PILOT IN THE LOOP: LIVE SELF-IMPROVEMENT FOR LONG-HORIZON AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agent runs generate experience that can improve both the current run and future runs: successful attempts reveal reusable procedures, while failed attempts expose failure modes. Most self-improvement methods process this experience only after a run ends, so they can neither recover that run nor immediately apply and validate the lessons learned, making self-improvement less efficient and less reliable. We argue that self-improvement should instead be live, using emerging experience both to redirect the active run and to update the persistent harness. However, live self-improvement exposes an architectural gap: existing agent architectures do not simultaneously support live correction and a dedicated self-improvement role. Single-agent self-correction can revise the active run, but the same agent must execute the task and judge its trajectory within a limited context, splitting attention between execution and oversight. Subagent delegation separates execution from the main agent, but the main agent typically cannot redirect the active subagent while the subagent is still running. To this end, we present **PILOT**, a supervisor–worker harness that realizes live self-improvement through two coupled mechanisms: (1) *live steering* lets a separate supervisor redirect or abort the active worker during execution; and (2) *live self-evolution* distils procedures and failure modes revealed during execution into reusable skills and memory. Across three frozen backbones and four benchmarks, PILOT ranks first in eight of twelve configurations. On Terminal-Bench 2.0, PILOT outperforms counterpart harnesses by as much as 11.8 percentage points. In the self-improvement setting, PILOT gains 14.6 points with GLM-5.1; mean output tokens fall by 42.9%, and successful evaluations per million output tokens rise by 110.3%.

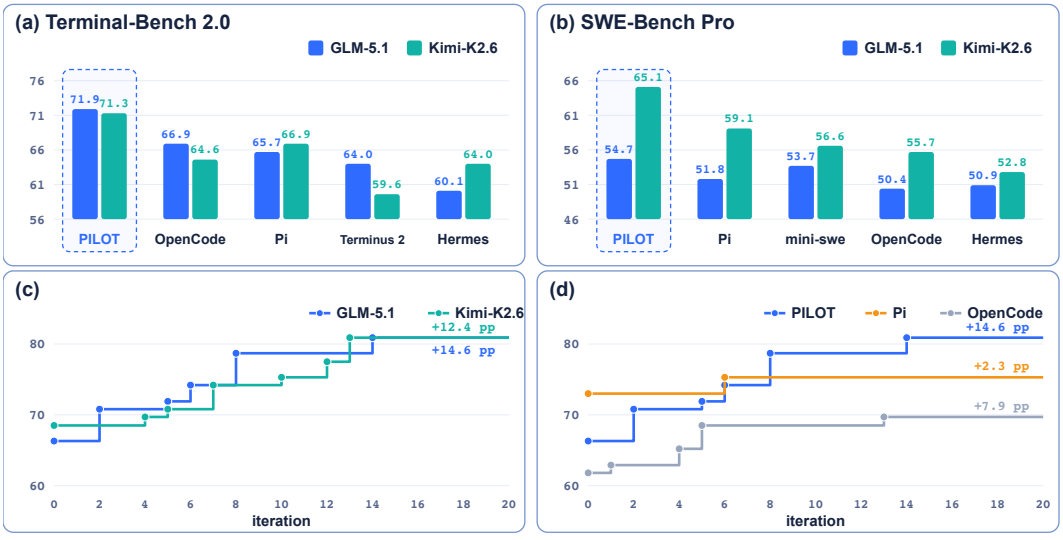

Figure 1: Main results in two evaluation settings. (a)–(b) One-shot pass rate (%) on Terminal-Bench 2.0 and SWE-bench Pro with GLM-5.1 and Kimi-K2.6; dashed boxes highlight PILOT. (c)–(d) Best-so-far Terminal-Bench 2.0 pass rate across 20 self-improvement iterations: (c) compares PILOT across backbones, and (d) compares PILOT, Pi, and OpenCode on GLM-5.1. Endpoint labels report improvement from iteration 0.

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.

1 Introduction

When agents work on long-horizon tasks, they generate experience that can improve both the active run and later work. Successful trajectories reveal procedures worth retaining as reusable skills, particularly when a task can be solved but not yet solved reliably. Failed or inefficient trajectories reveal strategies and recurring failure modes to avoid. Agent self-improvement aims to learn from such experience and improve subsequent behavior. A common approach is post-hoc review: reflection, judge-based evaluation, and self-evolving harness updates, all of which can improve the harness but cannot redirect the active run. Reflection critiques a completed trajectory, judge-based evaluation assesses the final result, and self-evolving harnesses revise prompts, skills, or memory from completed traces and feedback (Shinn et al., 2023; Zhao et al., 2024; Lou et al., 2026; Lee et al., 2026; Lin et al., 2026). These approaches can preserve experience, but their updates begin after the relevant execution has ended. Newly extracted knowledge cannot help the run that produced it, and can only be applied and validated in a later rollout or a separate evaluation. We therefore argue that self-improvement should be live, using emerging experience both to redirect the active run and to improve the persistent harness for later runs.

However, existing agent architectures do not simultaneously support live correction and a dedicated self-improvement role. Single-agent self-correction is timely, but the same agent performs the task and diagnoses its own mistakes in a context filled with execution details (Yao et al., 2023; Madaan et al., 2023; Gou et al., 2024). Execution details occupy the same context needed for diagnosis, making failing strategies harder to recognize and correct. Subagent delegation separates execution from the main agent, but the main agent commonly receives the subagent's final summary only after the delegated run returns (Fourney et al., 2024; Anthropic; Liu et al., 2026). Subagent delegation therefore cannot usually redirect the active subagent while the subagent is still running. Together, these limitations call for self-improvement that is both live and separate from task execution.

These requirements motivate *live self-improvement*: a closed loop that uses emerging experience to redirect the active run and improve the persistent harness. We present **PILOT**, a supervisor–worker harness that realizes live self-improvement through two coupled mechanisms: (1) *live steering* lets the supervisor redirect the active worker during execution; and (2) *live self-evolution* distils useful procedures, project conventions, and failure modes revealed during execution into reusable skills and memory. PILOT makes self-improvement live by separating the roles of task execution and self-improvement, correcting the current run, using the live run to assess the correction, and retaining useful experience in the harness during execution. In PILOT, a frozen *supervisor* remains connected to a *worker* throughout the worker's run rather than waiting for a completed trace. Over a live channel, the supervisor receives the worker's questions, notifications, and results, and can steer the worker or abort the worker's run in return. The worker's context absorbs execution details and dead ends, while the supervisor's separate context stays focused on the goal, recent events, and signs that the run is going off track. The supervisor can also record reusable knowledge in the persistent harness without taking task execution away from the worker.

We evaluate PILOT in one-shot, self-improvement, and experience-transfer settings. In the one-shot setting across three frozen backbones and four benchmarks, PILOT ranks first in eight of twelve combinations; on Terminal-Bench 2.0 (Merrill et al., 2026), it outperforms counterpart harnesses by as much as 11.8 percentage points. In the self-improvement setting, we organize Terminal-Bench 2.0 tasks into iterations. All tasks in one iteration start from the same harness state and receive no benchmark feedback during execution. After the iteration, the harness retains skills distilled during successful runs for use in the next iteration. PILOT's best observed pass rate rises by 14.6 percentage points with GLM-5.1 (Z.ai, 2026) and 12.4 points with Kimi-K2.6 (Moonshot AI, 2026). Across iterations, the reusable skill set grows by 21 and 31 skills, mean output tokens per evaluated task fall by 42.9% and 47.4%, and successful evaluations per million output tokens rise by 110.3% and 134.0%, respectively. In the experience-transfer setting, experience accumulated on Terminal-Bench 2.0 generalizes to LHTB, improving the GLM-5.1 backbone by up to 8.6 points.

We make three contributions: (1) we formulate *live self-improvement* as a closed loop that uses emerging experience both to redirect the active run and to update the persistent harness; (2) we introduce PILOT, a supervisor–worker harness that couples live steering with live self-evolution over a shared stream of execution experience; and (3) we evaluate PILOT across three frozen backbones

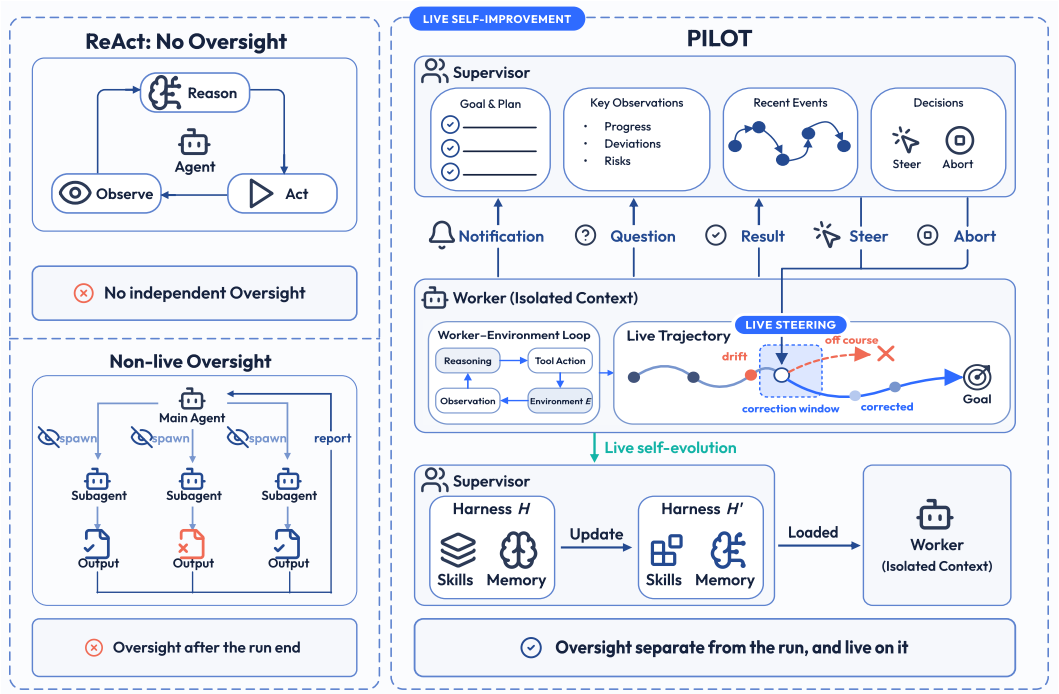

Figure 2: PILOT implements live self-improvement through two mechanisms: live steering and live self-evolution. *Top left*: a single ReAct loop has no oversight separate from the work. *Bottom left*: a main agent receives a subagent's output only after the delegated run ends. *Right*: a separate supervisor redirects or aborts the active worker, distils reusable skills and memory from supervision, and makes the evolved harness available to later workers.

and four long-horizon benchmarks, together with trajectory and efficiency analyses of task recovery, skill growth, token efficiency, and cross-benchmark experience transfer.

2 PILOT: A Live Self-Improvement Loop

PILOT treats live self-improvement as a closed loop rather than a post-hoc update: with the worker responsible for task execution and the supervisor for self-improvement, the supervisor uses *live steering* to redirect the active worker during execution and *live self-evolution* to record reusable knowledge in the persistent harness (Figure 2). Worker sessions spawned after a harness update load the evolved harness and enter the same two loops (pseudocode in Appendix A.2).

Roles and persistent state. A long-horizon task τ is attempted in a single episode in an environment $\mathcal{E}$ whose state changes with the worker's actions. The model parameters θ remain frozen, while a persistent harness H includes a skill library $\mathcal{K}$ and memory $\mathcal{M}$ that persist across episodes. Throughout this paper, *self-improvement* therefore refers to the evolution of the persistent harness H, not an update to the model parameters θ. During an episode, the supervisor can spawn one or more workers, $W_j \leftarrow \textsc{Spawn}(\theta, \tau_j, H)$, either concurrently or as earlier workers settle. Each worker loads the current harness H, operates in an isolated context, and produces a trajectory ξ_j of actions and observations in $\mathcal{E}$. The workers' exploration, dead ends, and verbose tool output remain in their isolated contexts by default. The supervisor reads the relevant portion of each ξ_j only when diagnosis is needed, preserving the supervisor's context for the goal, recent events, and recurring failure patterns.

Live steering through a two-way channel. During the episode, the supervisor remains connected to each active worker through a two-way live channel. For every worker session, the channel supports three worker-to-supervisor events and two supervisor-to-worker actions. **(1) Notification.** Each worker decides when to report progress, an intermediate result, or a potential risk; its execution con-

tinues after the notification is sent. **(2) Question.** A worker decides when its next step requires supervisory input and pauses until the supervisor replies. **(3) Result.** The runtime automatically delivers a worker's final result and worker index j to the supervisor when that worker finishes. **(4) Steer.** When live evidence indicates that a worker's current course should change, the supervisor can inspect the relevant portion of ξ_j and queue guidance for that worker's next turn; the current turn finishes first. **(5) Abort.** The supervisor interrupts an active worker W_j when continuing that worker session is no longer useful.

Live self-evolution. An active run can reveal a successful procedure or project convention worth reusing, or a recurring failure mode worth avoiding. When the supervisor identifies such knowledge in the live trajectory, the supervisor records it in $\mathcal{K}$ or $\mathcal{M}$, refining the harness from H to H'. Every worker spawned after the update, within the same episode or in a later episode, loads H'. The new worker's trajectory is again exposed to live steering and live self-evolution, closing the self-improvement loop.

Implementation. PILOT is implemented as an extension to the Pi coding-agent runtime (Zechner, 2025); the supervisor is an agent session, and workers are spawned in-process as separate sessions. Our experiments follow a real-world usage scenario in which the same frozen model serves as both supervisor and worker.

3 Experiments

3.1 Setup

Benchmarks. We evaluate on four benchmarks of long-horizon, real-world agent tasks. Terminal-Bench 2.0 contains 89 runnable tasks that require an agent to drive a shell over many steps to reach a system or engineering goal. SWE-bench Multilingual (Yang et al., 2025) extends repository-level software repair beyond Python, while SWE-bench Pro (Deng et al., 2025) contains longer and harder repository-level issues than the original SWE-bench (Jimenez et al., 2024). Long-Horizon-Terminal-Bench (LHTB) (Li et al., 2026) contains 46 terminal tasks with subtask-based grading that awards partial credit for progress toward the final goal. We report pass rate on the first three benchmarks and average normalized reward on LHTB. All scores are presented on a scale from 0 to 100.

Backbones. We use three open-weights models as frozen backbones, Kimi-K2.6 (Moonshot AI, 2026), GLM-5.1 (Z.ai, 2026), and DeepSeek-V4-Flash (DeepSeek-AI et al., 2026).[1] Within each condition, the same frozen backbone fills both the supervisor and the worker. Using the same frozen model in both roles isolates the supervisor–worker orchestration from any capability gap between the supervisor and the worker (§2).

Harness baselines. On Terminal-Bench 2.0 we compare PILOT against five single-agent harnesses driven by the same backbone: Pi, the coding agent that PILOT extends; OpenCode (Anomaly, 2025); Terminus-2 (Harbor Framework Team, 2025), maintained by the Terminal-Bench team; Hermes (Nous Research, 2025); and Meta-Harness (Lee et al., 2026), a single-agent harness optimized end to end over execution traces. On SWE-bench Multilingual and SWE-bench Pro we compare against Pi, OpenCode, Mini-SWE-Agent (SWE-agent Team, 2025), and Hermes on the same backbones. On LHTB, we compare against Pi, OpenCode, and Hermes on the same backbones. Holding the backbone fixed across harnesses isolates the effect of the harness from the effect of the model.

Evaluation settings. We evaluate the two parts of PILOT's live self-improvement loop in settings that reflect real-world agent use. (1) *One-shot setting.* A developer hands the harness a new task whose environment changes as work proceeds. Every task starts from a fresh harness state, isolating whether live steering keeps the worker on track during the run. (2) *Self-improvement setting.* A developer or team returns to the same harness for related work. To evaluate the complete loop, including whether live self-evolution turns experience accumulated during supervision into reusable harness knowledge, we organize the Terminal-Bench 2.0 runs into iterations, where one iteration is a complete sweep over the benchmark tasks. At the start of iteration i, every run receives an isolated copy of the same shared harness state H_i, which includes the skill library $\mathcal{K}_i$ and memory $\mathcal{M}_i$. While a task is running, the harness can create or revise skills and memory using only the live agent

[1] We use the DeepSeek-V4-Flash-0731 checkpoint.

Table 1: One-shot performance across four long-horizon benchmarks with three frozen backbones. We report pass rate for Terminal-Bench 2.0 and the two SWE-bench datasets, and average normalized reward for LHTB. All scores are presented on a scale from 0 to 100. *Avg.* reports the mean across the three backbones.

(a) Terminal-Bench 2.0

Harness	GLM-5.1	Kimi-K2.6	DeepSeek-V4-Flash	Avg.
Hermes	60.1	64.0	59.6	61.2
Meta-Harness	59.6	62.9	62.9	61.8
Pi	65.7	66.9	52.8	61.8
Terminus-2	64.0	59.6	**64.0**	62.5
OpenCode	66.9	64.6	59.6	63.7
PILOT	**71.9**	**71.3**	62.9	**68.7**

(b) Long-Horizon-Terminal-Bench

Harness	GLM-5.1	Kimi-K2.6	DeepSeek-V4-Flash	Avg.
Hermes	25.5	33.0	27.1	28.5
OpenCode	24.4	32.0	32.9	29.8
Pi	**31.5**	32.0	33.6	**32.4**
PILOT	26.5	**33.8**	**34.0**	31.4

(c) SWE-bench Multilingual

Harness	GLM-5.1	Kimi-K2.6	DeepSeek-V4-Flash	Avg.
Mini-SWE-Agent	68.5	73.7	77.4	73.2
Pi	69.8	**75.9**	79.4	75.0
OpenCode	68.9	74.1	83.7	75.6
Hermes	67.5	72.2	**88.3**	**76.0**
PILOT	**71.6**	73.7	81.3	75.5

(d) SWE-bench Pro

Harness	GLM-5.1	Kimi-K2.6	DeepSeek-V4-Flash	Avg.
Hermes	50.9	52.8	68.7	57.5
OpenCode	50.4	55.7	68.3	58.1
Mini-SWE-Agent	53.7	56.6	65.7	58.7
Pi	51.8	59.1	78.2	63.0
PILOT	**54.7**	**65.1**	**83.9**	**67.9**

trajectory and environment feedback. These updates occur during task execution, before the verifier outcome is available; neither the supervisor nor the worker receives any benchmark evaluation signal or reward. Thus, a candidate update is skill or memory content created or revised during a run, not a post-hoc summary generated from its evaluation result. After every run in iteration i has finished, verifier outcomes only decide which updates are carried into H_{i+1}: updates from successful runs are retained, whereas updates from failed runs are not. Verifier outcomes are never used to create or modify those updates. Every run in iteration $i+1$ then starts from the merged shared state H_{i+1}. For the backbone comparison, PILOT runs with GLM-5.1 and Kimi-K2.6 begin from the same H_0; for the harness comparison, the GLM-5.1 runs of PILOT, Pi, and OpenCode also begin from H_0, with each configuration updating independently thereafter. We measure skill-library size as the number of distinct skills in $\mathcal{K}_i$. This design mirrors a real-world use scenario in which developers explicitly instruct an agent harness to retain reusable skills across related tasks. Accordingly, the setting includes a dedicated self-improvement instruction that specifies how to reuse and record experience. Every evaluated harness receives the same instruction throughout the experiment; Appendix A.3 provides the full text. (3) *Experience transfer setting.* To investigate whether the accumulated skills and memory improve performance on new tasks, we freeze them after 20 iterations on Terminal-Bench 2.0. We then evaluate PILOT on LHTB with these skills and memory.

Protocol. Each backbone runs with the enable-thinking and preserve-thinking settings on, a maximum generation length of 32k tokens, and all other parameters left at the official defaults for that backbone. We run each configuration twice and report the mean pass rate across the two runs. Every task runs in an isolated container sandbox. On Terminal-Bench 2.0, following common practice, each task is capped at three hours of wall-clock time, with CPU, memory, and other resources left at the task's own configuration. On SWE-bench Multilingual and SWE-bench Pro, a few tasks pin environment versions that conflict with the environment some harnesses require; we apply the same exclusions to every harness. Appendix A.1 describes the exclusion criterion. For each iteration, we compute the mean output tokens generated per evaluated task; for PILOT, the calculation includes all supervisor and worker assistant turns. Figure 5(c) then averages these per-iteration means within each five-iteration window.



Figure 3: PILOT on LHTB with experience transferred from Terminal-Bench 2.0. Each panel evaluates the named backbone. *PILOT* uses a fresh harness; + *GLM-5.1 experience* and + *Kimi-K2.6 experience* add frozen experience collected with the corresponding backbone over 20 Terminal-Bench 2.0 iterations.

3.2 Live steering keeps long-horizon work on track

In the one-shot setting, we compare PILOT against the baseline harnesses with the backbone held fixed, first on Terminal-Bench 2.0 and then across backbones and task domains.

Live steering improves performance on long-horizon tasks. As shown in Table 1(a), PILOT reaches the highest one-shot pass rate on Terminal-Bench 2.0 with GLM-5.1 and Kimi-K2.6. On Terminal-Bench 2.0, PILOT outperforms OpenCode, the strongest baseline using GLM-5.1, by 5.0 percentage points. With Kimi-K2.6, PILOT outperforms the strongest baseline, Pi, by 4.4 percentage points. Across the three backbones, PILOT averages 68.7%, 5.0 points above OpenCode at 63.7%, the strongest single-agent baseline by average. Among these baselines, Meta-Harness (Lee et al., 2026) optimizes the harness post hoc using completed execution traces and evaluation feedback. PILOT outperforms this optimized harness by 6.9 percentage points on average across the three backbones.

Generalization across backbone models and task domains. As shown in Table 1, PILOT's live-steering mechanism generalizes across both backbone models and task domains. On SWE-bench Pro, PILOT ranks first with all three backbones and outperforms Pi, the strongest baseline by average, by 4.9 percentage points. On SWE-bench Multilingual, PILOT ranks first with GLM-5.1, and its average across the three backbones is within 0.5 points of Hermes, the strongest baseline by average. LHTB evaluates challenging terminal tasks with many interdependent steps, requiring sustained planning and iterative debugging. PILOT achieves the highest average reward on LHTB with Kimi-K2.6 and DeepSeek-V4-Flash. Overall, PILOT ranks first in eight of the twelve backbone–benchmark combinations, showing that the gains are not confined to a particular backbone model or task domain.

3.3 Live self-evolution makes supervision reusable across tasks

Real-world harness use spans sequences of related long-horizon tasks, so we evaluate whether live self-evolution distills experience from earlier tasks into reusable skills and memory for later ones. We first measure the complete PILOT loop across backbones, then compare PILOT with Pi and OpenCode from the same initial skill library, and finally investigate whether the accumulated experience transfers to new tasks.

The closed loop yields consistent gains across iterations and backbones. As shown in Figure 1(c), PILOT's best observed pass rate on GLM-5.1 rises from 66.3% at iteration 0 to 80.9%, a gain of 14.6 percentage points. On Kimi-K2.6, PILOT's best observed pass rate rises from 68.5% to 80.9%, a gain of 12.4 points. The harness accumulates reusable skills and memory across iterations. The consistent gains on GLM-5.1 and Kimi-K2.6 show that the benefit of closing the loop between supervision and persistent harness updates is not specific to one backbone.

Live self-evolution strengthens both skill accumulation and reuse. Figure 1(d) compares PILOT, Pi, and OpenCode using the same frozen GLM-5.1 backbone and the same initial skill library. For every task, all three harnesses receive identical user input, including the same task description

Worker trajectory before live steering. The worker computed `RowParallelLinear` with `F.linear(x, self.weight, self.bias)` and then summed the partial outputs with `all_reduce`. This added the full bias once per rank.

Live steering by the supervisor. The supervisor sent the following live correction: "Important correction for `RowParallelLinear`: The forward method should NOT include the bias parameter in `F.linear`. … The bias should be added after the `all_reduce`, not before."

Worker adoption. The worker explicitly accepted the correction: "The supervisor is correct. … the result after summing would be `world_size * bias` instead of just `bias`." The worker then changed the code to `F.linear(x, self.weight, None)`, followed by `all_reduce` and a single bias addition.

Worker trajectory after live steering. A later repair replaced the bare communication calls with autograd-compatible operations while preserving the corrected bias placement. The final implementation passed all 13 verifier tests across world sizes 1, 2, and 4, including forward and gradient checks.

Figure 4: A live-steering case on Terminal-Bench 2.0. The supervisor identifies a concrete implementation error in the active worker, and the worker adopts the correction and completes the task.

and instructions. After iteration 0, each harness incorporates experience from its own runs into its skill library. Figure 1(d) therefore measures how effectively each harness turns accumulated experience into performance on later tasks. From iteration 0 to each harness's best observed result, PILOT improves by 14.6 points, compared with 7.9 points for OpenCode and 2.3 points for Pi. The stronger gains over Pi and OpenCode show that, in the real-world self-improvement scenario, PILOT both accumulates more useful experience and applies that experience more effectively to later tasks.

Accumulated experience transfers across long-horizon tasks. To investigate whether accumulated experience generalizes to new tasks, we freeze the skill library and long-term memory after 20 self-improvement iterations on Terminal-Bench 2.0 and evaluate their transfer to LHTB. GLM-5.1 experience denotes the persistent skill library $\mathcal{K}$ and memory $\mathcal{M}$ accumulated over these iterations with GLM-5.1 as the backbone (Figure 1). Kimi-K2.6 experience is defined analogously. Figure 3 compares each backbone on LHTB with and without experience transferred from Terminal-Bench 2.0. On the GLM-5.1 backbone, transferred experience yields clear gains: experience accumulated with GLM-5.1 improves the score by 4.3 points, and experience accumulated with Kimi-K2.6 by 8.6 points. These results show that experience accumulated through PILOT's live self-evolution transfers across benchmarks, with gains depending on the receiving backbone and the source of the accumulated experience.

4 Analysis of Live Self-Improvement

4.1 Live steering concentrates on harder tasks

Section 3.2 shows that live steering improves one-shot performance on long-horizon tasks. Within the one-shot setting, we manually inspect the complete supervisor–worker trajectories to identify successful Terminal-Bench 2.0 runs aided by live steering. We classify a successful run as aided by live steering only when the trace shows that the supervisor identifies a concrete error, stalled branch, or unproductive strategy and provides corrective direction; the worker or a replacement branch follows that direction; and the worker ultimately completes the task successfully along the corrected path. An intervention followed by PASS is not sufficient: we exclude interventions that are ignored, stale, or redundant, as well as cases in which success follows an unrelated execution path. Table 2 reports the percentage of successful runs classified as aided by live steering within each difficulty group. From Medium to Hard, the percentage of steering-aided successful runs rises from 1.1% to 6.1% for GLM-5.1, from 8.1% to 19.7% for Kimi-K2.6, and from 5.4% to 6.3% for DeepSeek-V4-Flash. Across all difficulties, the percentage is 2.3% for GLM-5.1, 10.6% for Kimi-K2.6, and 5.4% for DeepSeek-V4-Flash. Trace-supported correction contributions are absent from the Easy split and are more frequent

Table 2: Successful PILOT runs aided by live steering on Terminal-Bench 2.0 (%).

Difficulty	GLM-5.1	Kimi-K2.6	DeepSeek-V4-Flash
Easy	0.0	0.0	0.0
Medium	1.1	8.1	5.4
Hard	6.1	19.7	6.3
All	2.3	10.6	5.4

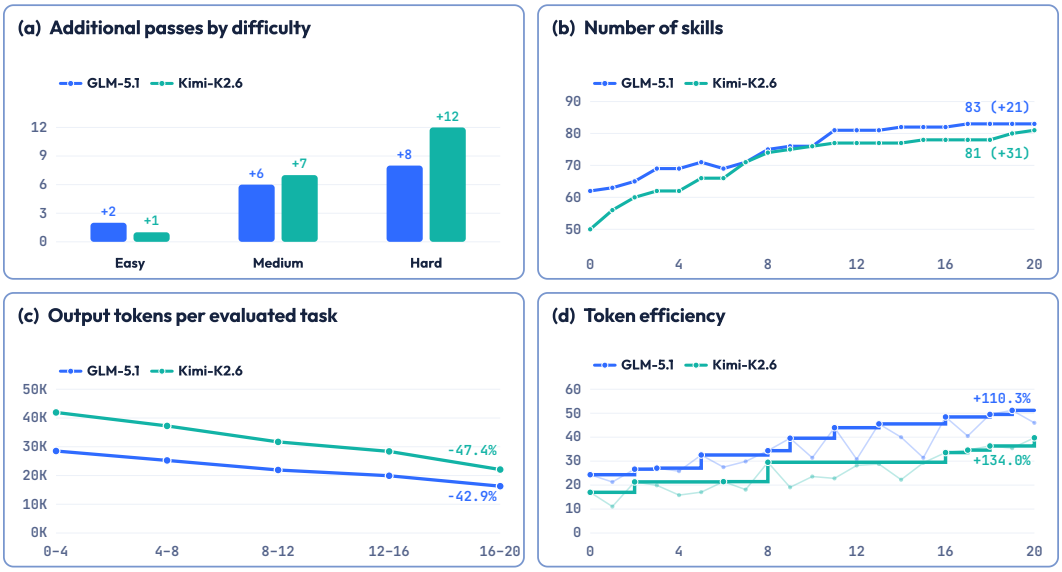

Figure 5: PILOT across iterations in the self-improvement setting. (a) Additional passing tasks from iteration 0 to the first maximum for each backbone. (b) Number of distinct skills accumulated at each iteration. (c) Mean output tokens per evaluated task, averaged over iteration ranges 0–4, 4–8, 8–12, 12–16, and 16–20; PILOT includes all supervisor and worker assistant turns. (d) Successful evaluations per million output tokens. Endpoint labels report improvement relative to iteration 0.

on Hard than Medium tasks for all three backbones. Easy tasks generally fall within the worker's existing capabilities and can be completed without external redirection. Harder tasks require longer, more fragile execution chains in which errors can compound, leaving more opportunities for the supervisor to recover the run. The pattern suggests that live steering is most useful for difficult tasks whose long execution horizon creates greater risk of drift or stalled progress.

Representative live-steering cases. Figure 4 illustrates a successful trajectory in which the supervisor identifies an implementation error and the worker follows the resulting correction. Such mid-run corrections are possible because PILOT separates execution from oversight: the worker's context stays focused on tool use and intermediate results, while the supervisor's separate view of the goal and recent events lets the supervisor detect a stalled or incorrect branch before the run is lost. The worker remains responsible for the final solution; the supervisor provides the timely direction that helps the worker recover.

4.2 Live self-evolution makes accumulated experience more useful

Section 3.3 shows that the complete PILOT loop improves Terminal-Bench 2.0 performance across iterations. We now investigate whether those gains coincide with the persistent changes expected from live self-evolution: broader task coverage, a growing skill set, and more efficient execution.

Performance gains extend across all three difficulty levels and are largest on Hard tasks. Section 3.3 showed that live self-evolution makes supervision reusable across tasks in real-world scenarios; we now further investigate how these gains distribute across difficulty levels. Figure 5(a) shows the corresponding gain for each difficulty level. GLM-5.1 gains 2 additional passes on Easy, 6 on Medium, and 8 on Hard; Kimi-K2.6 gains 1, 7, and 12 additional passes, respectively. The largest gains occur on Hard tasks. Hard tasks more often require specific procedures, recovery strategies, and tool-use patterns that the backbone does not reliably reconstruct from scratch. By distilling these patterns into reusable harness knowledge, PILOT reduces repeated exploration and makes later attempts more reliable.

Live self-evolution increases the number of reusable skills. As shown in Figure 5(b), the number of GLM-5.1 skills grows from 62 to 83, while the number of Kimi-K2.6 skills grows from 50 to 81. The consistent growth across both backbones shows that PILOT does not treat each run as an isolated task; it turns execution experience into an expanding pool of reusable procedures.

The closed loop improves token efficiency across iterations. Figure 5(c) reports mean output tokens per evaluated task, counting all supervisor and worker assistant turns and averaging the per-iteration means across five-iteration windows. The mean falls from 28.5K to 16.3K tokens per evaluated task on GLM-5.1, a 42.9% reduction, and from 41.9K to 22.1K on Kimi-K2.6, a 47.4% reduction. Figure 5(d) reports successful evaluations per million output tokens. The best observed efficiency rises by 110.3% on GLM-5.1 and 134.0% on Kimi-K2.6 relative to iteration 0. Together, the lower generation cost and higher success per token show that accumulated harness knowledge reduces repeated reasoning and exploration, allowing later tasks to better reuse established procedures.

5 Related Work

Agent systems and self-evolving agents. Language-model agents choose actions, invoke tools, incorporate environment feedback, and retain useful experience across multiple steps (Schick et al., 2023; Wang et al., 2023). Existing systems improve execution through single-agent self-correction such as ReAct, Self-Refine, and CRITIC; delegated roles in AutoGen, MetaGPT, Magentic-One, and Claude Code (Wu et al., 2023; Hong et al., 2024); or post-hoc memories derived from completed tasks in Reflexion and ExpeL. A growing line of work instead evolves persistent agent components while keeping the base model fixed. ADAS searches over agent programs, ACE evolves a context playbook (Hu et al., 2025; Zhang et al., 2026b), AutoHarness synthesizes code harnesses, and Meta-Harness and AHE optimize broader harness components from execution traces and evaluation feedback. Group-Evolving Agents shares experience across evolving agent populations (Weng et al., 2026); EvoSkill and Memento-Skills refine reusable skills, while Mem^{2}Evolve co-evolves experience and agent assets (Alzubi et al., 2026; Zhou et al., 2026; Cheng et al., 2026). Continual Harness adapts prompts, subagents, skills, and memory within a reset-free run, while Self-Harness mines weaknesses and retains validated harness edits (Karten et al., 2026; Zhang et al., 2026a). Most of these methods generate and select an improved context, agent, or harness across completed work. PILOT additionally acts on the current run: live steering redirects the active worker during execution, while live self-evolution records reusable harness knowledge during the same supervision process.

Long-horizon agent tasks. Agent benchmarks are moving from short, isolated tasks toward longer sequences of interdependent actions in stateful environments. WebArena and OSWorld introduced realistic web and desktop interaction, while SWE-bench brought repository repair into executable environments (Zhou et al., 2024; Xie et al., 2024). Terminal-Bench 2.0 extends this direction to multi-step system and engineering work performed through an interactive shell. Recent benchmarks make the horizon itself a central design axis. LongCLI-Bench evaluates sequential engineering projects with step-level scoring and finds that state-of-the-art agents pass fewer than 20% of tasks, with most runs stalling before 30% completion (Feng et al., 2026). OSWorld 2.0 extends computer-use evaluation to workflows that take people a median of 1.6 hours and require hundreds of tool calls (Yuan et al., 2026), while EdgeBench studies ultra-long real-world tasks that sustain at least 12 hours of agent–environment interaction (Zhu et al., 2026). This shift makes goal retention, interpretation of environment feedback, and recovery from compounding errors central evaluation concerns.

6 Conclusion

We presented PILOT, a supervisor–worker harness for live self-improvement: live steering redirects the active worker during execution, and live self-evolution turns procedures and failure modes observed during supervision into reusable experience. Across three frozen backbones and four benchmarks, PILOT ranks first in eight of twelve backbone–benchmark combinations, outperforming counterpart harnesses on Terminal-Bench 2.0 by as much as 11.8 percentage points and ranking first on SWE-bench Pro with all three backbones. In the self-improvement setting, its best Terminal-Bench 2.0 pass rate rises by up to 14.6 points while mean output tokens fall by up to 47.4%, and the reusable skill set grows across iterations; the accumulated experience also transfers to LHTB, improving the GLM-5.1 backbone by up to 8.6 points. Our trajectory analysis shows that live steering concentrates on harder tasks, where longer execution chains leave more room for the supervisor to recover the run. These results support live self-improvement: a closed loop that uses emerging experience to redirect the active run and improve the persistent harness for later runs.

AI use statement

During the preparation of this work, the authors used AI assistants for code development, experimental data analysis, and language editing. The authors reviewed all AI-assisted content and take full responsibility for the final paper.

Reproducibility statement

Each experimental condition uses the same frozen backbone for the supervisor and the worker, without updating model weights. Section 3.1 specifies the experimental protocol, and the supplementary material provides the supervisor implementation, harness configuration, per-task settings, self-improvement instruction, and trace-audit criteria.

References

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

Anomaly. OpenCode: The open source AI coding agent. https://github.com/anomalyco/opencode, 2025. Software repository, accessed August 10, 2026.

Anthropic. Create custom subagents. https://code.claude.com/docs/en/sub-agents. Claude Code documentation, accessed August 10, 2026.

Zihao Cheng, Zeming Liu, Yingyu Shan, Xinyi Wang, Xiangrong Zhu, Yunpu Ma, Hongru Wang, Yuhang Guo, Wei Lin, and Yunhong Wang. Mem^{2}evolve: Towards self-evolving agents via co-evolutionary capability expansion and experience distillation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20784–20831, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.952. URL https://aclanthology.org/2026.acl-long.952/.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang

Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, Jie Sun, Yang Xiao, Sizhuo Zhou, Wenxiao Wu, Yiming Liu, Pengfei Liu, Shenglin Zhang, and Kaipeng Zhang. LongCLI-bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 29952–29963. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1497. URL https://aclanthology.org/2026.findings-acl.1497/.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing, 2024. URL https://arxiv.org/abs/2305.11738.

Harbor Framework Team. Terminus 2. https://github.com/harbor-framework/harbor/tree/main/src/harbor/agents/terminus_2, 2025. Agent implementation, accessed August 10, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6507b115562bb0a305f1958ccc87355a-Paper-Conference.pdf.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github is-

sues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Zongxia Li, Zhongzhi Li, Yucheng Shi, Ruhan Wang, Junyao Yang, Zhichao Liu, Xiyang Wu, Anhao Li, Yue Yu, Ninghao Liu, Lichao Sun, Haotao Mi, and Leowei Liang. Long-Horizon-Terminal-Bench: Testing the limits of agents on long-horizon terminal tasks with dense reward-based grading, 2026. URL https://arxiv.org/abs/2607.08964.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. Dive into Claude Code: The design space of today's and future ai agent systems, 2026. URL https://arxiv.org/abs/2604.14228.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: improving LLM agents by automatically synthesizing a code harness, 2026. URL https://arxiv.org/abs/2603.03329.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong (Ryan) Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/444a3737adaee10d86ad2ef5f74468e6-Paper-Conference.pdf.

Moonshot AI. Kimi K2.6. https://huggingface.co/moonshotai/Kimi-K2.6, 2026. Official model card, accessed August 10, 2026.

Nous Research. Hermes Agent. https://github.com/NousResearch/hermes-agent, 2025. Software repository, accessed August 10, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

SWE-agent Team. mini-swe-agent. https://github.com/SWE-agent/mini-swe-agent, 2025. Software repository, accessed August 10, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing, 2026. URL https://arxiv.org/abs/2602.04837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-3239. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8b86cf5ace600c48fd188efbb8dedec8-Paper-Datasets_and_Benchmarks_Track.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. URL https://arxiv.org/abs/2606.29537.

Z.ai. GLM-5.1. https://huggingface.co/zai-org/GLM-5.1, 2026. Official model card, accessed August 10, 2026.

Mario Zechner. Pi Agent Harness. https://github.com/earendil-works/pi, 2025. Software repository, accessed August 6, 2026.

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

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8a94ff6f922d995d7d3f4ebf4143e442-Paper-Conference.pdf.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-skills: Let agents design agents, 2026. URL https://arxiv.org/abs/2603.18743.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, Zixin Wen, Zhonglin Xie, Chenhui Gou, Linxuan Ren, Yueyang Wang, Junfeng Zhong, Rui Liu, Tian Gao, Yangguang Lin, Jingyuan Zhang, Maojia Song, Xuan Qi, Jinhong Wu, Chenyang Zhang, Yinzhu Piao, Ziru Niu, Hongbin Lin, Lingxiang Meng, Peng Tang, Chengyao Tang, Shanyu Wu, Huanyu Zheng, Yu Liu, Liya Zhu, He Wang, Ming Ding, Ziyu Wan, Hao Liu, Sibo Wang, Haotian Zhu, Xintian Zhang, Nan Chai, Yipeng Liu, Panhao Lai, Sihang Yuan, Zixin Su, Ge Zhang, Wangchunshu Zhou, Yantao Du, Wenhao Huang, and Guang Shi. EdgeBench: Unveiling scaling laws of learning from real-world environments, 2026. URL https://arxiv.org/abs/2607.05155.

A Appendix

A.1 Excluded SWE-bench tasks

We exclude 43 SWE-bench Multilingual tasks and 198 SWE-bench Pro tasks whose pinned environments cannot bootstrap the JavaScript agent runtimes required by the evaluated harnesses. The SWE-bench Pro exclusions also cover task images whose Node.js versions are below the runtimes' minimum requirement. We apply the same exclusions to every harness. For reproducibility, the complete excluded-task lists are provided in the supplementary materials.

A.2 PILOT pseudocode

Algorithm 1 summarizes one supervisor session coordinating multiple worker sessions. Workers may run concurrently or be spawned later in the same episode; every event and action is associated with its worker index j. The five emphasized channel operations correspond directly to Figure 2: workers emit **Notification**, **Question**, and **Result** events, while the supervisor can issue **Steer** and **Abort** actions.

Algorithm 1 The coupled live steering and live self-evolution loops across worker sessions in PILOT.

Supervisor loop

Require: task τ, frozen model θ, harness H with skills $\mathcal{K}$ and memory $\mathcal{M}$
  repeat
    if another worker session is needed **then**
      choose an objective τ_j for the worker
      $W_j \leftarrow \textsc{Spawn}(\theta, \tau_j, H)$
    end if
    receive an event (j, e) from any active W_j
    if the event is a question **then**
      reply to W_j
    end if
    if redirection of W_j is warranted **then**
      inspect the relevant portion of ξ_j
      Steer: queue guidance for W_j
    else if continuing W_j is no longer useful **then**
      Abort: interrupt W_j
    end if
    if reusable knowledge is found in ξ_j **then**
      $H' \leftarrow \textsc{Update}(H, \xi_j)$
      $H \leftarrow H'$
    end if
    if e is a result, error, or abort **then**
      mark W_j as settled
    end if
  until the task is resolved or no further worker is needed
  return outcome and updated harness H

Worker-session loop (for each W_j)

Require: objective τ_j, frozen model θ, harness H at spawn time
  repeat
    incorporate queued supervisor guidance
    reason, act in $\mathcal{E}$, and extend ξ_j
    if the worker decides to surface an update **then**
      Notification: report progress, an intermediate result, or a risk
      continue execution
    end if
    if the worker decides supervisory input is needed **then**
      Question: send the question and pause
      receive the reply and resume
    end if
  until completion, error, or abort
  if complete **then**
    Result: the runtime sends (j, result) to the supervisor
  end if

A.3 Self-improvement instruction

The self-improvement setting prepends the same instruction to every task for every harness evaluated in that setting. Only the harness-native skill and memory paths differ. The instruction below uses `[SKILL_PATH]` and `[MEMORY_PATH]` for those substitutions.

> You already have a library of reusable skills (auto-loaded from `[SKILL_PATH]`) plus long-term notes (`[MEMORY_PATH]`), distilled from prior experience. They're available right now—use them.
>
> This library keeps growing: whatever new you save also persists, so the next run of a similar task can replay your approach instead of rediscovering it. So whenever you solve a task that took real work—finding the right tool or library, setting up an environment, working out a multi-step approach—capture that approach as a skill named after the task (e.g., "`<task-name>`"), EVEN IF it felt routine in hindsight. The test is NOT "was this non-obvious?"—it's "if a similar task came up later, would replaying this save me from figuring it out again?" If yes, save it. Record the exact tools, libraries, commands and setup you used to crack it. If solving this task also built on skills already in your library, note which ones in the new skill—so the reuse is tracked and a later run can combine them. Use EXACTLY ONE skill per task, named EXACTLY after the task name (a task "`chess-best-move`" → a skill "`chess-best-move`", not a descriptive name). Before writing a skill, check whether your library already has one for this task: if it exists, update it ONLY when this run genuinely found a better or more reliable approach (a fix, a sturdier step, a cleaner method)—if it's already good enough, leave it as-is, don't churn it; if it doesn't exist yet, create it. Never add a second skill for a task that already has one. Use `AGENTS.md` for the more general cross-task lessons. Err on the side of saving.
>
> Here's the task:

Harness	`[SKILL_PATH]`	`[MEMORY_PATH]`
PILOT	`~/.pilot/skills`	`~/.pi/agent/AGENTS.md`
Pi	`~/.pi/agent/skills`	`~/.pi/agent/AGENTS.md`
OpenCode	`~/.config/opencode/skills`	`~/.config/opencode/AGENTS.md`

A.4 Terminal-Bench difficulty breakdown

The table below reports the per-difficulty breakdown of the Terminal-Bench 2.0 results in Table 1(a); the 89 tasks comprise 4 Easy, 55 Medium, and 30 Hard tasks.

Table 3: Terminal-Bench 2.0 pass rate (%) by task difficulty. Overall pass rates are reported in Table 1. Column maxima are bold.

Harness	GLM-5.1			Kimi-K2.6			DeepSeek-V4-Flash		
	Easy	Medium	Hard	Easy	Medium	Hard	Easy	Medium	Hard
Hermes	87.5	67.3	43.3	87.5	77.3	36.7	**100.0**	69.1	36.7
Meta-Harness	75.0	64.5	48.3	87.5	71.8	43.3	75.0	72.7	43.3
Pi	87.5	72.7	50.0	**100.0**	74.5	48.3	**100.0**	**74.5**	6.7
Terminus-2	75.0	72.7	46.7	87.5	69.1	38.3	50.0	72.7	50.0
OpenCode	**100.0**	75.5	46.7	75.0	75.5	43.3	**100.0**	67.3	40.0
PILOT	87.5	**80.0**	**55.0**	**100.0**	**78.2**	**55.0**	75.0	67.3	**53.3**

Worker trajectory before live steering. The worker spent over twenty minutes tuning DAT-clear variants. The experiments repeatedly produced ties against multi-process opponents, and the required win rates remained unmet.

Live steering by the supervisor. The supervisor sent the following live correction: "Stop testing synthetic opponents and change strategy completely. … Find a published warrior source … and test it against the actual five opponents."

Worker adoption. The worker immediately acknowledged the correction: "The supervisor is right. … I need to pivot to using a proven classic warrior."

Worker trajectory after live steering. The worker consulted public CoreWars resources, adapted Silk Warrior 1.3, and tuned its replication strategy against the five actual opponents. The worker retrieved the source from the public `mbarbon/corewar-koth` archive; the trace contains no access to the Terminal-Bench repository or a benchmark solution. The final warrior passed every threshold: Stone 93%, Paper 94%, Vampire 84%, Snake 55%, and G2-Clear 53%.

Figure 6: A strategy-level live-steering case on Terminal-Bench 2.0, complementing Figure 4. The supervisor redirects an unproductive strategy, and the worker adopts the correction and completes the task.

A.5 Additional live-steering case

This section presents a second representative live-steering case (Figure 6), complementing the implementation-level correction in Figure 4.