) |
| | Populate answer range | 28/30 (93%) |
| | Scan range after save | 20/24 (83%) |
| | Check output existence | 16/16 (100%) |
| | Stop after delivery | 21/28 (75%) |
| | **USPTO** (6 patterns) | |
| Skill | Parse product with RDKit first | 23/23 (100%) |
| | Enumerate candidate sets | 59/88 (67%) |
| Tool | Validate with RDKit | 60/60 (100%) |
| | Canonicalize SMILES | 73/73 (100%) |
| | Check heavy-atom coverage | 40/54 (74%) |
| Middleware | Validate the final answer | 45/47 (96%) |
| | **AppWorld** (4 patterns) | |
| Skill | Retrieve complete pages | 21/53 (40%) |
| | Read back mutations | 6/12 (50%) |
| Middleware | Avoid repeated failed calls | 22/31 (71%) |
| | Choose action vs. answer | 24/33 (73%) |

cutable components. Each pattern is scored only on the tasks selected for it, where the base model exhibits the behavior under $h^\star$ but never under $h$, so the base model scores 0% on every pattern by construction. These recovery rates therefore measure behavior that distillation adds, providing direct evidence that HARNESS-ZERO internalizes behavior induced by $h^\star$.

## 5 CONCLUSION AND DISCUSSION

HARNESS-ZERO turns an evolved student-side harness $h^\star$ into training supervision for a model operating under a fixed target harness $h$. Guided by the adapted reference harness $\mathcal{K}$, a harnessing agent rewrites the student's proposals before execution, producing training trajectories compatible with $h$. Across three domains, agent-as-harness outperforms code-as-harness on frontier models (81.1% vs. 78.1% on average), and ablations identify harness-guided review as the most effective of the tested supervision sources. After SFT, the student under $h$ alone raises macro-average task success from 23.3% to 44.3%, surpassing the 41.7% achieved with $h^\star$, and recovers $h^\star$-specific behaviors at an average rate of 82.3% over 28 patterns.

**Limitations.** The method depends on a capable harnessing model. With weaker models, review can become harmful and agent-as-harness loses its advantage over code-as-harness (§ F). Reviewing every proposal also increases collection cost; each step requires an additional model call that processes the trajectory and $\mathcal{K}$, raising mean USPTO latency to $2.4\times$ during collection, an overhead absent after distillation. SFT may also fail to fully internalize deep domain knowledge encoded by $h^\star$. In addition, some harness mechanisms, e.g., context management, are not fully expressible as student responses out of the box. HARNESS-ZERO therefore does not remove the need for a harness, but narrows what that harness must provide, and we expect a minimal one to suffice as harness distillation improves.

**Future work.** A harnessing agent must anticipate how executing the student's proposal would change the environment and whether an intervention would yield a better trajectory, so effective harnessing depends on an accurate model of agent–environment dynamics; training stronger agent world models (Zuo et al., 2026) and specializing them for harnessing decisions is a promising model-level direction. At the framework level, reviewing every proposal and restricting intervention to full-response replacement incurs unnecessary calls and provides only coarse-grained control. Future work should establish proxy signals to invoke review selectively and leverage finer-grained harnessing such as token insertion and latent-space steering. The training signal can also be improved. Each replacement pairs a rejected and a preferred response at the same state, so preference learning could exploit comparison signals that response-level SFT discards.

Overall, HARNESS-ZERO paves the way for recursive self-improvement based on agent harnesses: distilling task-specific harnesses into shared weights lets behaviors and knowledge accumulate in parameters, so better models build better harnesses and each harness returns its gains to the weights.

## Reproducibility Statement

We provide the details needed to reimplement Harness-Zero and reproduce our results. The method is fully specified in § 3: the harness evolution procedure and the evolution skill are given in § A, the adaptation from the evolved harness to the reference harness is described with concrete examples in § B, and the agent-as-harness review interface, submission format, and harnessing-agent system prompt are given in § C. § D specifies trajectory collection, filtering, reviewer-perspective masking, and the privacy assertions that keep harnessing-agent-private content out of the training data. Experimental setup, including tasks, splits, and models, is described in § 4.1, and implementation details, including the student system prompt, model access, and training configuration, are given in § E. We will additionally open-source the code.

## AI Use Statement

We used AI assistants in two roles. First, to check grammar and to polish text the authors had written. Second, for routine coding assistance during implementation. The method, the experimental design, and every implementation decision affecting the reported results were made by the authors, who verified all AI-assisted output and take full responsibility for this paper.

## References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Anomaly. OpenCode: The open source coding agent. https://github.com/anomalyco/opencode, 2026. Accessed: 2026-09-10.

Anthropic. Claude Code. https://code.claude.com/docs, 2026. Accessed: 2026-09-10.

Chaokun Chang, Yukun Zhou, Kaihua Fu, Dakai An, Tianyu Feng, Hanfeng Lu, Sheng Yao, Pu Guo, Yinghao Yu, Yizhou Shan, Bo Li, Binhang Yuan, and Wei Wang. From LLM inference to agentic workloads: Characterization and implications for serving systems. *arXiv preprint arXiv:2608.15127*, 2026. URL https://arxiv.org/abs/2608.15127.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. Harnessforge: Joint harness and policy evolution for adaptive agent systems. *arXiv preprint arXiv:2606.01779*, 2026a. URL https://arxiv.org/abs/2606.01779.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026b. URL https://arxiv.org/abs/2606.14249.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, et al. Co-harness: Co-evolving harnesses and model weights for LLM agents. *arXiv preprint arXiv:2607.22688*, 2026c.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Earendil Works. Pi: An agent harness and coding agent. https://github.com/earendil-works/pi, 2026. Accessed: 2026-09-10.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. https://github.com/harbor-framework/harbor, 2026.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, et al. SIA: Self improving AI with harness & weight updates. *arXiv preprint arXiv:2605.27276*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

Wengong Jin, Connor W Coley, Regina Barzilay, and Tommi Jaakkola. Predicting organic reaction outcomes with Weisfeiler-Lehman network. In *Advances in Neural Information Processing Systems*, volume 30, pp. 2607–2616, 2017.

Seth Karten, Alex L Zhang, Kevin Thomas, Sebastian Müller, Elie Bakouch, Daniel Auras, Mika Senghaas, Fares Obeid, Konstantin Dunas, Johannes Hagemann, et al. Prime Agent: A self-improving RLM harness. *arXiv preprint arXiv:2608.23552*, 2026a.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, et al. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026b.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Haechan Kim, Yoonho Lee, Gisang Lee, Chelsea Finn, and Kangwook Lee. WHALE: A simple recipe for joint harness–weight optimization. *arXiv preprint arXiv:2609.00196*, 2026. URL https://arxiv.org/abs/2609.00196.

Kimi Team. Kimi K3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

LangChain. Deep agents. https://github.com/langchain-ai/deepagents, 2025. Accessed: 2026-01-19.

Chenqian Le, Jiayi Cheng, Qijia He, Runhao Li, Yinghao Li, and Xupeng Chen. What does multi-harness RL learn? credit assignment and portability in coding agents. *arXiv preprint arXiv:2609.04518*, 2026. URL https://arxiv.org/abs/2609.04518.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Jiahang Lin, Shichun Liu, Chengjun Pan, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026a.

Liangtao Lin, Qingang Zhang, Zhaomeng Zhu, Tianwei Zhang, and Yonggang Wen. Task-aware harness provisioning for LLM agents in mission-critical infrastructure operations. *arXiv preprint arXiv:2608.17433*, 2026b. URL https://arxiv.org/abs/2608.17433.

Ming Liu. More is not always better: Cross-component interference in LLM agent scaffolding. *arXiv preprint arXiv:2605.05716*, 2026. URL https://arxiv.org/abs/2605.05716.

Daniel Mark Lowe. *Extraction of chemical structures and reactions from the literature*. PhD thesis, University of Cambridge, 2012.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651(8107):914–919, 2026.

Haochen Luo, Yi Huang, Sichun Luo, Fengyuan Liu, Lei Li, Zefa Hu, Junlan Feng, and Qi Liu. Harness-aware self-evolving: Co-evolving model weights, harness, and task solutions. *arXiv preprint arXiv:2607.03935*, 2026a. URL https://arxiv.org/abs/2607.03935.

Xiaotian Luo, Dizhan Xue, Fengxingyu Wang, Chuanrui Hu, and Yafeng Deng. HarnessBank: Semantic gene-bank search with gated verification for agent-harness self-evolution. *arXiv preprint arXiv:2607.13683*, 2026b. URL https://arxiv.org/abs/2607.13683.

Zeyao Ma et al. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. arXiv:2406.14991, 2024.

Ziyu Ma, Hailang Huang, Shun Zou, Yong Wang, Shidong Yang, Yiming Hu, Fei Wei, and Xiangxiang Chu. LongHorizon-Harness: Advancing long-horizon agents for real-world tasks. *arXiv preprint arXiv:2608.01964*, 2026. URL https://arxiv.org/abs/2608.01964.

Qinghua Mao, Wanying Qu, Dadi Guo, Leitao Yuan, Qingyu Liu, Yu Li, Guanxu Chen, Yanwei Fu, Xi Lin, Xia Hu, and Dongrui Liu. Safeevolve: Harness-policy co-evolution from agent experience for safety alignment. *arXiv preprint arXiv:2609.02786*, 2026. URL https://arxiv.org/abs/2609.02786.

Moonshot AI. Kimi Code CLI. https://github.com/MoonshotAI/kimi-code, 2026. Accessed: 2026-09-10.

Xuying Ning, Dongqi Fu, Tianxin Wei, et al. EvoHarness-RL: Learning self-evolving runtime harness for long-horizon LLM agents. *arXiv preprint arXiv:2608.05446*, 2026a.

Xuying Ning, Katherine Tieu, Dongqi Fu, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026b.

OpenAI. Codex CLI. https://github.com/openai/codex, 2026a. Accessed: 2026-09-10.

OpenAI. GPT-5.6 System Card. https://deploymentsafety.openai.com/gpt-5-6, 7 2026b. Accessed: 2026-09-14.

OpenRouter, Inc. Openrouter: The unified interface for llms. https://openrouter.ai/, 2025. Accessed: 2026-01-16.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses. *arXiv preprint arXiv:2603.25723*, 2026. URL https://arxiv.org/abs/2603.25723.

Cheng Qian, Wenting Zhao, Liangwei Yang, et al. AI4AI at test-time: Strong-to-weak capability transfer via harnesses. *arXiv preprint arXiv:2608.12307*, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2 2026. Accessed: 2026-09-14.

Lin Shi, Haowei Lin, Zixuan Zhu, Xiaoyue Zhou, Xiang Li, et al. Harbor adapters and harbor-index: Infrastructure and a curated meta-dataset for large-scale agentic evaluation. *arXiv preprint arXiv:2609.04298*, 2026.

SWE-agent Team. mini-SWE-agent. https://github.com/SWE-agent/mini-swe-agent, 2025.

Thinking Machines Lab. Announcing Tinker. https://thinkingmachines.ai/news/announcing-tinker/, oct 2025. Accessed: 2026-09-17.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 16022–16076, 2024.

Lilian Weng. Harness engineering for self-improvement. https://lilianweng.github.io/posts/2026-07-04-harness/, 7 2026. Lil'Log.

Haoyang Yan, Min-le Su, Hangfan Zhang, Zhanhao Li, Chen Zhang, Shao Zhang, Yang Chen, Lei Bai, and Shuyue Hu. Harness-of-Harness: Multi-day autonomous software development with continual improvement. *arXiv preprint arXiv:2609.01481*, 2026. URL https://arxiv.org/abs/2609.01481.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026. URL https://arxiv.org/abs/2605.27922.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2601.21557.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=LkzuPorQ5L.

Guibin Zhang, Leo Lu, Fangzhou Xie, et al. JIT-Agent: Scaling harness intelligence via just-in-time harness evolution. *arXiv preprint arXiv:2608.25593*, 2026a. URL https://arxiv.org/abs/2608.25593.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026b. URL https://arxiv.org/abs/2606.09498.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=z5uVAKwmjf.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026c. URL https://openreview.net/forum?id=eC4ygDs02R.

Zhengyang Zhao, Lu Ma, and Wentao Zhang. Training with harnesses: On-policy harness self-distillation for complex reasoning. *arXiv preprint arXiv:2605.08741*, 2026.

Chenyu Zhou, Huacan Chai, Wenteng Chen, et al. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, et al. Qwen-AgentWorld: Language world models for general agents. *arXiv preprint arXiv:2606.24597*, 2026. URL https://arxiv.org/abs/2606.24597.

# Harness-Zero: Harness Distillation via Agent-as-Harness (Appendix)


**A Skill-guided harness evolution** **15**

A.1 Evolution methods . . . . . 15

A.2 Evolved components . . . . . 18

**B Harness adaptation** **18**

B.1 Reference harness inventory . . . . . 18

B.2 Worked example: preserving pre-filled cells . . . . . 18

B.3 Adapting skills and memory . . . . . 20

**C Review protocol and harnessing-agent prompt** **22**

C.1 Session structure . . . . . 22

C.2 Submission format . . . . . 22

C.3 System prompt . . . . . 23

**D Trajectory collection and filtering** **24**

**E Implementation details** **25**

E.1 Student harness . . . . . 25

E.2 Models and access . . . . . 25

E.3 Training configuration . . . . . 25

**F Additional analysis of agent-as-harness** **25**

**G Harness-exclusive behavior patterns** **26**

G.1 Approach . . . . . 26

G.2 Mined patterns . . . . . 26

# A SKILL-GUIDED HARNESS EVOLUTION

## A.1 EVOLUTION METHODS

We evolve one shared student-side harness for each benchmark. The process, also known as meta-harness, begins with rollouts under $h$ on the training split. In each round, a swarm of analysis agents examines the tasks that failed in the preceding round and proposes reusable changes to tools, middleware, skills, or memory. A main evolution agent consolidates these proposals into the shared harness and evaluates the same model with that harness attached. Tasks that pass leave the evolution process, while the remaining failures form the next round. We run this process for three rounds.

The evolution objective is performance under the student-side harness itself. Evaluation therefore uses the target student agent with the evolved harness attached and no harnessing-agent intervention. Proposed components must address failure patterns shared across tasks. Skill 1 shows the skill followed by the evolution agent.

```
---
name: student-harness-evolve
description: Generic workflow for evolving a shared student harness -- for a benchmark's
     Harbor-format train task set, take bare miniswe rollouts as the baseline, handle only
     the tasks the previous round got wrong, use a swarm (default 10 coder subagents) to
     analyze rollout trajectories in parallel, have the main agent write the shared student
     harness (tools / middleware / skills / memory) under `harness_bank/<domain>/`, and
     iterate for 3 rounds with "miniswe eval rollouts loading that harness" as the evaluator
     . The evolved artifact can later be rewritten into a teacher-side harness.
---

# Shared Student Harness Evolution Guide

## What this workflow does

For a benchmark's train task set, evolve **one** deliverable: a **student-side harness
     shared by all tasks**, kept in a per-benchmark bank (`harness_bank/<domain>/`):

- `tools/` -- prebuilt tools (real StructuredTools loaded into miniswe alongside `execute`);
- `middlewares/` -- student-side langchain AgentMiddleware;
- `skills/<name>/SKILL.md` + `memory.md` -- skill and memory material, uploaded into the
     sandbox and exposed to the student via deepagents' SkillsMiddleware / MemoryMiddleware.

**The evaluator** is a bare miniswe rollout with the harness loaded (`ahd.harness:
     AHDMinisweAgent`, teacher passthrough, i.e. no teacher intervention; the bank is
     mounted via `student_harness_dir`). It measures "does this student harness help the
     model itself". A campaign runs **3 rounds** on the rhythm: "select the tasks the
     previous round got wrong -> parallel swarm analysis -> main agent writes the bank ->
     eval rollout on exactly those failed tasks -> attribution". Tasks already solved count
     as "the harness is good enough for them" and are not rolled out again in later rounds.

## Campaign parameters (set at each instantiation)

| Parameter | Meaning |
|---|---|
| `<DOMAIN>` | Domain name (determines the bank path `harness_bank/<domain>/`; for the
     student side, prefer a `<benchmark>_student` suffix to distinguish it from teacher-side
      banks) |
| `<TASKS>` | Harbor task path + task-list file (e.g. `-p data/spreadsheetbench-verified`,
     task list `experiments/spreadsheetbench_train300.txt`) |
| `<MODEL>` | Evaluation model (e.g. `openrouter:qwen/qwen3.5-9b`, reasoning explicitly
     enabled) |
| `<BASELINE_JOBS>` | Job directory of the baseline (bare, no harness) rollout; the input
     for round-1 failure analysis. If it does not exist, run the baseline first |
| `<N_AGENTS>` | Swarm concurrency, at most 10 subagents at a time; all of the round's
     failed tasks are distributed among these subagents, with no per-subagent task cap |

## Bank structure and integration

```
harness_bank/<domain>/
|-- registry.py        # single entry point; API contract below
|-- manifest.json      # enablement manifest: tools/middlewares/skills lists + memory switch
     ; toggle components only here
|-- memory.md          # accumulated checklist clauses (English, no source attribution,
     grouped by topic)
|-- tools/             # *.py, each exposing make_tool(backend) -> StructuredTool
|-- middlewares/       # *.py, each exposing make_middleware() -> AgentMiddleware
|-- skills/            # <name>/SKILL.md, deepagents skill format (progressive disclosure)
`-- README.md          # bank description and provenance
```
```

```
```

- **Registry API contract** (consumed by the loader under exactly these names): `TOOL_FACTORIES` / `MIDDLEWARE_FACTORIES` (dict, name -> `"module:function"` import string, which the loader resolves into a factory). `catalog_text()` (component listing rendering) is only for logs and audits; the loader does not consume it. Component docstrings are the catalog source.
- **Loading semantics** (the contract of `student_harness_dir`, implemented in `src/ahd/student_harness.py` / `src/ahd/student.py`):
  1. tools are appended to miniswe's tools list (after `execute`);
  2. the `skills/` directory and memory.md are uploaded into the sandbox (`/opt/ahd/harness/skills/`, `/opt/ahd/harness/memory.md`): skills are progressively disclosed through deepagents' `SkillsMiddleware` (only the index goes into the system prompt; the student reads full text via `execute cat`), and memory.md is injected into the system prompt through `MemoryMiddleware`; SkillsMiddleware and MemoryMiddleware run **before** the review middleware so the teacher sees the same context as the student;
  3. the bank's middleware is appended after the review middleware, followed by `StudentRequestCaptureMiddleware` (records the final model request, for audit);
  4. **bank bytes are hashed into `bank_sha256`** -- the rollout log writes `student_harness_bank.txt` recording the mounted bank's hash, for provenance only, with no assertions;
  5. when `student_harness_dir` is not passed, behavior is exactly the status quo (bare compatible).
- **The student core is not evolvable**: the `execute` tool, the student.md base prompt, the review middleware, the turn limit, and the tool_call adapter are fixed layers; evolution happens only in bank content.

## Per-round workflow

### Phase 0: determine this round's failed-task set

- Round 1: select tasks whose reward did not pass from the `<BASELINE_JOBS>` results (if the baseline does not exist, first run bare rollouts over the train task list with `<MODEL>`).
- Rounds 2/3: select tasks whose reward did not pass from the previous round's harnessed eval rollout.
- This round's swarm, bank edits, and eval rollout cover only these failed tasks. Tasks already solved leave the campaign.
- If the failed-task set is empty, end the campaign early; completed rounds still count as valid results.

### Phase 1: swarm (at most 10 coder subagents at a time; analyze and propose changes only)

Launch with AgentSwarm, item = this round's failed-task group. Each subagent does two things for its assigned tasks:

1. **Failure analysis -> `experiments/<domain>_evolution/round<N>/notes/<TASK>.md`**
   - Round 1: read the task's bare trajectory in `<BASELINE_JOBS>` (`<job>/<TASK>__*/agent/trajectory.json`, `llm_calls.jsonl`, `verifier/`), and understand how the agent worked and the failure class (insufficient exploration / mechanism misused / miscalculation / deliverable missing or misplaced / missing verification discipline).
   - Rounds 2/3: read the previous round's harnessed trajectory and attribute per the Attribution checklist section.
2. **Propose bank changes**: per Evolution discipline, decide whether the failure should be addressed by generalizing an existing component, adding a new component, or editing prompt/memory material; write the proposal and its rationale in the notes (it must argue "this helps a class of tasks"); do not write to `harness_bank/` directly.

Subagent forbidden zone: do not modify `harness_bank/`, `src/`, or `data/`; do not run rollouts; do not perform git operations.

**WARNING -- component hard rule:** middleware hooks run on the harbor event loop; sandbox probing may only use `await backend.aexecute(...)`; never call the synchronous `backend.execute(...)` inside an async hook -- internally it is `run_coroutine_threadsafe(...).result()`, which deadlocks the entire event loop when called on the loop thread (0% CPU, all trials frozen, even trial-level timeouts cannot fire). Tool function bodies run on worker threads and are not subject to this restriction; message-level checks in middleware are always the safest choice.

### Phase 2: main-agent write-up and acceptance

1. **Consolidated writing**: read all of the round's notes, resolve duplicate or conflicting proposals, and have the main agent uniformly modify the bank (tools / middleware / skills / memory.md / manifest.json).
2. **Reconciliation**: grep every reference name in manifest.json against the registered names in the registry; stale references must be zeroed out (otherwise loading fails outright).
3. **Smoke test**: import smoke (`from harness_bank.<domain>.registry import ...` + render the catalog); `load_student_harness(bank_dir)` resolves successfully (all manifest references hit, `skills/<name>/SKILL.md` files all present); `pytest tests/ -q` shows no regressions.

```
4. **Eval rollout**: submit a rollout of bare miniswe with the bank loaded, using this round
     's failed-task IDs as the exact include set (teacher passthrough, same model as the
     baseline, job name carries a date tag such as `YYYYMMDD-<domain>-evol-r<N>`). Before
     the campaign starts, review with the user once per AGENTS.md: the full train task list,
      the failure-selection criterion, the number of rounds, the fixed model and
     hyperparameters, and a directly executable command template; after user confirmation,
     later rounds proceed automatically under the confirmed rules without asking again each
     round. Re-review whenever the model, dataset, selection criterion, or hyperparameters
     change.

Reading results: the reward distribution in `runs/<job>/result.json`; per-trial details in
     `<job>/<trial>/{agent,verifier}`. Immediately after the rollout finishes, generate the
     next round's failed-task list and record the tasks that passed this round and left the
     campaign.

## Evolution discipline

1. **The lib is the only layer.** Shared mechanism -> bank; specific to one task -> do not
     write it, accept the residual failure. The one-sentence test: is this failure "shared
     by a class of tasks" or "specific to this one task" -- shared -> bank; specific -> give
      up on that task, and never write task-specific content (concrete cell addresses,
     concrete answers, steps unique to one task, concrete file names) into any shared
     component, skill, or memory. This is both overfitting prevention and the foundation of
     the claim that "the harness is generic".
2. **Generalize before adding.** If attribution points to a mechanism the bank already has
     but that does not quite fit -> generalize and improve the existing component (near-
     duplicate variants are forbidden); if it truly does not exist -> write a new component
     (first ask yourself "would this be useful on other tasks"; only if generic does it
     enter bank + manifest + registry).
3. **Prompt material is also a shared asset.** For behavioral failures (finishing without
     verification, acting before reading the task, writing the deliverable to the wrong
     place), prefer editing the corresponding `skills/<name>/SKILL.md` or appending generic
     clauses to memory.md (English, no source attribution, merged into the matching topic
     section); do not write task-specific exhortations.
4. **Anti-bloat discipline.** Near-duplicate proposals are deduplicated by the main agent
     during consolidation; component survival is adjudicated by evaluator performance --
     components with no evidence of benefit for two consecutive rounds are moved out of the
     manifest (files kept for traceability).
5. **Traces are evidence.** Every change must point back to a trajectory attribution in the
     notes; components that "feel like they should help" are not allowed.

## Attribution checklist (round >=2, go through in order, record in notes)

1. **Did the harness load?** -> In the trial log, confirm the bank_sha256 in `
     student_harness_bank.txt` matches the current bank and that the skills index appears in
      the system prompt; if not mounted: manifest/registry reconciliation or a loader
     problem.
2. **Did the student use the evolved tools?** -> Search `llm_calls.jsonl` for the evolved
     tools' names; if unused: the tool description is not discoverable or the skills index
     in the system prompt gives no guidance -- fix descriptions / material rather than
     adding new tools.
3. **Did the middleware fire?** -> If it fired but did not help: the clauses are not
     actionable enough -- rewrite abstract principles into verbatim rules; if it did not
     fire: the hook condition does not match the actual trajectory shape.
4. **Is the student dying on mechanics?** (hangs / malformed tool calls / validation loops /
      repeatedly retrying a failing tool) -> mechanics problem, go back to Evolution
     discipline and fix the bank.
5. **Dead-trial classification**: infra (does not count) vs turn wall (non-convergence,
     attribution 2/3/4) vs verifier failure (real failure -- read the verifier details and
     distinguish "wrong value / wrong location / missing deliverable / wrong format").

## Rounds and acceptance

- Run 3 rounds in total; end early if the failed-task set empties. Record per round: number
     of input failed tasks, number of tasks that passed this round and left, number of
     remaining failed tasks, mean reward (vs the bare baseline on the same task set),
     component additions/changes, recurring failure patterns, and the composition of
     abnormal trials (infra deaths vs real failures).
- Low scores in round 1 are expected (the v1 artifact comes from static analysis alone, with
      no attribution iterations); watch the trend, not the absolute value.
- Final delivery after all rounds: the reward curve, the bank's final-state inventory (
     manifest + component catalog), a map of failure patterns, open issues (including the
     list of abandoned task-specific residual failures), and suggested material for
     rewriting into a teacher-side harness.
```

Skill 1: Harness evolution skill.

### A.2 Evolved components

Table 5 lists every harness component that survives in the final $h^\star$ of each domain. SpreadsheetBench and USPTO keep their accumulated failure notes in an enabled memory component, whereas AppWorld disables the memory slot and injects the same material through the `bootstrap_instruction` middleware, which is why the table lists no memory row for it.

## B Harness adaptation

The adaptation from $h^\star$ to $\mathcal{K}$ preserves what each component does while changing how and where it acts. Table 6 summarizes the component-level mapping. Beyond individual components, the adaptation can also produce a domain prompt for the harnessing agent that states the domain's review policy.

### B.1 Reference harness inventory

$\mathcal{K}$ is mounted as a read-only `/components` directory in the harnessing agent's own workspace, which is separate from the student sandbox. Its entry point is an `index.json` that lists every readable component with an identifier, a kind, a relative path, and a one-line summary. The harnessing agent reads the index before its first decision and then opens individual component files as they become relevant. Table 7 gives the full component inventory for each domain.

Review middleware is not reachable as files. These components run automatically on every proposal and, when a condition matches, append a short block of triggered guidance to the update the harnessing agent receives. The domain prompt is appended directly to the harnessing agent's system prompt.

The adaptation is not one-to-one. For example, a student-side guard that probes the sandbox becomes a review-time condition on the visible trajectory, plus an action recipe whenever the correction itself must run in the environment. One guard can also split into several checks. USPTO's `answer_guard` is translated into separate conditions for a missing answer file, a hand-written file that bypasses canonicalization, and a commitment made without candidate enumeration.

### B.2 Worked example: preserving pre-filled cells

Many SpreadsheetBench tasks place a few already-filled example cells inside the requested answer range. Those cells are the specification. The student should infer the filling rule from them, match their format, and write only into the remaining blanks. Overwriting an example is a common failure. The evolved harness therefore inspects the sandbox at finish time, diffs the input and output workbooks inside the answer range, and rejects the finish if any pre-filled literal has changed. Code 1 shows this control flow.

```
async def on_proposed_finish(state):
    input_path, answer_range, output_path = parse_task(state)
    result = await run_in_student_sandbox(
        PREFILLED_DIFF, input_path, output_path, answer_range
    )
    if result.violations:
        reject_finish(
            cells=result.violations,
            instruction="Restore the examples and re-derive the rule."
        )
```

Code 1: Abridged student-side middleware in $h^\star$.

The harnessing agent cannot run this check privately because it has no separate interface to the student environment, and it should not have one. Otherwise the behavior patterns behind this middleware cannot be internalized. The adapted review middleware instead looks for evidence of the comparison in the student-visible trajectory. If the student tries to finish without that evidence, the middleware adds a private instruction asking the harnessing agent to replace the finish with a verification action:

```
def review(candidate, visible_trajectory):
    if not candidate.is_finish():
        return None

    evidence = scan_student_actions(visible_trajectory)
    if not evidence.has_prefilled_comparison:
        return PrivateInstruction(
```

Table 5: Component inventory of the evolved student-side harness $h^\star$ in each domain.

| Type | Component and function |
|---|---|
| | **SpreadsheetBench** |
| Memory | `memory.md`. Seven hard rules: read the skill before exploring, deliver the input workbook edited in place, treat pre-filled cells and worked examples as the specification, write computed literals, address cells by Excel coordinates, verify by assertion against independent recomputation, and remember that the deliverable is always the workbook. |
| Skill | `spreadsheet-manipulation`. Workflow and pitfall catalog for `.xlsx` editing: explore before editing, write literal values instead of formulas, read formula cells with `data_only`, normalize lookup keys, keep the target range clean. |
| Middleware | `empty_turn_guard`. Reject a finish whose assistant message carries neither a tool call nor visible content. |
| Middleware | `output_guard`. Reject a finish when no non-empty workbook exists in the output directory. |
| Middleware | `prefilled_guard`. Diff the input and output workbooks inside the answer range and reject a finish that overwrote a pre-filled literal cell. |
| Middleware | `answer_range_guard`. Mechanical health check of the deliverable range: all-empty range, formula strings, numbers stored as text, error literals, degenerate constant fills. |
| Middleware | `turn_budget`. Advise on remaining budget at turns 25 and 35, and interrupt a loop that repeats the same exception. |
| | **AppWorld** |
| Skill | `appworld-workflow`. Public-API helpers: strict parameters, per-app authentication, complete pagination, verified mutations, explicit action and answer completion. |
| Middleware | `bootstrap_instruction`. Load the workflow helper and keep one concise guide visible in the system prompt. |
| Middleware | `failure_recovery_instruction`. After a failed execution, supply concrete recovery steps for that specific failure. |
| Middleware | `context_guard`. Bound tool observations before they enter message history or the next model request. |
| Middleware | `submit_gate`. Nudge the student when an observed supervisor status reports the task unfinished. |
| | **USPTO** |
| Memory | `memory.md`. Checklist clauses in three groups. (1) *Answer discipline.* The grader matches strings exactly, so ship RDKit-canonical SMILES without atom maps, write a best-guess answer file first and refine it in place, and validate after the file's last write. (2) *Mechanical verification.* Every fragment parses, reactant heavy atoms account for every product heavy atom, and each fragment overlaps the product almost completely. (3) *Chemical priors.* The recorded set contains every species contributing heavy atoms, the stated reaction type is a hard filter over candidate transforms, and ties break toward the smallest atom edit. |
| Skill | `retrosynthesis-candidates`. Enumerate candidate disconnections for the named reaction, filter them mechanically, and commit to one; reactant-set conventions and representation pitfalls that score zero like a wrong molecule. |
| Tool | `propose_retrosynthesis`. Enumerate candidate reactant sets with a curated library of named one-step transforms. |
| Tool | `smiles_check`. RDKit-backed per-candidate scoring: validity, atom budget, maximum-common-substructure coverage, changed reaction sites. |
| Tool | `verify_answer`. Output-contract battery over the answer file, plus an RDKit-canonical rewrite of the submitted SMILES. |
| Middleware | `answer_guard`. Reject a finish when the answer file is missing, empty, or grossly malformed. |
| Middleware | `completeness_guard`. Compare product and reactant heavy-atom counts with RDKit and reject a finish whose reactant set is missing a co-reactant. |

Table 6: Component-level adaptation from the evolved student-side harness to the reference harness.

| Student-side harness $h^\star$ | Private reference harness $\mathcal{K}$ |
|---|---|
| **Tool.** Expose a utility directly in the student's action space. | **Action recipe.** Specify how to construct an equivalent action that the student can execute through $h$. |
| **Middleware.** Inspect the student's state or execution and modify or block the loop when a condition is met. | **Review middleware.** Detect the corresponding condition from the proposal and visible trajectory, then privately alert the harnessing agent. |
| **Skill.** Give the student a workflow for inspecting, editing, and verifying its work. | **Review guidance.** Use the workflow to diagnose the current step and construct a student-native replacement when needed. |
| **Memory.** Record recurring failures and the practices that avoid them. | **Failure patterns.** Turn each failure into a recognizable review condition and a targeted intervention. |

```
            decision="REPLACE",
            action="Run the pre-filled-cell comparison before finishing."
        )
```

Code 2: Abridged review middleware in $\mathcal{K}$.

The corresponding tool component in $\mathcal{K}$ takes the form of an action recipe. It contains the logic needed to generate a utility script inside the accepted student response. A shortened version is shown in Code 3. The response creates and invokes the script through $h$; its output then becomes visible to the student on the next turn.

```
cat > .agent-tools/prefilled_diff.py <<'PY'
import sys
import openpyxl
from openpyxl.utils import range_boundaries

input_path, output_path, cell_range = sys.argv[1:4]
wb_in = openpyxl.load_workbook(input_path)
wb_out = openpyxl.load_workbook(output_path)
ws_in, ws_out = wb_in.active, wb_out.active
min_c, min_r, max_c, max_r = range_boundaries(cell_range)

changed = []
for row in range(min_r, max_r + 1):
    for col in range(min_c, max_c + 1):
        before = ws_in.cell(row=row, column=col)
        after = ws_out.cell(row=row, column=col)
        if before.value is not None and before.data_type != "f":
            if after.value != before.value:
                changed.append(before.coordinate)

print("PASS" if not changed else f"FAIL changed cells: {changed}")
raise SystemExit(bool(changed))
PY
python3 .agent-tools/prefilled_diff.py INPUT.xlsx OUTPUT.xlsx B3:B40
```

Code 3: Student-native verification action generated from the action recipe in $\mathcal{K}$.

### B.3 Adapting skills and memory

Skills and memory are adapted by changing their audience. The student-side skill states the workflow as direct instructions. Its adaptation in $\mathcal{K}$ tells the harnessing agent how to recognize a missing step and realize that step as a replacement. The following condensed excerpts illustrate the change:

```
Student-side skill:
  Read pre-filled examples before editing.
  Infer the rule from them and preserve their values.
  Before finishing, compare the output with the input.

Reference-harness guidance:
  Check whether the visible trajectory inspected the examples.
  If not, replace the current proposal with a bounded inspection.
  Before accepting a finish, require an input-output comparison.
```

Table 7: Component inventory of the private reference harness $\mathcal{K}$ in each domain.

| Type | Component and content |
|---|---|
| | **SpreadsheetBench** (8 components) |
| Failure patterns | `accumulated-failures`. Recurring student mistakes on spreadsheet tasks and how to correct them, such as editing in place and preserving pre-filled cells. |
| Review guidance | `spreadsheet-manipulation`. The correct `.xlsx` editing workflow and its pitfalls; used to diagnose what the current proposal got wrong. |
| Action recipe | `prefilled-diff`. How to build a student-native command that checks whether any pre-filled example cell was overwritten. |
| Action recipe | `answer-range-check`. How to build a student-native command that sanity-checks the answer range before finishing. |
| Review middleware | `stall`. Catches proposals that declare intent without acting, and calls for a concrete action. |
| Review middleware | `error-loop`. Catches the student repeating the same failing command, and calls for consulting the documentation instead of retrying. |
| Review middleware | `turn-budget`. Reminds the harnessing agent late in the trial to steer the student toward finishing. |
| Review middleware | `finish-guard`. Catches a finish submitted without the verification steps. |
| | **AppWorld** (6 components) |
| Failure patterns | `appworld-review`. Recurring review failures on AppWorld tasks, such as inexact entities or metrics and submission without evidence. |
| Review guidance | `api-workflow`. How to work with the public APIs correctly: contracts, credentials, and pagination. |
| Review guidance | `actions-files`. How to mutate app state safely: act on original identifiers, preserve files, and verify before deleting or completing. |
| Action recipe | `public-patterns`. How to express complete pagination and app-file read-back as ordinary student `execute` commands. |
| Review middleware | `appworld-review-v1`. A battery of checks on each proposal, covering malformed responses, failed executions, premature submission, and unsafe writes. |
| Domain prompt | `teacher-prompt`. Domain review rules appended to the harnessing agent's system prompt: the student's exact action space, and how to spend interventions on substantive fixes. |
| | **USPTO** (13 components) |
| Failure patterns | `accumulated-failures`. Recurring student mistakes on retrosynthesis tasks and how to correct them, such as ranking candidates on graph evidence without chemical priors. |
| Review guidance | `retrosynthesis-candidates`. How to compare candidate reactant sets on reaction evidence instead of superficial heuristics. |
| Action recipe | `propose-retrosynthesis`. Reference implementation for enumerating candidate reactant sets. |
| Action recipe | `smiles-check`. Reference implementation for scoring each candidate mechanically. |
| Action recipe | `verify-answer`. Reference implementation for checking the output contract and rewriting the answer in RDKit-canonical form. |
| Review middleware | `candidate-format`. Catches structurally invalid proposals, which a PASS would preserve. |
| Review middleware | `command-timeout`. Catches commands that could block indefinitely, and calls for a shell timeout. |
| Review middleware | `enumeration`. Catches a hand-written answer file, the signature of skipping mechanical candidate enumeration. |
| Review middleware | `canonical-form`. Catches answer files not produced by the canonicalizing verification script. |
| Review middleware | `stall`. The same intent-without-action check as in SpreadsheetBench. |
| Review middleware | `answer-file`. Catches a finish without evidence that the answer file was written and verified. |
| Review middleware | `completeness`. Catches a finish whose reactant set may not account for every heavy atom in the product. |
| Domain prompt | `teacher-prompt`. Domain review rules appended to the harnessing agent's system prompt: the stepwise derivation every trajectory must follow, and when to intervene. |

Skill 2: Condensed skill adaptation for SpreadsheetBench.

Memory is adapted in the same way, but starts from failures observed across training tasks. For example, the student-side memory records that agents often overwrite worked examples or verify formulas only against their own outputs. In $\mathcal{K}$, these observations become review-time patterns with concrete interventions:

```
Observed failure:
  The student overwrites pre-filled examples with its inferred rule.
Review condition:
  An edit spans the answer range without preserving existing literals.
Intervention:
  Replace with inspection or restore-and-diff actions.

Observed failure:
  Verification only rereads values produced by the same script.
Review condition:
  No assertion uses worked examples or independent recomputation.
Intervention:
  Replace with an assertion-based verification action.
```

Memory 1: Condensed memory adaptation for SpreadsheetBench.

Across these adaptations, $\mathcal{K}$ retains the domain-specific condition and corrective behavior. The harnessing agent supplies the final response, and $h$ remains the only interface through which that response acts on the environment.

# C Review protocol and harnessing-agent prompt

This section details the interface between the student and the harnessing agent, and gives the system prompt used during training-data collection.

## C.1 Session structure

One harnessing-agent session covers one complete student trial. The session is incremental: the harnessing agent keeps its full history across reviews, so earlier updates, its own earlier decisions, and any component files it has read remain in context.

Each review begins with a *student update* message containing two parts. The first is the set of student-visible events added since the previous review, which on a typical turn is the observation produced by executing the previous accepted response. The second is the current unexecuted proposal, rendered as its reasoning, its visible content, and its tool call, or marked as a final response when it carries no tool call. The first update of a session additionally carries the student system prompt and the task instruction.

In addition, if the student emitted a structurally invalid response, such as more than one tool call (invalid for mini-SWE-agent) or a response with neither content nor a tool call, the update names the defect and states that passing it would preserve the defect. If a review middleware from $\mathcal{K}$ matches the current proposal, the update carries a triggered-guidance block with the middleware's evidence and hint.

## C.2 Submission format

The harnessing agent may read adapted harness components under `/components` as many times as it needs before deciding, but it must submit exactly one decision per update by calling `submit_review`. The submission has four fields the agent controls:

- `decision`, either PASS or REPLACE;
- `replacement`, the complete student response to execute instead of the proposal;
- `components_used`, the identifiers of the components that informed the decision;
- `reason`, a private justification of at most 500 characters.

A replacement is a structured object with three fields: `reasoning`, `content`, and an optional `tool_call`. For mini-SWE-agent, the only admissible tool call is a single `execute` with a non-empty command, which matches the student's action space under $h$ exactly. A replacement with no tool call and non-empty content is a final answer and ends the trial.

The schema rejects three malformed harnessing submissions: a PASS that carries a replacement, a REPLACE that does not, and a replacement with neither visible content nor a tool call. A submission that fails validation is not recorded. When an update produces no valid submission, the runtime re-prompts the harnessing agent for the same candidate, at most three times, and the trial fails if no valid submission is recorded by then. Within one trial, the number of REPLACE decisions the harnessing agent may issue is a hyperparameter. The budget is stated in the harnessing agent's system prompt, so that it can allocate interventions across the trial. We set it to 5, 3, and 1 to collect trajectories for SpreadsheetBench and AppWorld, and set it to 5 consistently for USPTO.

### C.3 SYSTEM PROMPT

Prompt 1 gives the system prompt used to collect training data. Two variants exist. The training-free evaluations in § 4.2 use a prompt with the same session structure, submission format, and budget, differing only in framing: it describes the goal as turning the components into better student actions rather than as producing a training trajectory, and it omits the paragraph on instructions regarding remaining on-policy. The oracle-answer control in § 4.4 appends one extra section granting read access to the recorded reference answer for the current task. It also instructs the harnessing agent to use it only to decide whether the student's direction can still converge, to keep every verification step in the replacement, and never to reveal that the answer is known.

```
You are a strong harnessing agent supervising a student that solves a task with one `execute
    ` tool and optional subagent delegation through `agent "<task>"` in bash.

The accepted trajectory of this trial will be used directly as SFT training data for the
    student model. Produce a successful trajectory that stays close to the student's
    behavior while using the teacher-side harness to introduce reusable task-solving
    patterns through targeted interventions. The student should do most of the work. When
    intervention is needed, replace the next response rather than taking over the task,
    then return control to the student after that response is executed.

On-policy here means that reviews occur along a live student trial and every accepted
    response uses the student's visible information, native action space, and plausible
    level of complexity. It does not mean preserving the student's current policy unchanged
    : a replacement should teach a better behavioral pattern when the harness identifies
    one that materially improves correctness, progress, recovery, or verification.

## Session input

One teacher session covers one complete student trial. Each `Student update` message
    contains the student-visible events added since the previous review and the current
    unexecuted proposal. The first update also contains the student system prompt and task.

The student sandbox is not mounted in your workspace. Treat file contents, command results,
    installed programs, and service state as known only when they appear in a student-
    visible observation.

## Components

Read `/components/index.json` before the first decision. Choose and read memory sections or
    component files as they become relevant during the trial.

Mounted middleware may append `Triggered middleware guidance` to a student update when the
    current unexecuted proposal matches one of its checks. Use the stated evidence and hint
     when reviewing that proposal.

Use the components actively as your knowledge of what good behavior looks like. They should
    affect the accepted trajectory when their guidance is relevant, not merely help you
    recognize fatal errors. Procedural knowledge from a component may inform a replacement
    even if the student has not demonstrated it yet, provided the resulting response is a
    plausible next step in the student's native interface. Component-private facts, paths,
    review records, and unsupported claims about the current sandbox must never enter the
    replacement.

## Review

Review every proposed tool call and final answer before it is accepted.

Choose `PASS` when the proposal is a sound next action and no applicable component calls for
     a meaningful behavioral correction. Pass harmless inefficiencies, stylistic
    differences, valid alternative methods, and exploratory steps that can produce useful
    evidence. Also pass recoverable mistakes when observing the result is likely to let the
     student diagnose and repair them; useful self-recovery is valuable training behavior.

Choose `REPLACE` when a meaningful correction is needed for task success or to instantiate a
     reusable pattern supplied by the harness. Common reasons include:
```

```
- a missed requirement, damaged or skipped deliverable, unsupported conclusion, or premature
      final answer;
- an unsafe, unbounded, fragile, or budget-wasting command;
- an observed failure that the student ignores or repeats without a useful change;
- a missing inspection, dependency check, test, or verification step needed to ground later
     work;
- an applicable component identifies a behavior pattern that the proposal violates or omits,
      and correcting it now would materially improve progress, recoverability, or the value
     of the trajectory.

You may replace at most `5` student responses during the complete trial. The budget rewards
     selectivity, not passivity: do not polish sound actions, but do not withhold a useful
     pattern-level correction merely because the current proposal is not immediately fatal.

When you replace:

- Make the smallest coherent change to the student's next step that installs the correction.
      This may require a different action, not merely a textual patch.
- Preserve the student's high-level intent, established facts, language, variable names, and
      command structure when they remain compatible with the correction.
- Stay near the student's demonstrated level, but you may introduce a simple procedure,
     command, library, or idiom from the components when it is needed to express the target
     pattern. Make the reasoning understandable from student-visible evidence rather than
     relying on unexplained teacher expertise.
- Correct the next decision and hand control back. Do not complete several future steps,
     precompute the task's answer, or replace work the student can perform after seeing the
     next observation.
- The reasoning must read as the student's own first-person inner monologue, continuing the
     student's current line of thought -- never as advice, critique, or correction addressed
      to the student. A natural form is the student catching its own mistake: noticing the
     constraint, re-reading the evidence, and correcting course.
- A replacement must contain complete student-style reasoning, visible content, and at most
     one `execute` call, or a final answer without a tool call. Its reasoning, content, and
     tool call must agree. Base every factual claim on student-visible events.

Do not mention the teacher, review, harness components, component names, private paths, or
     training in the replacement.

Pass a final answer only after the student-visible observations support every material task
     requirement. Otherwise replace it with the next inspection, repair, test, or
     verification action, not with a teacher-written solution that bypasses those steps.

## Submission

Call `submit_review` exactly once for each student update. For `PASS`, set `replacement` to
     null. For `REPLACE`, provide the complete replacement. Record the components you used
     and a concise reason in the private submission metadata.
```

Prompt 1: System prompt of the harnessing agent during training-data collection.

## D TRAJECTORY COLLECTION AND FILTERING

On SpreadsheetBench and AppWorld, we retain a trial only if its verifier reward is 1.0 and the run finished without an execution exception. USPTO keeps every collected trial, so the ablation in § 4.4 trains each condition on the same 500 tasks. Training on this full USPTO set also yields higher test performance than keeping successes only.

Every training trajectory must contain only student-visible text. We scan each trajectory for strings that can appear only in the private review and filter out any trajectory that contains the following.

- an absolute `/components/` path, which can only refer to the reference harness mount;
- the path of the recorded candidate file that stores the unexecuted proposal;
- the name of the review submission tool;
- the review metadata key that records which components a decision used;
- the internal name of the review middleware.

A replacement is written inside the review context, so its reasoning can slip into the reviewer's voice and describe the student's proposal from the outside. We mask the reasoning token span of any accepted response that was produced by a REPLACE decision and whose reasoning matches a reviewer-perspective pattern. The patterns cover the words *proposal* and *proposed*, references to a draft, references to the student's or the candidate's response, reasoning, command, or action, and

explicit review verbs applied to a response, such as passing or rewriting it. The resulting training sets contain 487 trajectories for SpreadsheetBench, 282 for AppWorld, and 500 for USPTO, as reported in § 4.1.

# E Implementation Details

## E.1 Student Harness

The system prompt of mini-SWE-agent $h$ is given in Prompt 2. It is the only instruction the student receives beyond the task itself.

```
You are an agent that solves tasks in a Linux sandbox.

You have one tool, `execute`, which runs one bash command. Each call starts a fresh shell,
    so working-directory and environment changes do not persist between calls.

Work on the task by issuing one `execute` call at a time. When the task is complete, return
    a short final answer without calling `execute`.

The sandbox also provides `agent "<task>"` for optional subagent delegation.
```

Prompt 2: System prompt of the student under the target harness $h$.

The harness exposes exactly one tool `execute`, whose single argument is a non-empty bash command. Each call runs in a fresh shell, so the working directory and environment variables do not carry over; state persists only through the file system. The observation returned to the student is the command's combined output followed by a line giving its exit code. A response with no tool call ends the trial and is taken as the final answer.

The sandbox additionally provides an `agent "<task>"` CLI utility for single-level subagent delegation. A subagent runs the same base model with the single `execute` tool, receives a fixed prompt instructing it to return a concise report for its bounded task, and cannot delegate further. At most four subagents may run concurrently per sandbox. This capability ensures that bash plus one-level delegation forms an action space expressive enough to map diverse evolved $h^\star$ components onto $h$. In practice, the evolved $h^\star$ in this work do not rely on subagents, and delegation is used sparingly.

## E.2 Models and Access

We access GPT-5.6 Sol (OpenAI, 2026b), DeepSeek-V4-Pro, and DeepSeek-V4-Flash (DeepSeek-AI, 2026) through Microsoft Azure. Qwen3.5-9B (Qwen Team, 2026) is evaluated as the base model through OpenRouter (OpenRouter, Inc., 2025), as is Qwen3.6-35B-A3B in the analysis of § F, and the fine-tuned checkpoints are served on NVIDIA A100 GPUs. All harness-evolution runs are performed by Kimi K3 (Kimi Team, 2026) using Kimi Code (Moonshot AI, 2026). We use the default decoding parameters provided by each model provider. Reasoning is enabled for all models, with reasoning effort set to *high* when applicable.

## E.3 Training Configuration

We perform LoRA supervised fine-tuning on Qwen3.5-9B with the Tinker supervised-training recipe (Thinking Machines Lab, 2025), using rank 32, $\alpha$=32, batch size 8, a 65,536-token sequence length, and two epochs. The learning rate follows a linear schedule with 5% warmup to a peak of $2 \times 10^{-4}$, followed by decay to $10^{-6}$. All runs use seed 42. Each domain is trained separately on its own collected trajectories. Examples longer than the sequence length are dropped.

# F Additional Analysis of Agent-as-Harness

We examine whether the advantage of agent-as-harness extends to weaker models when the same model serves as both the student and the harnessing agent. Table 8 extends the USPTO comparison of Table 1 to two additional, weaker models.

The relative advantage of agent-as-harness over meta-harness ($(h, \mathcal{K}) - h^\star$) closely correlates with model capability. It remains positive on stronger models but turns negative on weaker ones (+1.0% on GPT-5.6 Sol, +4.0% on DeepSeek-V4-Pro, -1.0% on DeepSeek-V4-Flash, and -12.0% on Qwen3.6-35B-A3B). On the weakest model, the harnessing agent intervenes aggressively, replacing 66.0% of reviewed steps, but its edits are net harmful. Even review without evolved guidance

Table 8: USPTO pass@1 (%) on two weaker models, using the four conditions of Table 1. Together with the USPTO rows of Table 1, the models span four capability tiers in decreasing order: GPT-5.6 Sol, DeepSeek-V4-Pro, DeepSeek-V4-Flash, Qwen3.6-35B-A3B.

| Model | Code-as-Harness | | Agent-as-Harness | |
|---|---|---|---|---|
| | mini-SWE-agent ($h$) | meta-harness ($h^\star$) | w/o evolved ($h, \varnothing$) | w/ evolved ($h, \mathcal{K}$) |
| DeepSeek-V4-Flash | 33.0 | **58.0** | 32.3 | 57.0 |
| Qwen3.6-35B-A3B | 16.0 | **48.0** | 12.0 | 36.0 |

falls below mini-SWE-agent (12.0 vs. 16.0). Effective review and intervention therefore require sufficient underlying model capability.

The current HARNESS-ZERO design reviews every student proposal before execution, requiring at least two model calls per step: one to generate the proposal and another to review it. Each review processes the trajectory prefix and reference harness, and must finish before the next student step, which increases both token use and latency. On USPTO, agent-as-harness with evolved guidance takes 237.2 s per trial on average, about 2.4× the 100.1 s required by mini-SWE-agent. Reducing the overhead, for example by reviewing only a subset of steps, is left to future work.

## G HARNESS-EXCLUSIVE BEHAVIOR PATTERNS

Here we detail how the behavior patterns of § 4.5 are mined and explain the mined patterns on the three benchmarks. All measurements use the trajectories of three deployments from Table 2: mini-SWE-agent ($h$), meta-harness ($h^\star$), and HARNESS-ZERO ($h$).

### G.1 APPROACH

For each domain we start from the enabled components of the student-side harness $h^\star$ and read every enabled memory file, skill file, tool, and middleware. Then, we translate each rule into a deterministic binary detector over the trajectory (agent messages, bash commands, and observations). Detectors match the target behavior semantically. For example, AppWorld pagination counts if the trajectory uses the `aw.pages` helper or an explicit loop over `page_index`; a USPTO answer write counts if it uses a shell redirect or Python `write_text`. The USPTO tools `smiles_check`, `propose_retrosynthesis`, and `verify_answer` exist only under $h^\star$. A call to them counts on $h^\star$ trajectories, and an equivalent bash or Python command counts under $h$.

For each detector, we define the support tasks as the test tasks on which, for the base model, the pattern appears under $h^\star$ but not under $h$. We keep a detector only if it has at least 10 support tasks. If two detectors give the same result on every applicable task, they are measuring the same behavior, so we merge them into one pattern. This procedure yields 18 patterns on SpreadsheetBench, 6 on USPTO, and 4 on AppWorld. A pattern's *recovery rate* is the fraction of its support tasks on which HARNESS-ZERO exhibits the behavior. By construction, the base model under $h$ and $h^\star$ exhibits it on 0% and 100% of these tasks, respectively.

### G.2 MINED PATTERNS

Tables 9 to 11 list every retained pattern.

Table 9: 4 harness-exclusive AppWorld patterns.

| Pattern | Source | Mined rule | Recovery |
|---|---|---|---|
| Retrieve complete pages | aw.py | Fetch until a terminal empty page; never rely on the implicit first page. | 21/53 (40%) |
| Read back mutations | skill | Read back affected state after a mutation. | 6/12 (50%) |
| Avoid repeated failed calls | `failure_recovery_instruction` | Do not repeatedly retry an unchanged failed call. | 22/31 (71%) |
| Choose action vs. answer | submit_gate | Use an answer only for information requests; none for actions. | 24/33 (73%) |

Table 10: 18 harness-exclusive SpreadsheetBench patterns. We gather the harness source, the mined rule (abbreviated), the support size, and HARNESS-ZERO recovery.

| Pattern | Source | Mined rule | Recovery |
|---|---|---|---|
| Edit existing workbook | memory.md | Deliver the input workbook edited in place; never rebuild it. | 31/34 (91%) |
| Assert workbook structure | memory.md | Reload and assert sheet names and headers match the input. | 20/27 (74%) |
| Protect pre-filled cells | memory.md + `prefilled_guard` | Read and preserve pre-filled cells; write only blank cells. | 24/26 (92%) |
| Write computed literals | memory.md + `answer_range_guard` | Write literal values, not unverifiable formula strings. | 19/21 (90%) |
| Avoid fragile coordinates | memory.md | Use Excel coordinates; never `chr(64+n)` or positional writes. | 19/19 (100%) |
| Save requested workbook | memory.md + output_guard | The deliverable is the manipulated workbook at the requested path. | 14/14 (100%) |
| Inspect before editing | skill | Explore sheets, headers, dimensions, and examples first. | 32/32 (100%) |
| Inspect target/examples | skill + `prefilled_guard` | Inspect the requested output area and examples before writing. | 14/18 (78%) |
| Use `solution.py` | skill | Write `solution.py` with input and output paths at the top. | 53/76 (70%) |
| Execute final script | skill | Execute `python solution.py` before finishing. | 52/75 (69%) |
| Read cached formulas | skill | Use `data_only=True` when existing formulas are inputs. | 28/43 (65%) |
| Normalize matching keys | skill | Normalize text, numeric, and date keys deliberately for matching. | 6/10 (60%) |
| Use closed target ranges | skill | Treat requested ranges as closed intervals and honor their bounds. | 30/32 (94%) |
| Reload saved workbook | skill | Reload the saved workbook and verify representative target cells. | 17/17 (100%) |
| Check output existence | output_guard | Check that the output workbook exists and is non-empty. | 16/16 (100%) |
| Populate answer range | answer_range_guard | Do not finish with an all-empty answer range. | 28/30 (93%) |
| Scan range after save | answer_range_guard | Reload and mechanically inspect answer-range cells before finishing. | 20/24 (83%) |
| Stop after delivery | turn_budget | Stop no-op verification once a valid deliverable is saved. | 21/28 (75%) |

Table 11: 6 harness-exclusive USPTO patterns.

| Pattern | Source | Mined rule | Recovery |
|---|---|---|---|
| Parse product with RDKit first | skill | Parse the product with RDKit first; never reason from the raw string. | 23/23 (100%) |
| Enumerate candidate sets | skill + `propose_retrosynthesis` | Enumerate at least 3 candidate reactant sets before committing. | 59/88 (67%) |
| Validate with RDKit | memory.md + `smiles_check` | Never eyeball-parse a SMILES; validate every structure with RDKit. | 60/60 (100%) |
| Canonicalize SMILES | memory.md + `smiles_check` | Ship every reactant in RDKit-canonical form. | 73/73 (100%) |
| Check heavy-atom coverage | memory.md + `verify_answer` | Reactant heavy atoms must account for every product heavy atom. | 40/54 (74%) |
| Validate the final answer | memory.md + `answer_guard` | Validation must cover the answer file's last write. | 45/47 (96%) |