DON'T FORGET WHAT YOU JUST LEARNED: SELF-DIRECTED REPAIR MAKES AGENT SKILLS PERSIST

Anonymous authors
Paper under double-blind review

ABSTRACT

Intelligent agents should learn from their mistakes so experience compounds into continuous improvement. Yet today's agents recover from a mistake only temporarily: in-context reflections are easily forgotten across sessions and compete for limited context, while full-parameter fine-tuning is expensive and can interfere with existing capabilities. We introduce Self-Direction, a mechanism for agents to learn from mistakes that combines the benefits of temporary retries in-context and policy updates. When an agent fails a task, it reflects on its own trajectory in order to extract a lesson and immediately distills it into a lightweight low-rank adapter. By keeping the original weights frozen, Self-Direction preserves existing capabilities while only training 0.02% additional parameters for each adapter. The agent then saves these adapters into a persistent skill library, automatically routing them to solve related problems later. Experiments with open agents based on the Qwen-family across open-ended research (GAIA), tool-use workflows (Claw-Eval), and terminal tasks (Terminal-Bench 2.1) show our method improves overall success rates by 8.9 to 17.0 percentage points and performs competitively with frontier systems. We find Self-Direction avoids repeating mistakes across future encounters of a task, nearly doubling the reliability of an agent's corrections and enables transfer of learned skills with just a handful of adapters generalizing to dozens of new tasks without additional training. Ultimately, this self-directed repair transforms everyday mistakes into lasting, reusable capabilities.

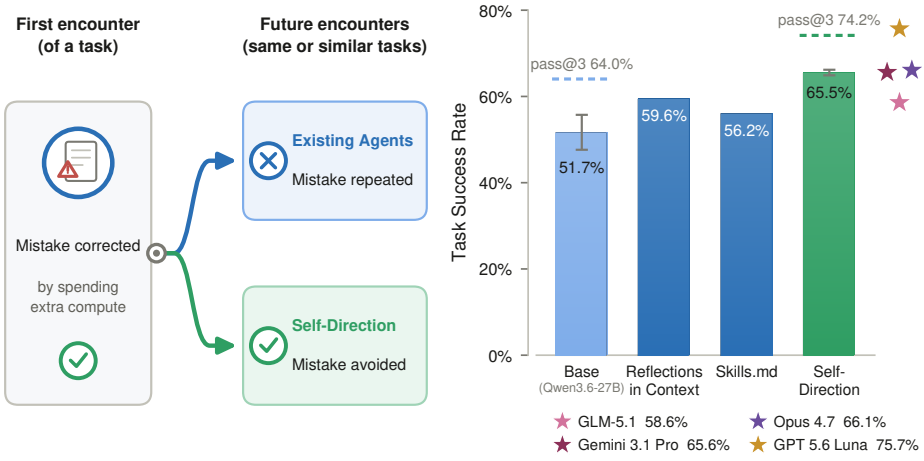


Figure 1: Self-Direction **turns an agent's own failures into reusable LoRA skills. Left:** Current agents fail to reliably reuse lessons learned by spending extra resources correcting a mistake. Self-Direction enables storage and reuse of skills to avoid repeated mistakes. **Right:** On Terminal-Bench 2.1, Self-Direction lifts performance from 51.7% to 65.5%, outperforming existing baselines and frontier models[1].

[1]Frontier model numbers were obtained from https://www.tbench.ai/?version=2.1.

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

Modern agents can often correct their mistakes and complete the task at hand. Yet, this recovery is ephemeral: In a new session, the agent is just as likely to repeat the same mistake (Rabanser et al., 2026; Gonzalez-Pumariega et al., 2026). As agents tackle longer-horizon workflows, each additional step introduces another pitfall for a mistake. Even frontier agents make mistakes across tool use, coding, and general assistant tasks (Yuan et al., 2026; Bandi et al., 2026). Repeatedly rediscovering correctable lessons is costly, leads to inconsistent performance, and leaves the potential for continuous improvement unrealized. Our goal is not just to fix an agent's mistake in the current session, but to *learn persistent lessons to avoid repeating the mistake in the future*.

Existing approaches encapsulate lessons either in context or model weights, but enable only part of the desired behavior. Context-based methods, e.g., demonstrations, prompt optimization, reflection, hard-coded skill files, and persistent agent harnesses, can change behavior quickly during deployment (Shinn et al., 2023; Majumder et al., 2024; Zhao et al., 2024). They enable agents to accumulate experience without access to model parameters, but their lessons remain external: They must be retrieved when relevant, compete for limited context, and may not reliably influence behavior on related future tasks. What is missing is a continuous mechanism to fix mistakes in the current session and avoid repeating mistakes in future sessions, without impacting existing capabilities.

To address this gap, we introduce Self-Direction, a mechanism for agents to learn from their mistakes. After a failed execution, the agent examines its trajectory to extract a lesson and then distills the lesson into a light-weight Low-Rank Adaptation (LoRA) repair adapter. Self-Direction turns feedback from the environment or a user of a mistake into a LoRA lesson that can be employed immediately to retry the current task and retained for future use. Self-Direction uses a router to determine if an existing lesson is relevant to the current task based on similarity of the current task to previously seen tasks, enabling lessons to be used on future tasks. Since each lesson is encoded as an adapter around the frozen model, existing capabilities are preserved. Furthermore, as each lesson only requires a modest memory (e.g., $\sim$ 85MB for Qwen3.6-27B), the lessons can scale with the agent's experience with the help of a growing skill library. An overview of Self-Direction is presented in Fig. 3.

We evaluate Self-Direction on open-weight agents across terminal coding, tool-use, web search, and personal assistant tasks. We find:

1. Self-Direction **increases success across task types.** By repairing mistakes, Self-Direction raises task success rate by 17.0 percentage points on GAIA (Mialon et al., 2023), 13.8 points on Terminal-Bench 2.1 (Merrill et al., 2026), and 8.9 points on Claw-Eval (Ye et al., 2026) (tool-using workflows), outperforming in-context reflections and distilling reflections in `Skills.md`. On Terminal-Bench 2.1, official scores show that Self-Direction with a 27B-parameter Qwen agent outperforms GLM-5.1, a model with approximately $30\times$ more parameters, and performs competitively with frontier closed-weight systems including Claude Opus 4.7 and Gemini 3.1 Pro (see Section 4).
2. Self-Direction **prevents repeated mistakes across future encounters of a task.** Self-Direction repairs are reliable across repeated encounters. It increases recovery from a mistake over three encounters by $2\times$ compared to retrying by a base agent. On Claw-Eval, Self-Direction raises pass^3, which measures success in all three independent runs, from 51.6% to 64.6%, reaching a competitive performance with frontier agents that use GPT 5.4, Kimi-K2.6, and Gemini-3.1 Pro (see Section 4.2).
3. Self-Direction **generalizes to new tasks without sacrificing existing capabilities.** With a frozen library of just three adapters, occupying 256MB of memory in all, Self-Direction raises task success on GAIA from 36.4% to 52.7%. Skills learned exclusively from easier GAIA tasks improve success on unseen, harder tasks by 10.5 points, whereas three adapters learned from Claw-Eval operations tasks raise success on workflow category tasks from 75.0% to 87.5%, without additional training. Notably, Self-Direction enables these gains without interfering with existing capabilities (Section 4.3).

Together, these results bring open-weight agents closer to continuous improvement, where lessons accumulate with continued use.

2 Related work

After failing to complete a task, an agent could simply re-run the same task (Brown et al., 2024) or spend additional inference-time compute (Snell et al., 2025). Both methods improve the fraction of tasks solved by at least one attempt. However, once the episode ends, they discard the effort spent recovering from failure, thus are just as likely to repeat the same mistake. Methods that do retain lessons differ mainly in where the lesson is stored: in the agent's context, its harness, or its weights.

Lessons kept in context. The most direct way to preserve a lesson is to write it down. Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), CLIN (Majumder et al., 2024), and Voyager (Wang et al., 2024) store verbal reflections, causal abstractions, or executable skills in episodic memory to condition later trials. This enables rapid adaptation without parameter updates but leaves lessons external to the policy: Reuse requires repeatedly retrieving them into a finite context window. This process is fragile, as performance is sensitive to prompt composition (Lu et al., 2022) and models use relevant information less reliably in longer contexts (Liu et al., 2023). Appendix A further discusses the expressive limits of context-based adaptation.

Lessons kept in the harness. An orthogonal approach modifies the harness rather than the model: the prompts, tools, and memory through which it acts (Chen et al., 2026; Ning et al., 2026). Recent systems search over agent designs or self-improve by rewriting their code in response to environmental feedback (Hu et al., 2025; Zhang et al., 2026c; Robeyns et al., 2025). Closest to our setting, Zhang et al. (2026b) let a fixed model mine its failure traces to propose minimal, regression-tested harness edits. We share this failure-driven, teacher-less trigger but instead incorporate repairs into the policy itself. Harness-only methods change *how* a fixed policy is invoked, not *what* it has learned; gains remain in the scaffold, leaving the underlying model no more capable after deployment. Appendix A reviews self-synthesizing and evolving harnesses more broadly.

Lessons kept in weights. Parameter updates write lessons directly into the policy, though full fine-tuning can interfere with existing capabilities (Luo et al., 2023). Parameter-efficient methods, such as LoRA (Hu et al., 2022), constrain these updates, motivating approaches that convert failures into weights. FailForge (Lv et al., 2026) and SKILL0 (Lu et al., 2026), however, typically rely on global fine-tuning, privileged reference solutions, or stronger external teachers. Test-Time Training (TTT) instead adapts parameters online to the current input (Sun et al., 2020; Akyürek et al., 2025) but applies updates unconditionally and discards them after the episode. Self-Direction differs from both: Triggered only by a verified failure, it derives supervision from the agent's own failed trajectory without a stronger teacher, then uses a targeted adapter (Ren et al., 2026) to store each repair as a routable entry in a persistent skill library. Appendix A compares other parameter-efficient, self-rewarding, and TTT methods.

3 Learning from Mistakes

Consider an agent governed by an initial policy π_0 deployed across a task set $\mathcal{T}$. During execution, the tasks on which the agent fails are denoted by a mistake subset $\mathcal{T}_{fail}$. When an agent encounters a task $t \in \mathcal{T}_{\text{fail}}$, the goal is to repair the immediate mistake while updating the system so that similar mistakes are avoided in the future and base capabilities are preserved.

3.1 Existing Paradigms for Repairing Mistakes

Motivated by current mechanisms for repairing mistakes outlined in Section 2, we consider four strategies. They differ in where the lesson is stored once the episode ends and a failure occurs: nowhere, in the context, in the weights, or in the harness (see Fig. 2).

- **Resampling** *(lesson not stored)*: Agents spend extra test-time compute by drawing multiple independent samples from the unmodified base model π_0 to hopefully generate a new, correct rollout (Muennighoff et al., 2025). No lesson is stored, even in the current session: Each agent action is selected from the same policy, π_0.
- **Reflect In-Context** *(lesson stored in current session's context)*: Reflecting on mistakes and inserting verbal critiques, error rationales, or raw failed trajectories into the prompt context has been shown to be an effective method for boosting agent performance (Shinn et al., 2023; Majumder et al., 2024). By self-reflecting and loading the procedural per-task skills in-context, agents can avoid mistakes when retrying the task. However, by default, the lesson is forgotten once a new session without the current context begins.

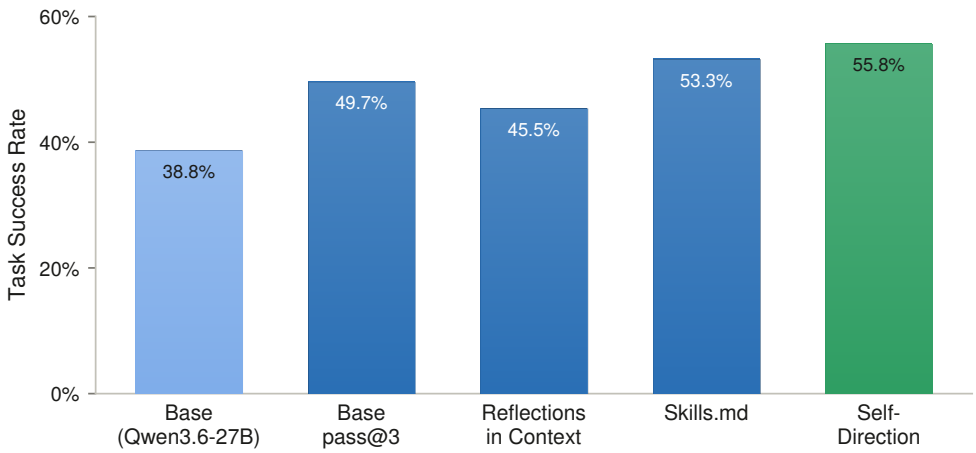


Figure 2: Self-Direction **outperforms alternative mistake-repair strategies on GAIA.** SVD-initialized LoRA achieves success on $92/165$ tasks (55.8%), outperforming Retrying (pass@3), Reflections in Context, and Accumulating Reflections in Skills.md. Self-Direction recovers 28 mistakes, yielding a 17.0-point gain over the initial base execution. We also compare our results with Failed Trajectory in Context and doubling the tool call budget in Appendix C.2.

- **Reflect In-Weights** *(lessons persist across sessions in-weights)*: As an alternative to loading procedural per-task reflections in context as textual skills, agents can incorporate the same reflections in-weight through a lightweight adapter, which enables the lessons to persist in the policy and can lead to corrective behavior.
- **Accumulating a `Skills.md`** *(lessons are accumulated across sessions in a text file)*: Instead of relying on an external mechanism to determine relevant in-context reflections for a task and retrying, agents can learn from mistakes by distilling and accumulating reflections across sessions and tasks in a shared markdown text file, often called `Skills.md`. Agents can load the relevant `Skills.md` in-context when the same or similar tasks are encountered.

We compare these approaches using open-weight agents across general assistant, coding, and tool-calling agentic tasks.

3.2 Experimental setup and Evaluation on GAIA

Experimental setup. To evaluate mechanisms for repairing mistakes under controlled conditions, we standardize our primary agent architecture on open-weight models from the Qwen family. The main part of this article contains results for Qwen3.6-27B (Qwen Team, 2026a). The agent operates inside a ReAct interaction loop using standard benchmark system prompts outlined in Appendix G (Yao et al., 2022). Appendix F lists all hyperparameters, Appendix G lists the prompts, and Appendix D repeats the loop on the stronger Qwen3.8-27B (with additional results for Qwen3.5-9B) (Qwen Team, 2026b). Open-weight models enable us to directly compare in-weight versus in-context reflections, described in Section 3.1, while keeping the agent's base model fixed.

Metrics. Beyond accuracy, denoted by the *Task Success Rate*, *Pass@k and pass^k* separate coverage from consistency across k independent executions of the whole pipeline: pass@k counts a task solved in at least one run, pass^k only when solved in every run.

Evaluating assistant, tool-calling, and coding agents. To cover a range of agentic capabilities, our evaluation spans three distinct domains — **GAIA** (Mialon et al., 2023): 165 validation tasks across three difficulty levels, solved with web search, page retrieval, Python, a shell, and file inspection, and verified by matching a short final answer against a reference; **Claw-Eval** (Ye et al., 2026) that measures tool-using workflows over mock email, calendar, task-tracking, and business services, verified from the whole trace (tool calls, audit logs, and produced artifacts) together with an LLM judge: Hence, a task can fail because of the actions the agent took even when its final message looks correct; and coding via **Terminal-Bench 2.1** (Merrill et al., 2026): 89 tasks that drive one persistent shell inside a fresh container, verified by hidden tests run after the attempt. We provide

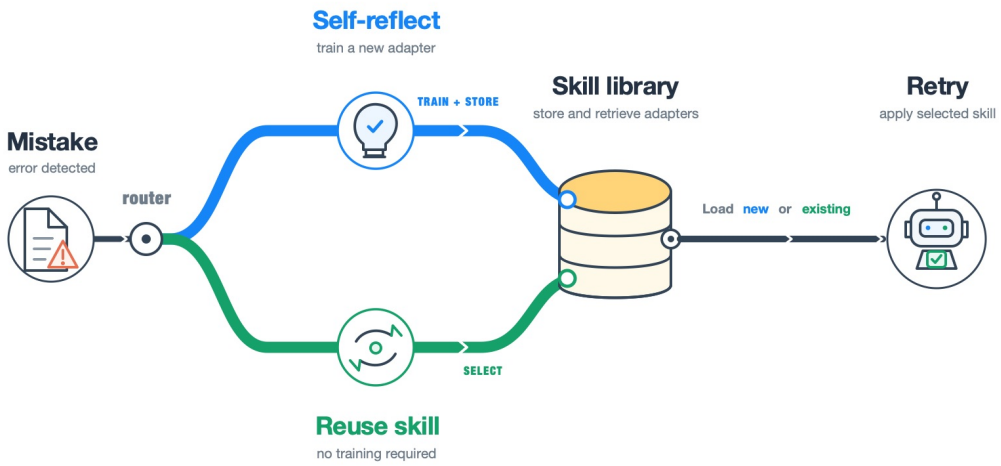


Figure 3: Self-Direction **pipeline overview.** On detecting a mistake, the agent either self-reflects on the failed trajectory and trains a new LoRA adapter, or retrieves one the skill library already holds; the selected adapter is then loaded and the task retried.

additional experiments for browser agents on MiniWob++ in Appendix B. Additional details on benchmark-specific tool calls are presented in Appendix F.4.

In this section, we employ GAIA as a diagnostic testbed to motivate our pipeline design, reserving comprehensive cross-domain evaluations on Claw-Eval and Terminal-Bench 2.1 for Section 4.

Findings on GAIA. We evaluate task success across standard retrying, in-context reflections, and in-weight parametric updates on the 165 GAIA validation tasks (see Fig. 2).

Retrying boosts coverage but lacks persistence: Resampling from the base model by doing parallel rollouts is effective for immediate task recovery, yielding a 10.9 pp gain with pass@3 compared to the initial base execution. However, retrying leaves model parameters untouched; once an episode ends, the agent remains just as likely to repeat the same mistake when encountering a similar task in future sessions. We also compare our results with Failed Trajectory in Context in Appendix C.2.

In-Weight updates outperform in-context reflections: To enable lessons to persist, we compare adding failure reflections in-context versus incorporating them in-weights. Encoding reflections in-weights achieves higher task success than loading them in-context ($+10.3$ pp). As mistakes accumulate, we find that encoding reflections into a single LoRA adapter is not as effective as individual LoRA adapters per task (see Fig. 14), suggesting that LoRA updates per task are effective in capturing repairs without using global parameter updates. A qualitative example of a successful recovery from a mistake on a Level 1 task is presented in Fig. 15.

Harness summaries (Skills.md) underperform and constrain context budget: As execution failures accumulate, storing textual reflections in context hits severe scaling limits. Storing full failure trajectories and reflections across GAIA tasks exceeds Qwen3.6-27B's native context window by over $10\times$ (Appendix E.2). Distilling reflections into a high-level Skills.md file avoids context overflow but task performance drops below retaining reflections in-weight (see Fig. 2).

Across 165 GAIA tasks, Self-Direction improved accuracy from 38.8% to 55.8%, a gain of 17.0 points (paired task-level t-test: $(t(164) = 5.79), (p = 3.50 \times 10^{-8})$; 95% CI: $[11.2, 22.8]$ pp). We also find that loading an incorrect or mismatched adapter degrades performance below the clean base policy (Appendix E.1), motivating us to include a `router` to select an adapter relevant to the task at hand based on the similarity of the current task with the previously trained adapter. Additional details about the t-test are presented in Appendix C.9.

3.3 The Self-Direction Loop

Motivated by these diagnostic findings, we introduce Self-Direction: a mechanism that enables agents to learn from a failed execution by reflecting on its mistakes and distilling the resulting lessons into a LoRA adapter, which can later be loaded for similar tasks (see Fig. 3). This enables

learning from mistakes to scale while preserving the agent's existing capabilities (see Section 4.3). Self-Direction operates in two main modes:

Training repair adapters. When an agent encounters a failure, it performs self-directed reflection to synthesize a concise set ($N \leq 24$) of procedural instruction-response examples. Crucially, the model acts as its own teacher without relying on external supervisor models and does not require full rollouts; the procedural examples are behavioral corrections (see Appendix I). These examples are distilled into a lightweight LoRA adapter. To maintain numerical stability and non-interference, adapter matrix A is initialized from the leading right-singular vectors of the targeted base weights and frozen, whereas matrix B (initialized to zero) is optimized (see Appendix J). Each adapter trains $\sim 0.02\%$ additional parameters, enabling continuous accumulation in a persistent skill library $\mathcal{L}$ with a modest memory footprint ($\sim$ 85MB for Qwen3.6-27B). Training a new adapter takes ~ 1.5 minutes with our setup on an H200 GPU, demonstrating how adapters can be trained along with solving the current task. Additional details about the whole reflection, synthesis and training steps are presented in Appendix F.3.

Reusing adapters. When the same or a similar failed task is encountered, Self-Direction uses a sentence-embedding router[2] (Reimers & Gurevych, 2019) to compare the incoming task with the task descriptions stored alongside each skill in $\mathcal{L}$ (the similarity is 1 if a new adapter is trained for the current task). The corresponding adapter is then loaded into the frozen base model for re-execution.

Base retry fallback: Motivated by the benefits of resampling, if an adapter-guided retry fails, Self-Direction unloads the adapter and executes a final attempt on the base model with a prompt that indicates that a prior attempt failed (Appendix G). This fallback maximizes immediate recovery compute while isolating parametric repairs from compounding errors.

4 Results

Beyond GAIA, we expand our evaluation to coding, tool-calling assistant tasks, and browser agent tasks. We find Self-Direction can repair mistakes across diverse settings, avoid repeating mistakes in the future, and scale learning while preserving existing capabilities, with repair adapters that generalize to unseen task categories and difficulties.

4.1 Self-Direction repairs mistakes across various task types

First, we measure whether Self-Direction improves task success across diverse agentic environments. We compare it to the base agent over three independent runs across Claw-Eval, GAIA, Terminal-Bench 2.1, as well as browser gym tasks from MiniWob++. We find Self-Direction improves task success across all task types. On Claw-Eval, mean success increases from $64.4\% \pm 3.6\%$ to $73.3\% \pm 2.8\%$, a gain of 8.9 percentage points (Fig. 4). On Terminal-Bench 2.1, mean success increases from $51.7\% \pm 4.1\%$ to $65.5\% \pm 0.6\%$, a gain of 13.8 points (Fig. 1). On GAIA, the gain is 17.0 points (Fig. 2). We observe additional gains compared to the base agent on MiniWoB++ browser tasks (Appendix B.1).

Self-Direction **outperforms repair baselines and is agent-rollout efficient.** We also compare Self-Direction with other strategies, such as adding reflection in-context or distilling reflections in a `Skills.md` file. As shown in Fig. 4, Self-Direction outperforms both settings. In addition, we find that Self-Direction compares favorably when controlling compute in terms of the number of rollouts (see Fig. 10); e.g., Self-Direction achieves 55.8% task success on GAIA with only 2.10 agent rollouts per task, matching the base agent's pass@6.

Self-Direction **narrows the gap to reported frontier agents.** For Claw-Eval and Terminal-Bench 2.1, we also compare Self-Direction with officially reported numbers for frontier agents. We find Self-Direction with a 27B parameter Qwen model achieves a competitive task success rate. E.g., on Terminal-Bench 2.1, Self-Direction outperforms GLM-5.1, a model with $30\times$ more parameters, and is competitive with closed-weight models, such as Opus 4.7 and Gemini 3.1 Pro (see Fig. 1). Similarly, on Claw-Eval, Self-Direction outperforms GLM-5.1 and is competitive with Gemini 3.1 Pro and Claude Opus 4.7 (see Fig. 4).

The gains extend across GAIA difficulty levels (Appendix C.2), Claw-Eval task categories and execution structures (Table 5), and Terminal-Bench 2.1 difficulty levels (Table 6). The gains also

[2]We use the sentence-transformers/all-MiniLM-L6-v2 embedding model (`https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`).

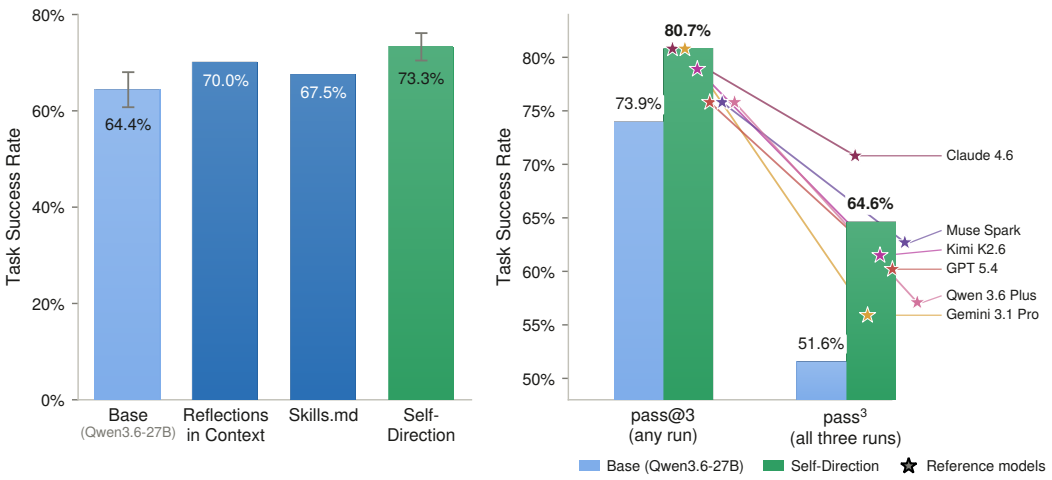


Figure 4: **Self-Direction improves task success across distinct agentic environments. Left:** Across three runs, Self-Direction improves mean success on Claw-Eval from 64.4% to 73.3% and outperforms other baselines. Error bars denote one sample standard deviation; cross-run pass@3 and pass^3 results are reported in Appendix C.3. Terminal Bench 2.1 and GAIA results are reported in Figs. 1, and 2, respectively. **Right:** Self-Direction enables a 27B model to outperform several larger frontier models across pass@3 and pass^3 metrics for Claw-Eval tasks.[3]

extend to agents with Qwen-3.5 and Qwen-3.8 model variants (Table 4 and Appendix D). Together, these results demonstrate that Self-Direction improves diverse agents across open-ended research, tool-using workflows, terminal coding, and browser interaction.

Next, we study whether a repair persists across repeated encounters of a task.

4.2 Learned skills prevent repeated mistakes

After repairing a mistake, we study how consistently Self-Direction prevents repetitions of the same mistakes in the future. We evaluate whether a repair is merely the result of a lucky retry or a persistent skill in two ways. First, we measure whether a repair remains effective over future repeated encounters of a task. Second, we assess variability in how often a learned skill repairs a mistake across multiple independent runs.

Self-Direction **avoids repeating mistakes across future encounters of a task.** Across 185 prior mistakes from the three benchmarks, Self-Direction nearly doubles the fraction solved consistently across three encounters: pass^3 increases from 9.7% to 18.9% (+9.2 pp) across GAIA, Terminal-Bench 2.1, and Claw-Eval (see Table 1). At each encounter, both base retrying and Self-Direction receive up to two full rollouts. We report *mistakes avoided*, pass^k, as the fraction of prior mistakes solved in every encounter through encounter k. We also report *corrections preserved*, pass^k/pass^1, as the fraction of first-encounter successes that remain successful through encounter k. More importantly, after Self-Direction repairs a mistake once, 76.9% of the repairs remain effective through a second encounter and 67.3% through a third; versus only 52.4% and 42.9% for the base agent.

In Fig. 5, we illustrate the jagged performance of the base agent on Terminal-Bench 2.1 across three independent runs, which range from 48.3% to 56.2%; Self-Direction improves every run with gains ranging from +9.0 to +18.0 pp. Notably, pass^3, which requires a task to succeed in all three runs, increases from 34.8% to 57.3% (+22.5 pp). We show a similar plot for eight runs in Section C.4.

Therefore, Self-Direction spends compute not just on repairing the mistake now, but on enabling the skill to persist and avoid mistakes consistently across future task encounters.

Next, we ask whether a growing skill library preserves the agent's existing capabilities.

[3]Frontier model numbers are quoted verbatim from original benchmark numbers at https://claw-eval.github.io/.

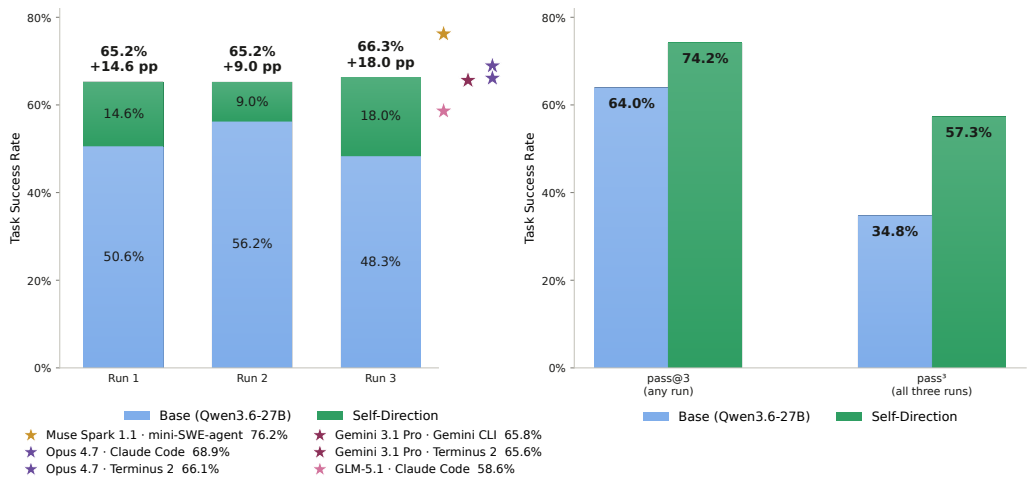


Figure 5: **Self-Direction normalizes the variable performance of the base agent. Left:** Base agent performance varies within the same benchmark in the exact same setting, ranging from 48.3% to 56.2%. Self-Direction helps maintain reliable performance across all runs. **Right:** Self-Direction increases pass@3 from 64.0% to 74.2% and pass^3 from 34.8% to 57.3%. Reference frontier scores are provided for context to highlight differences between different harnesses and agentic settings.

Table 1: **Self-Direction prevents repaired mistakes from recurring.** Learned skills increase success across repeated encounters and preserve repairs more reliably than base-model retrying. Results cover 101 GAIA, 44 Terminal-Bench 2.1, and 40 Claw-Eval prior mistakes.

Benchmark	**Encounters**	**Mistakes avoided** (pass^k)			**Corrections preserved**	
	k	**Base retry**	**Self-Direction**	**Gain**	**Base retry**	**Self-Direction**
GAIA	2	9.9%	21.8%	**+11.9 pp**	45.5%	78.6%
	3	6.9%	19.8%	**+12.9 pp**	31.8%	71.4%
Terminal-Bench 2.1	2	13.6%	20.5%	**+6.8 pp**	50.0%	69.2%
	3	13.6%	15.9%	**+2.3 pp**	50.0%	53.8%
Claw-Eval	2	15.0%	22.5%	**+7.5 pp**	75.0%	81.8%
	3	12.5%	20.0%	**+7.5 pp**	62.5%	72.7%
Total	2	11.9%	21.6%	**+9.7 pp**	52.4%	76.9%
	3	9.7%	18.9%	**+9.2 pp**	42.9%	67.3%

4.3 Self-Direction learns new skills without impacting existing capabilities

Continual skill acquisition is useful only if learning a new skill does not disrupt the agent's existing capabilities. Self-Direction limits this interference in two ways. First, it routes every task to the unchanged base model for the initial attempt; adapter retrieval occurs only after a base agent makes a mistake. Second, each learned skill is stored as a separate lightweight adapter, leaving the base model and previously learned adapters frozen. Learning a new skill, therefore, cannot overwrite previously acquired behavior.

On a retry, Self-Direction loads an existing adapter only when its task goal is similar to that of an existing adapter, with threshold τ. An agent with $\tau = 1$ will load an existing adapter only if the same task is encountered again; otherwise, the agent will train a new adapter. In this setting, Self-Direction trains new adapters for each new task but perfectly preserves existing capabilities by only loading adapters for the task on which they were trained. When $\tau < 1$, Self-Direction loads adapters for similar, but not necessarily, identical task, enabling reuse of adapters across new tasks. We find that empirically, except for low thresholds $\tau < 0.30$, the router preserves or improves

existing capabilities even with a skill library containing full adapters from the two other unrelated benchmarks. The detailed findings are presented in Appendix C.8.

Next, we study a simple strategy of selecting the most similar adapter, without a fixed threshold. We find that selecting the most similar adapter leads to generalization across new tasks.

4.4 Learned skills transfer to unseen tasks across difficulties and types

Training a new skill for each failure may not always be practical. Here, we ask whether skills acquired through Self-Direction transfer to unseen tasks. We explore three types of transfer: to unseen tasks in a continual deployment stream, across unseen harder tasks, and to unseen task categories.

A three-skill library boosts task success across GAIA. We simulate a deployment stream by randomly interleaving all 165 GAIA tasks across all levels. The agent begins with an empty skill library and applies Self-Direction to learn new skills. With the library capped at three adapters, the complete Self-Direction loop increases GAIA task success from 36.4% to 52.7%, a gain of 16.3 points, as shown in Table 3. The complete three-adapter library occupies only 256 MB, showing that useful experience can be retained with a modest incremental memory footprint. We report additional experiments with a frozen library of six/twelve adapters and interleaved tasks in Appendix C.7.

Skills learned on easier tasks transfer to harder tasks. We study how well lessons learned on easier Level-1 GAIA tasks transfer to unseen, harder Level-2 tasks. Specifically, we train 28 adapters on failed Level-1 tasks, freeze the resulting library, and evaluate it on 86 unseen Level-2 tasks. We find that reusing Level-1 lessons recovers nine failures, increasing overall accuracy from 40.7% to 51.2% – a gain of 10.5 percentage points. The base agent achieves a pass@2 rate of 43% on this subset, demonstrating that the learned repairs remain effective even on more challenging tasks.

Skills transfer across related task categories. Finally, we test whether skills cross category boundaries within Claw-Eval. We freeze a library of three adapters learned exclusively from operations tasks and apply it to 24 unseen workflow tasks that failed base agent execution (Table 3). The base agent solves 18 tasks. On the six remaining failures, the operations library solves three, recovering half of the failures without training on a single workflow task. This raises accuracy from 75.0% to 87.5%, a gain of 12.5 percentage points. Qualitative examples of these transfers are presented in Appendix H.

Collectively, our findings present Self-Direction as a promising foundation for agents that improve through continual use. Across research, tool-use, browser, and coding environments, Self-Direction repairs mistakes now, prevents repeating mistakes in the future, preserves existing capabilities, and generalizes to new tasks of greater difficulty and related task categories. Self-Direction offers a path towards agents whose deployment experience compounds over time—where each mistake can contribute not only to solving the task at hand, but to building lasting capabilities.

5 Conclusion

Enabling lessons to persist in an ever-growing library of skills holds promise for scaling continuous self-improvement of agents. Lessons, which are encoded as LoRA adapters, offer a practical approach for storing a library of skills that can be shared across deployments and domains. Besides self-improvement, Self-Direction may also support specializing models for tailored applications, personalization, and patching of security vulnerabilities. Even though the current instantiation of Self-Direction focuses on in-weight repairs, the lessons can be complementarily instilled in self-evolving harness or in 'Skills.md' files used in-context. Future research can explore how to intelligently select the most suited medium to enable a given lesson to persist as well as opportunities for composing multiple lessons when faced with a mistake.

Limitations. Self-Direction begins with a mistake, which must be identified through self-evaluation, grader feedback, or a user. It then retains the resulting lesson. Encoding that lesson in model weights requires GPU resources and storage capacity beyond standard inference to train LoRA adapters. Textual reflections can instead be stored and accumulated until sufficient resources become available. Compared with in-context or textual lessons, which transfer across model families and remain human-interpretable, in-weight repairs avoid consuming the limited context window but remain model-specific.

AI USE STATEMENT

In this work, we used generative AI tools to refine the writing, provide feedback, discover related work, and support the implementation of experimental code. We did not use generative AI tools to formulate the research ideas or design the experimental settings: These were designed by the authors. Other required disclosure tasks are not applicable. All AI-assisted text and code were reviewed, verified, and tested by one author. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We support reproducibility by describing the experimental setup and evaluation methodology in the main part of this article. The Appendix provides a complete configuration of our agentic setup, including the exact hyperparameters and prompts used in all experiments. Together, these sections contain the information necessary to reproduce our experimental results.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2026. Oral. arXiv:2507.19457.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In *Proceedings of the International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 942–963. PMLR, 2025.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. Learning from mistakes makes LLM better reasoner, March 2024. arXiv:2310.20689 [cs.CL].

Anthropic, Jun 2026. URL https://claude.com/blog/complete-guide-to-building-skills-for-claude.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, S. K. Hong, Youngjune Gwon, and Sungjin Ahn. Understanding LoRA as knowledge memory: An empirical analysis. In *Proceedings of the International Conference on Machine Learning*, June 2026.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, et al. MCP-Atlas: A large-scale benchmark for tool-use competency with real MCP servers, 2026. arXiv:2602.00933 [cs.SE].

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. HarnessX: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

Gen Dong, Yanjie Gao, Liqun Li, Tianyin Xu, Yu Hua, and Fan Yang. Agent skills can be harmful: An empirical study of skill-induced failures in LLM agents. *arXiv preprint arXiv:2608.11888*, 2026.

Gonzalo Gonzalez-Pumariega, Saaket Agashe, Jiachen Yang, Ang Li, and Xin Eric Wang. On the reliability of computer use agents. *arXiv preprint arXiv:2604.17849*, 2026.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv: 2308.08998*, 2023.

Sharut Gupta, Phillip Isola, Stefanie Jegelka, David Lopez-Paz, Kartik Ahuja, Mark Ibrahim, and Mohammad Pezeshki. ReasonCache: Teaching LLMs to reason without weight updates. *arXiv preprint arXiv:2602.02366*, 2026.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *Proceedings of the International Conference on Learning Representations*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations*, October 2022.

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

Pengcheng Jiang, Zhiyi Shi, Kelly Hong, Xueqiang Xu, Jiashuo Sun, Jimeng Sun, Hammad Bashir, and Jiawei Han. Harness-1: Reinforcement learning for search agents with state-externalizing harnesses. *arXiv preprint arXiv:2606.02373*, 2026.

Seth Karten, Alex L. Zhang, Kevin Thomas, Sebastian Müller, and Prime Intellect Team. Prime agent: A self-improving RLM harness. Prime Intellect Blog, August 2026. URL https://www.primeintellect.ai/blog/prime-agent.

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, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, November 2023. arXiv:2307.03172 [cs.CL].

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. *arXiv preprint arXiv:2603.03329*, 2026.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity, March 2022. arXiv:2104.08786 [cs.CL].

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization, April 2026. arXiv:2604.02268 [cs.LG].

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning, January 2023. arXiv:2308.08747 [cs.CL].

Dongyi Lv, Fushun E, Aichen Cai, Liang Huang, Ya Zhang, Qiuyu Ding, Canhui Wu, Zhi Wang, Yuesong Zhang, Jiaqi Wang, and Nan Duan. FailForge: Distilling procedural competence from persistent failures into code agents, August 2026. arXiv:2608.08570 [cs.AI].

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*, May 2023.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. In *Proceedings of the Conference on Language Modeling*, 2024.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *Proceedings of Advances in Neural Information Processing Systems*, April 2024.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, 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 H. S. 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 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, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces, January 2026. arXiv:2601.11868 [cs.SE].

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-Pro: Learning reusable skills from experience via non-parametric PPO for LLM Agents. In *Proceedings of International Conference on Machine Learning*, February 2026.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants, November 2023. arXiv:2311.12983 [cs].

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, February 2025. arXiv:2501.19393 [cs].

Jun Nie, Yonggang Zhang, Qianshu Cai, Yiu-ming Cheung, Xinmei Tian, and Bo Han. EvolveNet: Collaborative harness evolution for agent self-improvement. *arXiv preprint arXiv:2608.04968*, 2026a.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. TTHE: Test-time harness evolution. *arXiv preprint arXiv:2607.08124*, 2026b.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

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

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026a. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. *arXiv preprint arXiv:2602.16666*, 2026.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Tao Ren, Weiyao Luo, Hui Yang, Rongzhi Zhu, Xiang Huang, Yuchuan Wu, Bingxue Chou, Jieping Ye, Jiafeng Liang, Yongbin Li, et al. Scaling self-evolving agents via parametric memory. *arXiv preprint arXiv:2606.04536*, 2026.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Biswa Sengupta and Jinhua Wang. Harbor: Automated harness optimization. *arXiv preprint arXiv:2604.20938*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *Proceedings of the International Conference on Learning Representations*, 2025.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR, 2020.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling agent experience into persistent knowledge for skill evolution, August 2026. arXiv:2608.27454 [cs.AI].

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. *Transactions on Machine Learning Research*, October 2024. arXiv:2305.16291 [cs.AI].

Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, Dongruo Zhou, et al. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic LLM agents. *arXiv preprint arXiv:2605.22166*, 2026.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-Eval: Towards trustworthy evaluation of autonomous agents, May 2026. arXiv:2604.06132 [cs.AI].

Xinlei Yu, Gen Li, Qingyi Si, Guibin Zhang, Yuqi Xu, Congcong Wang, Shuai Dong, Kaiwen Tuo, Xiangyu Zeng, Kaituo Feng, Qunzhong Wang, Yang Shi, Xiaobin Hu, Xiangyu Yue, Jiaqi Wang, and Shuicheng Yan. DOPD: Dual on-policy distillation, June 2026. arXiv:2606.30626 [cs.AI].

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, et al. Osworld2. 0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. arXiv:2606.29537 [cs.AI].

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Proceedings of the International Conference on Machine Learning*, March 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Proceedings of Advances in Neural Information Processing Systems*, May 2022.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, et al. JIT-Agent: Scaling harness intelligence via just-in-time harness evolution. *arXiv preprint arXiv:2608.25593*, 2026a.

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.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. In *Proceedings of the International Conference on Learning Representations*, 2026c.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Expanded Related Work

Lessons kept in context. Expanding on the context-based methods discussed in Section 2, systems frequently rely on iterative self-correction without training. For instance, Self-Refine allows a single model to critique and revise its own output (Madaan et al., 2023), whereas GEPA turns this reflective step into an optimizer by attributing outcomes to specific prompt content and evolving candidate pools (Agrawal et al., 2026). Maintaining textual skills has also appeared in deployed systems as markdown skill files (Anthropic, 2026), with recent work co-evolving such skills alongside persistent knowledge bases for cross-model transfer (Tang et al., 2026). Other approaches, like Skill-Pro, refine a skill memory using a PPO-style objective while keeping the underlying model frozen (Mi et al., 2026). However, relying on externalized skills has significant operational downsides. Retrieved skills have been shown to introduce functional failures and efficiency regressions even when they appear relevant (Dong et al., 2026). Furthermore, the limitation is fundamentally expressive: Gupta et al. (2026) argue that concatenating exemplars is only a shallow form of adaptation, as the context can only reach the internal states that a literal prompt induces through frozen weights.

Lessons kept in the harness. Expanding on the harness-based methods discussed in Section 2, there is extensive debate regarding how to best represent an agent's harness (Pan et al., 2026; Chen et al., 2026; Wang et al., 2026) and what elements the harness should own at runtime (Jiang et al., 2026). While early work focused on meta-agents programming new agents in code (Hu et al., 2025), recent systems have heavily specialized this self-evolving loop. For instance, Lou et al. (2026) synthesize code harnesses automatically, Sengupta & Wang (2026) treat harness configuration as a direct optimization target, and Lee et al. (2026) optimize a weaker agent's harness using a stronger external one. Lin et al. (2026) pair every harness edit with a prediction that is later verified against task outcomes. Beyond single-episode optimization, Nie et al. (2026a) distribute this process across deployments and Nie et al. (2026b) evolve harness code over an unlabeled test stream, though adaptation is scheduled per batch rather than conditioned on a verified failure. Other approaches synthesize harnesses just in time (Zhang et al., 2026a), adapt the model-environment interface offline (Xu et al., 2026), compose harnesses into multi-day development loops (Yan et al., 2026), or expose this evolving surface directly in deployed systems (Karten et al., 2026). Ultimately, all of these methods accumulate a lineage of scaffold code rather than a library of parametric repairs.

Lessons kept in weights. Expanding on the parameter-based methods discussed in Section 2, though low-rank adaptation reduces the risk of catastrophic forgetting, empirical analysis shows that routing among several adapters can still underperform an oracle (Back et al., 2026). Variations like PiSSA initialize an adapter from the principal singular components of a pretrained weight and train those components, refactoring the stored weight into a residual (Meng et al., 2024). In contrast, we freeze the spectral factor and train only the zero-initialized coefficient matrix, leaving base weights completely untouched such that the adapter remains detachable. This follows the initialization introduced for online parametric memory by Ren et al. (2026), though their adaptation is triggered by context-budget overflow and evolves a single intra-episode state rather than a retrievable library.

Furthermore, the training signal for in-weight learning is typically gathered in offline batches, as seen in STaR and ReST (Zelikman et al., 2022; Gulcehre et al., 2023). Methods that learn specifically from mistakes usually rely on a stronger corrector, such as GPT-4 (An et al., 2024), or the model judging itself (Yuan et al., 2024). When a teacher is given privileged information, the resulting performance gap conflates true transferable capability with information asymmetry (Yu et al., 2026). Closest to our setting, FailForge converts persistent failures into a training signal, but uses a stronger teacher, a reference patch as privileged input, and a single global fine-tune after which the textual skill is deliberately discarded (Lv et al., 2026). Finally, Test-Time Training (TTT) is related to our in-weight updates during goal execution. TTT adapts parameters on the current test input under a self-supervised objective (Sun et al., 2020), on text retrieved from neighbors before being reset (Hardt & Sun, 2024), or through a per-task adapter trained on rule-based augmentations (Akyürek et al., 2025). In each of these cases, adaptation is transient and unconditional, whereas our method conditions strictly on verified failures and preserves the repair for future reuse.

B Additional Browser and Computer-Use Benchmarks

We further evaluate whether Self-Direction extends to environments in which success depends on interacting with graphical interfaces rather than producing a textual answer. We consider MiniWoB++, which contains focused browser-control tasks. We retain the same fixed base model, reflection procedure, adapter training configuration, and adapter-first retry with base retry fallback used in the main experiments.

The benchmark provides the agent with a textual accessibility tree containing stable element identifiers. The agent receives no screenshots or visual pixels. The available tools support observing the page, executing BrowserGym actions, clicking, typing, selecting options, pressing keys, scrolling, waiting, and signaling completion.

B.1 MiniWoB++

Many MiniWoB++ categories are already saturated by the base agent and provide little opportunity for learning. We, therefore, focus on four categories with meaningful remaining headroom. These include a harder multi-target social-media task, collapsible interface elements, and two variants of email-inbox interaction. Simpler categories, such as `navigate-tree` and `multi-layouts`, were excluded because the base model already solves nearly all evaluated instances.

Across the selected categories, the common initial base execution solves $42/100$ tasks. Three independent base attempts increase coverage to $55/100$, whereas one execution of the full Self-Direction loop achieves $54/100$. Matching the base pass@3 on these categories, this provides additional evidence that the same mechanism to learn from mistakes can extend to browser interaction and benefit from persistence, reuse of skills, which standard resampling does not offer (see Table 2).

Table 2: **Additional results on browser use environments.** Self-Direction enables meaningful gains in tasks involving browser use.

Benchmark	Task group	Tasks	Initial base (Qwen-3.6 27B)	Base pass@3 (no lesson retention)	Full loop (lessons retained)
MiniWoB++	`social-media-all`	25	$6/25$ (24%)	$11/25$ (44%)	$10/25$ (40%)
MiniWoB++	`click-collapsible-2`	25	$8/25$ (32%)	$11/25$ (44%)	$11/25$ (44%)
MiniWoB++	`email-inbox`	25	$15/25$ (60%)	$18/25$ (72%)	$18/25$ (72%)
MiniWoB++	`email-inbox-nl-turk`	25	$13/25$ (52%)	$15/25$ (60%)	$15/25$ (60%)
MiniWoB++	**Overall**	**100**	$42/100$ **(42%)**	$55/100$ **(55%)**	$54/100$ **(54%)**

C Expanded Results Across Benchmarks

The main text reports pooled and average results to emphasize the central empirical findings. Here, we present the stratification across GAIA difficulty levels and repeated executions of Claw-Eval and Terminal-Bench 2.1. These expanded results test whether the reported improvements are concentrated in particular task groups or isolated runs.

C.1 Self-Direction trains skills that can be reused for future tasks

We find that Self-Direction enables successful transfer of learned skills to future related, unseen tasks as described in more detail in Section 4.4.

Table 3: Self-Direction **transfers learned skills to unseen tasks.** Stored skills generalize to new tasks in a continual deployment stream, harder tasks, and related task categories.

Generalization	Skills learned from	Library	Unseen evaluation tasks	Improvement
Tasks in a continual stream GAIA	3 GAIA tasks	3 adapters	Remaining GAIA tasks ($n = 162$)	**+16.3 pp** 36.4% → 52.7%
Harder tasks GAIA	Easy Level 1 tasks	28 adapters	Harder Level 2 tasks ($n = 86$)	**+10.47 pp** 40.7% → 51.2%
Tasks across categories Claw-Eval	Operations tasks	3 adapters	Unseen workflow tasks ($n = 24$)	**+12.5 pp** 75.0% → 87.5%

C.2 GAIA Results by Difficulty Level

The pooled GAIA result presented in Fig. 2 summarizes how different recovery strategies respond to an initial failure. Fig. 6 expands this comparison across GAIA Levels 1-3. We compare Base resampling, Base resampling with double the tool call budget, Failed Trajectory in Context, Reflections in Context, reflections accumulating in Skills.md, and SVD-initialized LoRA using the strategies defined in Section 3. Note that all settings use upto 50 tool call steps. The 100-step ablation demonstrates how sequential test-time scaling enables performance jumps. It should not be interpreted as a fair rollouts-matched comparison to other strategies and is reported for reference.

SVD-initialized LoRA improves performance at every difficulty level. Starting from the initial base execution, the full loop increases accuracy from $25/53$ to $35/53$ on Level 1, from $31/86$ to $46/86$ on Level 2, and from $8/26$ to $11/26$ on Level 3. Pooled across all levels, modular adapter skills recover 21 failures and the base retry fallback recovers seven more, producing the headline result of $92/165$ (55.8%).

C.3 Expanded Per-Run Results on Claw-Eval and Terminal-Bench

The aggregate results presented in Figs. 1 and 4 represent pooled results across all three runs. Here, we report all three complete executions of Claw-Eval, all eight of Terminal Bench 2.1, together with task-level pass@k and pass^k. Pass@k measures coverage – whether a task succeeds in at least one run – whereas pass^k measures consistency by requiring success in all three runs.

The frontier-model reference points shown in Figs. 5 and 7 are reproduced from the corresponding public benchmark harnesses and should be interpreted as system-level results rather than controlled comparisons of the underlying models alone. Agent performance can vary substantially with the surrounding scaffold, including its prompts, tool interfaces, context management, stopping criteria, and retry policy. In contrast, all of our experiments use the same deliberately minimal ReAct harness described in Section 3, enabling the effect of the Self-Direction loop to be isolated within our setting.

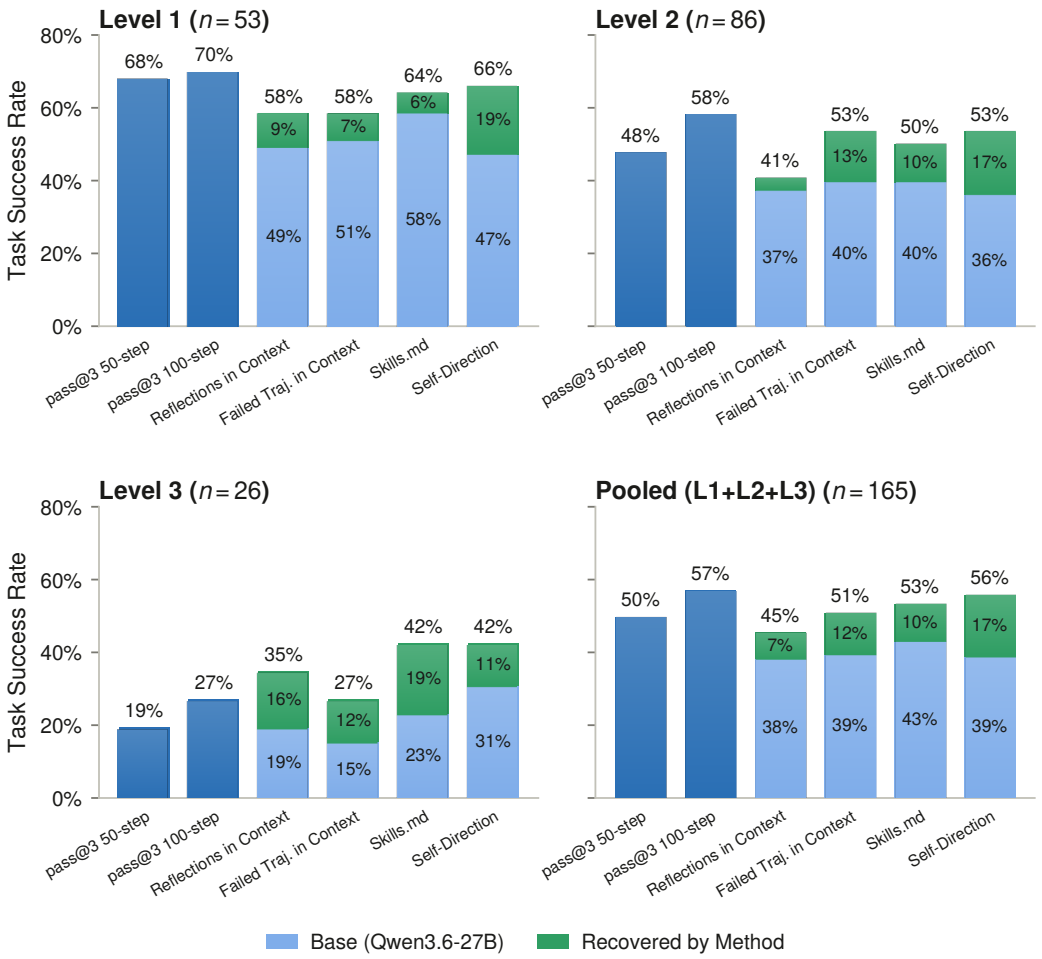


Figure 6: **Expanded GAIA recovery results by difficulty level.** Each panel compares base resampling, Failed Trajectory in Context, Reflections in Context, reflections in Skills.md, and SVD-LoRA. Stacked bars distinguish initial base successes from tasks recovered by each strategy. SVD-LoRA improves performance on all three levels and reaches $92/165$ tasks overall.

These comparisons, therefore, provide useful performance context but do not attribute differences solely to model capability. More broadly, sophisticated continual harnesses and Self-Direction are complementary: Harnesses improve how a fixed model is orchestrated, whereas Self-Direction gives an open model a mechanism to convert failures into persistent, modular in-weight skills. The same learning loop could consequently be incorporated into richer agent harnesses.

Claw-Eval. The full loop improves accuracy in every Claw-Eval run, with gains ranging from $+4.3$ to $+15.5$ percentage points (see Fig. 7). Across the 161 tasks, pass@3 increases from 73.9% to 80.7%, whereas pass^3 increases from 51.6% to 64.6%. Self-Direction, therefore, improves both the range of tasks the agent can solve and the set it solves consistently. We also report appropriate frontier model comparisons for context; the results are reported as is from the official harness used by the original benchmark authors (Ye et al., 2026).

C.4 Terminal-Bench Reliability Across Eight Runs

Three runs establish that the improvement is repeatable, but a larger campaign provides a stronger view of run-to-run reliability. We, therefore, execute the complete loop eight times on the same 89 Terminal-Bench 2.1 tasks. Self-Direction improves accuracy in all eight runs, with gains between

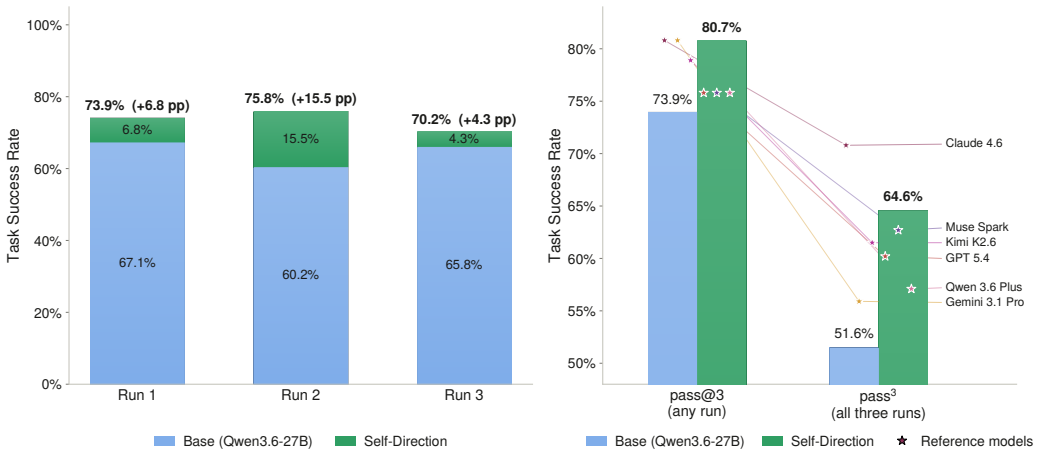


Figure 7: **Expanded three-run results on Claw-Eval.** The left panel decomposes each run into initial base successes, additional full-loop recoveries, and unresolved tasks. The right panel reports cross-run coverage (pass@3) and consistency (pass^3). The full loop improves both metrics and produces a positive accuracy gain in every run. Reference-model scores are included for context and may use different agent configurations.

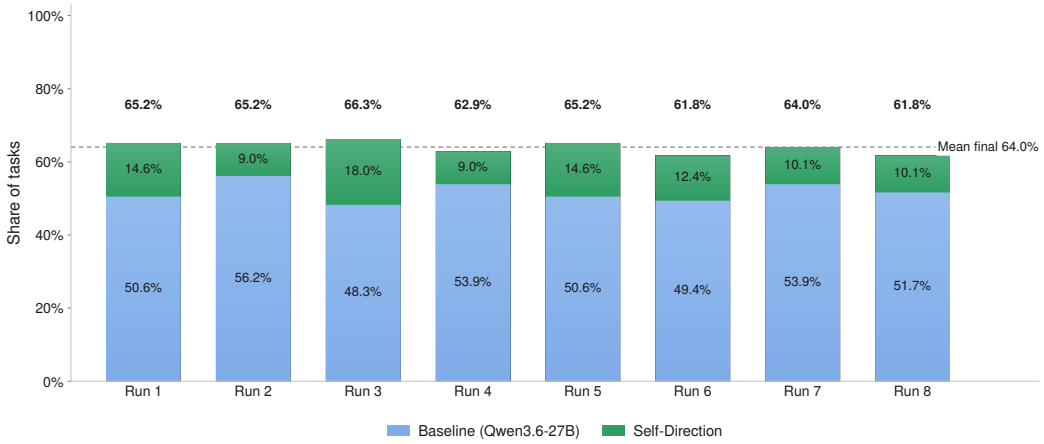
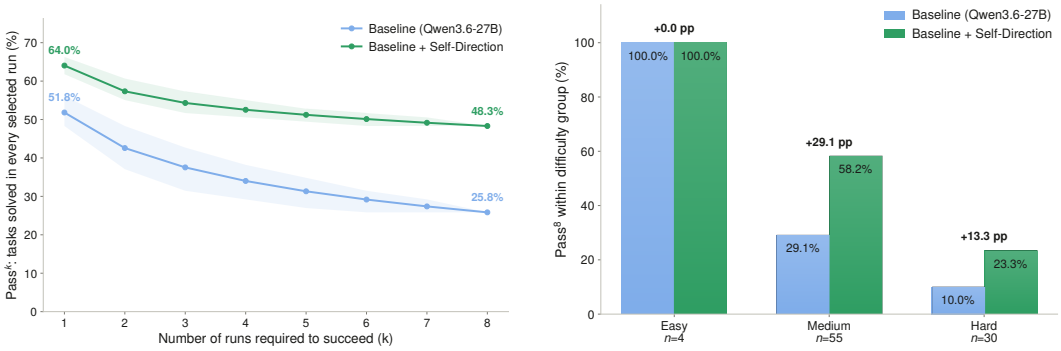


Figure 8: **Terminal-Bench 2.1 outcomes across eight complete runs.** Each bar decomposes success from the initial base execution and additional recovery by the full loop. Self-Direction improves all eight runs and raises mean accuracy from 51.8% to 64.0%.

9.0 and 18.0 percentage points. Mean success increases from 51.8% for the initial base execution to 64.0% for the full loop, an average gain of 12.2 points (see Figs. 8, 9).

The larger campaign also sharpens the distinction between coverage and reliability. The full loop increases pass@8 from 74.2% to 80.9%, but its larger effect is on pass^8: the number of tasks solved in every run increases from $23/89$ to $43/89$ and almost doubles from 25.8% to 48.3%. The loop moves 20 tasks from intermittent success to 8/8 success. In addition, all 23 initially 8/8-reliable tasks remain reliable and six never-solved tasks gain some coverage. Thus, Self-Direction does not merely produce occasional additional successes, it substantially enlarges the set of tasks solved consistently.

This eight-run analysis measures benchmark-level consistency when the complete learning process is executed independently in every run. Since Self-Direction may use upto three rollouts per task, the comparison with the base agent may not be compute-matched. The compute-matched analysis on recovery from mistakes is presented in Section 4.2. It evaluates whether the same stored repair remains successful across repeated encounters.



(a) Pass^k across repeated executions.

(b) Pass^8 stratified by task difficulty.

Figure 9: Self-Direction **improves repeated-run reliability on Terminal-Bench 2.1. Left:** pass^k, the fraction of tasks solved in each of k runs, remains consistently higher for the full loop and increases from 25.8% to 48.3% at $k = 8$. Lines report the mean over all subsets of k runs, with shaded bands showing the minimum and maximum. **Right:** The loop preserves 100% pass^8 on easy tasks while increasing reliability from 29.1% to 58.2% on medium tasks and from 10.0% to 23.3% on hard tasks.

In Table 4 we show additional results for Qwen3.5-9B on Terminal Bench.

Table 4: Terminal Bench 2.1 results for Qwen3.5-9B

Stage	Attempts	New wins	Stage success	Cumulative wins	Overall accuracy
Phase 1	89	23	25.8%	23	25.8%
Phase 4: adapter	65	7	10.8%	30	33.7%
Phase 4: base fallback	58	1	1.7%	31	34.8%
Final	**89**	**31**	–	**31**	**34.8%**

C.5 Gains Across Difficulty, Category, and Task Structure

Aggregate accuracy can conceal whether an improvement is concentrated in a narrow subset of tasks. We, therefore, stratify the results using the difficulty, category, and structural metadata provided by each benchmark. For every group, we report both mean accuracy across three runs and pass^3.

GAIA difficulty levels. Fig. 6 separates the GAIA comparison into Levels 1-3. Self-Direction improves performance at every level, showing that the pooled gain is not driven only by easier tasks.

Claw-Eval categories and task structure. Table 5 shows that the Claw-Eval improvement extends across heterogeneous workflows. On the two largest categories, Workflow and Ops, the full loop improves mean accuracy by 9.2 and 9.7 percentage points, respectively, while increasing pass^3 by 17.0 and 16.2 points. Multi-service tasks exhibit a similarly broad improvement, with mean accuracy increasing from 80.6% to 91.1% and pass^3 increasing from 66.7% to 85.0%.

The gains are not limited to one execution structure. Self-Direction improves both state-changing and non-state-changing tasks, as well as safety-primary and non-safety-primary tasks. The task-structure rows represent three separate binary attributes and are, therefore, not mutually exclusive across attributes.

Terminal-Bench difficulty and task categories. Table 6 shows that the improvement persists as the difficulty of Terminal Bench 2.1 tasks increases. Easy tasks are already solved in every run by the base agent. On medium tasks, however, the full loop raises mean accuracy from 62.4% to 78.2% and pass^3 from 43.6% to 70.9%. Hard-task accuracy increases from 25.6% to 37.8%, whereas pass^3 increases from 10.0% to 26.7%.

Table 5: **Claw-Eval results stratified by task category and structure.** Mean accuracy is computed across three independent runs, while pass^3 requires success in all three runs. Task categories partition the 161 tasks. Task-structure rows describe three separate binary attributes: multi-service, state-changing, and safety-primary.

Stratification	Group	N	Mean Accuracy		Pass^3	
			Initial base	Full loop	Initial base	Full loop
Overall	All tasks	161	64.4%	73.3%	51.6%	64.6%
Category	Workflow	47	78.7%	87.9%	61.7%	78.7%
	Ops	31	89.2%	98.9%	80.6%	96.8%
	Finance	14	28.6%	40.5%	14.3%	28.6%
	Office QA	10	33.3%	43.3%	20.0%	20.0%
	Communication	8	75.0%	75.0%	75.0%	75.0%
	Productivity	7	61.9%	66.7%	42.9%	57.1%
	Operations	6	83.3%	88.9%	83.3%	83.3%
	Safety	5	53.3%	60.0%	40.0%	60.0%
	Terminal	5	13.3%	33.3%	0.0%	0.0%
	Other categories	28	46.4%	54.8%	32.1%	46.4%
Task structure	Multi-service	60	80.6%	91.1%	66.7%	85.0%
	Other general tasks	101	54.8%	62.7%	42.6%	52.5%
	State-changing action required	39	71.8%	78.6%	59.0%	74.4%
	No expected state change	122	62.0%	71.6%	49.2%	61.5%
	Safety-primary	65	71.3%	75.4%	58.5%	70.8%
	Not safety-primary	96	59.7%	71.9%	46.9%	60.4%

Furthermore, the category breakdown in Table 6 shows improvements across substantially different technical capabilities, including software engineering, system administration, data science, security, scientific computing, and file operations. Improvements in pass^3 are particularly pronounced for system administration, data science, security, and machine-learning tasks. The aggregate result, therefore, does not arise from repeatedly repairing one narrow class of terminal problems.

Together, these stratifications show that failure-conditioned Self-Direction remains useful across task types, execution structures, and difficulty levels. Results for very small groups should be interpreted descriptively, particularly categories containing fewer than five tasks.

C.6 Controlling for the number of agent rollouts

Agent rollouts per task measure computational cost, counting task-environment executions only and excluding reflection, synthesis, and adapter training. We compare the loop's accuracy and rollout count against the base pass@k curve.

Self-Direction improves accuracy at comparable task-environment rollout budgets. As shown in Fig. 10, the loop reaches 55.8% accuracy on GAIA, with only 2.10 agent rollouts per task, matching the accuracy achieved by base pass@6. On Terminal-Bench 2.1, it reaches 66.3% accuracy with 1.90 rollouts per task, exceeding the 65.3% obtained by pass@3 using a similar number of rollouts. On Claw-Eval, the loop exceeds the accuracy of pass@3 while requiring a similar 1.54 rollouts per task. Across all three domains, the loop, therefore, obtains higher accuracy than simple retrying, while averaging a similar number of rollouts, making Self-Direction rollouts efficient compared to standard resampling. While this reinforces the strength of spending additional test-time compute, simply resampling parallel rollouts does not retain any lessons; the agent is just as likely to repeat the mistake if the same task is encountered again.

Table 6: **Terminal-Bench 2.1 results stratified by difficulty and task category.** Mean accuracy is computed across three independent runs, while pass^3 requires success in all three runs. Self-Direction improves both medium and hard tasks and produces gains across a broad range of technical categories.

Stratification	Group	N	Mean Accuracy		Pass^3	
			Initial base	Full loop	Initial base	Full loop
Overall	All tasks	89	51.7%	65.5%	34.8%	57.3%
Difficulty	Easy	4	100.0%	100.0%	100.0%	100.0%
	Medium	55	62.4%	78.2%	43.6%	70.9%
	Hard	30	25.6%	37.8%	10.0%	26.7%
Category	Software Engineering	26	38.5%	47.4%	26.9%	42.3%
	System Administration	9	63.0%	88.9%	33.3%	77.8%
	Data Science	8	50.0%	70.8%	25.0%	62.5%
	Scientific Computing	8	29.2%	37.5%	12.5%	37.5%
	Security	8	75.0%	91.7%	62.5%	87.5%
	Debugging	5	86.7%	93.3%	60.0%	80.0%
	File Operations	5	33.3%	60.0%	20.0%	40.0%
	Data Processing	4	83.3%	83.3%	75.0%	75.0%
	Mathematics	4	25.0%	41.7%	0.0%	25.0%
	Model Training	4	66.7%	83.3%	50.0%	75.0%
	Machine Learning	3	66.7%	88.9%	33.3%	66.7%
	Data Querying	1	100.0%	100.0%	100.0%	100.0%
	Games	1	0.0%	33.3%	0.0%	0.0%
	Optimization	1	100.0%	100.0%	100.0%	100.0%
	Personal Assistant	1	100.0%	100.0%	100.0%	100.0%
	Video Processing	1	0.0%	0.0%	0.0%	0.0%

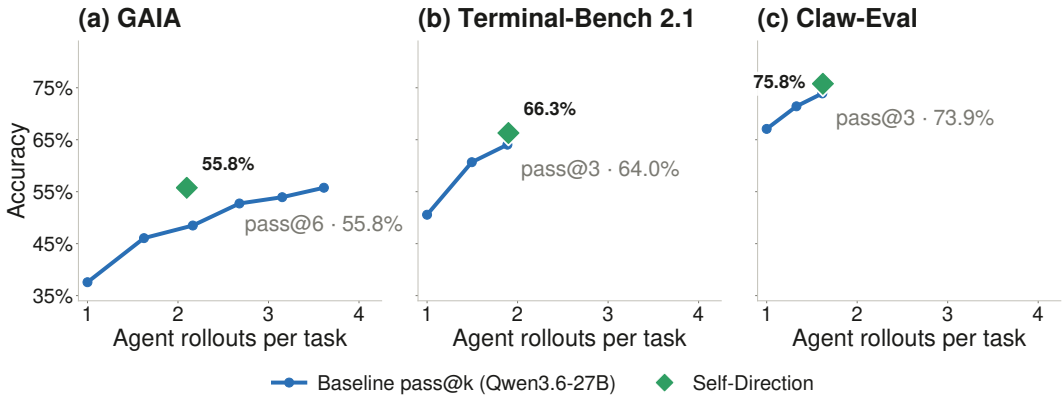


Figure 10: Self-Direction **reaches higher accuracy with comparable task-environment rollouts.** Relative to base pass@k, Self-Direction outperforms the pass@k baseline across all three benchmarks while using a comparable number of full agent rollouts. Reflection, synthesis, and adapter optimization details are presented in Appendix F.3.

C.7 GAIA Deployment Stream Adapters Trained

Fig. 11 reports the number of adapters trained for each of the P25-, P50-, and P75-th threshold values in the GAIA deployment stream experiments, as described in Section 4.4. Each new adapter occupies

a marginal 85 MB of storage. Qualitative examples of recoveries from these adapters are further presented in Appendix H.1.

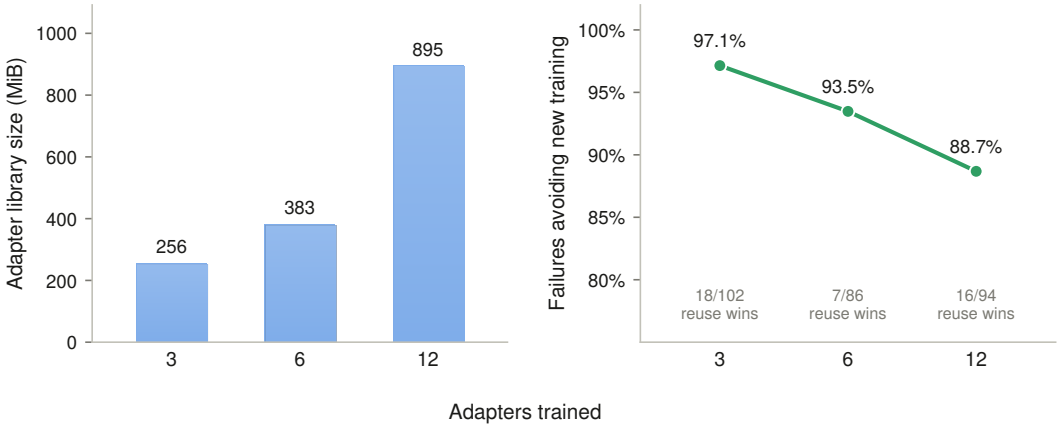


Figure 11: **A compact skill library amortizes learning across a deployment-like GAIA stream.** Existing skills avoid new adapter training for 88.7-97.1% of initial failures. Bars show adapters trained; annotations report storage and successful reuses; the green line shows training avoided through reuse.

C.8 Preserving existing skills

We show in Table 7 the threshold of similarity at which Self-Direction does not select an unrelated adapter, leaving existing capabilities perfectly intact in cross-benchmark settings. We find that any global threshold above 0.55 can perfectly preserve existing capabilities after Self-Direction improvements, with a full library of adapters across GAIA, Claw-Eval, and Terminal-Bench 2.1.

Table 7: Self-Direction **learns new skills without disturbing existing ones.** The safe threshold is the smallest two-decimal cutoff above the largest similarity observed across the pair. At this threshold, no task ever retrieves an adapter from the other domain, in either direction; hence, a capability the agent already has is never overwritten or misapplied (✓).

Benchmark Pair	Max. Similarity	Safe τ	Cross-Domain Retrievals	Capabilities Preserved
GAIA–TB 2.1	0.546	0.55	0 / 145	✓
GAIA–Claw	0.415	0.42	0 / 141	✓
TB 2.1–Claw	0.306	0.31	0 / 84	✓
All three, one global τ	0.546	0.55	0 / 370	✓

Existing capabilities are preserved for reasonable threshold values. In addition to evaluating perfect transfer through a global threshold, we set up several experiments where the threshold τ decreases below the optimal threshold allowing for cross-benchmark selection and loading of adapters. We vary τ from 0.2 to 0.6 in increments of 0.01 and present the results in Fig. 12. We find that except for low thresholds (dipping below 0.30), task success on GAIA and TerminalBench 2.1 is preserved even when the skill library contains a full library of skills from the two other unrelated benchmarks. When τ is very low ($\tau < 0.35$), we find that adapters from other benchmarks can help or hurt, depending on benchmark transfer.

C.9 Statistical comparison on GAIA

We compare Self-Direction with its initial base-agent execution using a paired, task-level t-test. The unit of analysis is an individual GAIA task, rather than an evaluation run or difficulty level.

For each of the $n = 165$ tasks, let

$$B_i \in \{0, 1\}$$

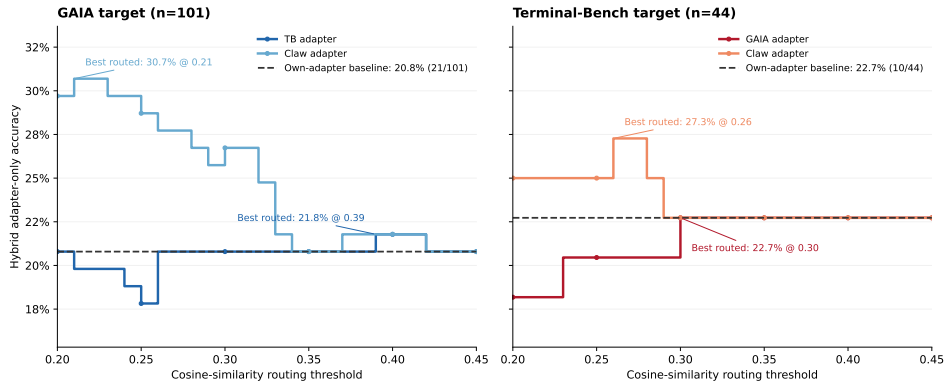


Figure 12: **Accuracy across router thresholds τ with a fixed library of cross-benchmark skills. Left:** GAIA tasks failed by the base agent are retried with the corresponding top-matched adapter from Terminal Bench and Claw-Eval libraries. **Right:** Terminal Bench 2.1 tasks failed by the base agent are retried with the corresponding top-matched adapter from GAIA and Claw-Eval libraries. We see existing capabilities are preserved except for low thresholds dipping below 0.30, where adapters from other benchmarks in some cases can help or hurt accuracy on the new benchmark.

denote whether the initial base-agent attempt is correct and let

$$S_i \in \{0, 1\}$$

denote whether the final output after executing the Self-Direction loop is correct. We define the paired difference

$$D_i = S_i - B_i.$$

The observed outcomes are:

Outcome	Count	Rate
Base correct	64	38.8%
Self-Direction correct	92	55.8%
Recovered by Self-Direction	28	17.0%

Thus, 64 tasks are solved by both stages, 28 are solved only after Self-Direction, and 73 are unsolved by both. The mean paired difference is

$$\bar{D} = \frac{28}{165} = 0.1697.$$

We compute the test statistic as

$$t = \frac{\bar{D}}{s_D/\sqrt{n}},$$

where s_D is the sample standard deviation of the 165 paired differences. The two-sided test gives

$$t(164) = 5.79, \qquad p = 3.50 \times 10^{-8}.$$

The corresponding 95% confidence interval for the accuracy improvement is $[11.2,\ 22.8]$ percentage points.

The results are pooled at the task level across GAIA Levels 1-3, comprising 53 Level-1, 86 Level-2, and 26 Level-3 tasks.

D Self-Direction WITH A STRONGER BASE MODEL

The benefits of Self-Direction should not depend on beginning with a weak base agent. We test this by applying the same loop to Qwen3.8-27B and comparing it with Qwen3.6-27B on identical GAIA, Claw-Eval, and Terminal-Bench 2.1 task sets.

As expected, Qwen3.8-27B solves more tasks during its initial execution on all three benchmarks. Self-Direction nevertheless provides substantial additional gains. On GAIA, it recovers 27 failures and raises accuracy from 59.4% to 75.8% ($+16.4$ pp). On Claw-Eval, it raises accuracy from 64.6% to 76.4% ($+11.8$ pp), whereas on Terminal-Bench 2.1, it improves accuracy from 55.1% to 73.0% ($+17.9$ pp). The corresponding Qwen3.6-27B runs also improve on every benchmark, with gains between 15.5 and 18.0 percentage points.

These results support the central role of failure as an intervention point. As the base model becomes more capable, fewer tasks enter the mistake subset but the remaining failures still provide useful signals for targeted in-weight repair. Self-Direction, therefore, complements improvements in the underlying model rather than merely compensating for a weaker checkpoint. This experiment establishes robustness across two capability levels within the same model family; broader generalization across model families and scales remains an important direction for future work.

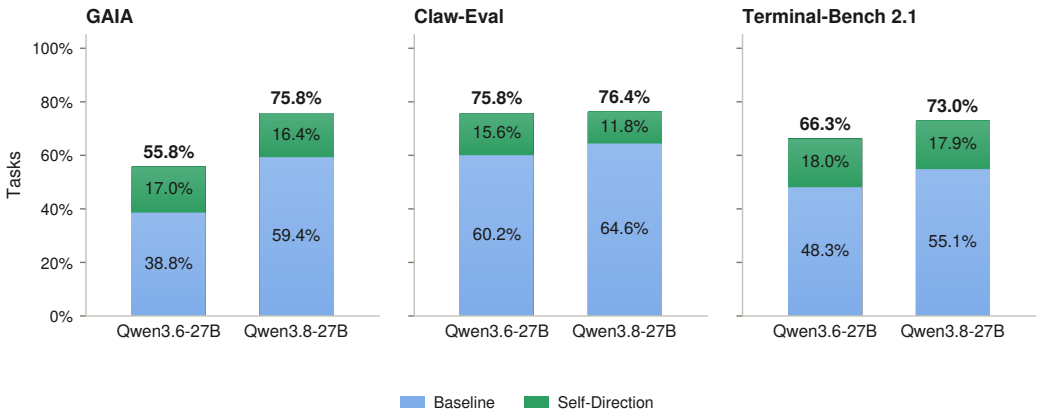


Figure 13: Self-Direction **continues to improve a stronger base model.** We apply the same failure-conditioned loop to Qwen3.6-27B and Qwen3.8-27B on identical benchmark tasks. Blue denotes tasks solved by initial base execution, green denotes additional tasks recovered by the complete loop. Self-Direction produces substantial gains for both models across GAIA, Claw-Eval, and Terminal-Bench 2.1.

E WHY Self-Direction USES MODULAR IN-WEIGHT SKILLS

Self-Direction relies on two design choices. Each failure produces a distinct adapter skill and relevant experience is retrieved on a later task. We examine both choices on GAIA. We first compare failure-specific adapters with pooled, mismatched, and randomly initialized LoRA alternatives. We then quantify why retaining the complete learning history directly in context does not scale.

E.1 SKILL SPECIFICITY AND RECOVERY

Adding or modifying existing parameters is useful only when it captures a lesson relevant to the failure being repaired. Fig. 14 compares the complete Self-Direction policy with several alternative representations of the same failure-derived experience.

The full loop recovers 27.7% of the initial failures. The recovery rate of a randomly Gaussian-initialized Dynamic LoRA (instead of SVD-initialized) is 22.2%. A textual `Skills.md` memory has a recovery rate of 18.1%, showing that explicit textual lessons can provide useful immediate guidance. However, a single pooled LoRA recovers only 16.7% and 14.6% of attempted failures at ranks 16 and 64, respectively. Deliberately applying a mismatched adapter by choosing the least similar one reduces recovery further to 9.6%, which is worse than pass@2 base retry on the failure.

These results offer evidence that the benefit does not arise from inserting an arbitrary parameter update or increasing adapter capacity. Recovery is strongest when the learned skill remains aligned with the failure that produced it.

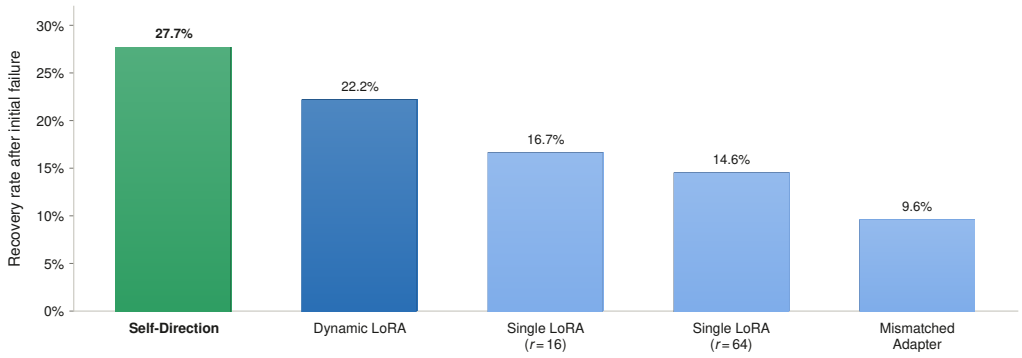


Figure 14: **Recovery after an initial failure on GAIA.** The complete Self-Direction loop recovers 27.7% of the failures. Text memory stores lessons in `Skills.md`; the single-LoRA baselines pool experience across failures and the mismatched condition applies an unrelated adapter.

E.2 WHY NOT PLACE EVERY LESSON IN CONTEXT?

Textual memory is useful but accumulating every lesson in the active context is not a scalable form of persistence. The 101 failures in the GAIA Self-Direction run produce 2,428 corrective examples: 672 from Level 1, 1,323 from Level 2, and 433 from Level 3. Using the Qwen3.6 tokenizer, their instruction-response content is estimated to contain approximately 362k tokens, implying that each self-reflection loop produces an additional $\sim 3.6k$ tokens per failure. Including chat formatting and stored rationales increases this total to approximately 410k tokens.

This already exceeds the model's native 262,144-token context window. Moreover, the run reserves up to 16,384 tokens for generation, leaving at most 245,760 tokens for the input. Even the pooled synthesized responses across all failures require approximately 276k tokens, before including their instructions, the current task, system instructions, tool definitions, or the agent's ongoing trajectory.

The failed executions are larger still. Their full trajectories contain approximately 2.72M tokens. The compact representations supplied to lesson synthesis contain 188k tokens after truncation but combining even these compact failures with the generated lessons requires approximately 599k tokens. Retaining the complete failure trajectories and lessons would require roughly 3.13M tokens, or nearly twelve native context windows. The summary of this analysis is presented in Table 8.

Table 8: **Context required to retain the complete GAIA learning history.** Counts use the Qwen3.6 tokenizer. Totals are rounded because this section is intended as a capacity analysis.

Stored content	Approximate tokens	Native context windows
Full trajectories for 101 failures	2.72M	10.4×
Truncated failure representations	188K	0.72×
2,428 lesson instruction-response pairs	362K	1.38×
Lessons with formatting and rationales	410K	1.56×
Full trajectories and lessons	3.13M	11.9×

The implication is not that textual memory is ineffective. The `Skills.md` baseline demonstrates that a small, relevant textual lesson can improve recovery. Rather, a continually growing agent cannot reinsert its entire learning history on every task. It must select a small amount of relevant experience. Self-Direction performs this selection over modular adapter skills, enabling acquired lessons to persist without occupying the active prompt context.

F AGENT EXECUTION AND TRAINING HYPERPARAMETERS

We use the same agent-generation and adapter-training configuration across benchmarks unless stated otherwise. This section separates the parameters governing task execution from those used to construct each modular adapter skill.

F.1 AGENT GENERATION AND REACT EXECUTION

The agent operates through a ReAct-style loop (Yao et al., 2022) that interleaves model decisions with environment interaction. At each step, the model receives the task and accumulated trajectory, then emits either a tool call or a final response. A tool call is executed by the environment, its observation is appended to the trajectory, and the model is invoked again. A rollout terminates when the model provides a final answer or exhausts its interaction budget.

The same decoding configuration is used for the initial base execution, adapter-guided retry, and base fallback (see Tables 9, 10). The larger Terminal-Bench budget accommodates longer coding and environment-manipulation tasks. These values are upper bounds: A rollout terminates early whenever the agent completes the task or emits a final answer without any tool calls.

Table 9: **Agent generation and interaction hyperparameters.** The token limit applies independently to each model turn within the ReAct loop, rather than to the complete task trajectory.

Hyperparameter	GAIA / Claw-Eval	Terminal-Bench 2.1
Temperature	0.6	0.6
Top-p	0.95	0.95
Maximum new tokens per turn	16,384	16,384
Maximum ReAct steps	50	100
Maximum tool calls	100	100
Stop generation after tool call	Yes	Yes

F.2 ADAPTER SYNTHESIS AND TRAINING

Each failed execution produces up to 24 corrective examples. Adapter training begins only when at least eight valid examples are available. All base-model parameters remain fixed. Each skill is represented by a rank-16 LoRA adapter initialized from the targeted base-model weights using the SVD procedure described in Section 3. The first LoRA factor is initialized from the leading right singular vectors and frozen, whereas the second factor begins at zero and is optimized on the corrective examples.

The target modules depend on the normalized failure reason rather than the benchmark. Failures involving planning or interaction limits update the attention projections, `q_proj` and `v_proj`; incorrect answers and execution exceptions update the multi-layer perceptron (MLP), `down_proj`. This keeps the in-weight repair localized to the type of behavior implicated by the failure.

Reflection uses the same base model with temperature 0.6. Its maximum generation length is 16,384 tokens for all benchmarks. No external teacher model is used.

Failure normalization. We normalize each failure using the agent's termination signal, the presence of an extractable response, and the binary verifier outcome. Budget exhaustion and execution exceptions take precedence over output-based labels. Otherwise, an empty response is assigned `no_final_response`, whereas a completed response rejected by the verifier is assigned `wrong_answer`. Table 11 summarizes the resulting vocabulary.

The normalized reason, therefore, affects adaptation in two ways: It focuses reflection on the procedural deficiency exposed by the failure and determines where the resulting lesson is represented in the adapter. This routing is a lightweight localization prior rather than an assumption that planning and answer correctness reside in disjoint model components.

Information boundary. Normalized failure reasons are deliberately categorical and low-bandwidth. Let t denote the agent's termination status, e indicate whether an extractable response was produced,

Table 10: **Adapter synthesis and training hyperparameters.** These settings are shared across GAIA, Claw-Eval, and Terminal-Bench 2.1. Unlisted `TrainingArguments` retain their library defaults.

Component	Hyperparameter	Value
Corrective data	Maximum examples per failure	24
Corrective data	Minimum examples required	8
Corrective data	Maximum assistant-response length	16,384 tokens
Corrective data	Trajectory steps retained for reflection	24
LoRA	Rank r	16
LoRA	Scaling parameter α	16
LoRA	Dropout	0.05
LoRA	Bias parameters	None
LoRA	Target modules	Failure-dependent
SVD initialization	Initialized factor	A from leading right singular vectors
SVD initialization	Trainable factor	B only; initialized to zero
SVD initialization	Power iterations	2
SVD initialization	Oversampling dimension	8
SVD initialization	SVD device	CPU, cached across skills
Optimization	Learning rate	5×10^{-4}
Optimization	Training epochs	2
Optimization	Per-device batch size	1
Optimization	Gradient accumulation steps	8
Optimization	Effective batch size	8
Optimization Optimization	Gradient checkpointing	Enabled
Optimization	Numerical precision	BF16 when supported

Table 11: **Normalized failure reasons used by** Self-Direction**.** Each categorical reason determines which projections are updated during adaptation.

Reason	Operational definition	Target modules
`step_budget`	The agent reaches its maximum number of interaction steps before verified completion.	`q_proj`, `v_proj`
`tool_call_budget`	The agent exhausts its permitted tool calls before completing the task.	`q_proj`, `v_proj`
`no_final_response`	The execution ends without an extractable final response and was not terminated by a budget or exception.	`q_proj`, `v_proj`
`wrong_answer`	The agent produces a response but the verifier rejects the resulting answer or environment state.	`down_proj`
`exception`	An execution error prevents the agent from completing the task.	`down_proj`

and $z = \mathcal{V}(x, \tau) \in \{0, 1\}$ denote the verifier's binary decision. The reason exposed to reflection and routing is

$$r = \text{Normalize}(t, e, z).$$

The reference answer, hidden-test contents, and verifier-internal artifacts are not arguments to this function. In particular, r contains neither gold-answer tokens nor test outputs, judge scores, rubrics, rationales, or free-form judge feedback. The loop observes only the categorical outcome of verification, not the information used internally to compute it.

We additionally enumerated every persisted reason value in the reported GAIA, Claw-Eval, and Terminal-Bench 2.1 runs. The observed values of r were exclusively `step_budget`, `wrong_answer`, and `exception`; no stored reason contained task-specific text, answers, test results, scores, or judge feedback. Thus, the failure classification and module-routing decisions cannot communicate how to solve the task. They convey only a coarse description of how the preceding execution failed.

F.3 Adapter Training and Compute Requirements

Self-Direction performs a small amount of training only after a mistake is detected, triggering self-reflection and adapter training. Each reflection call produces ~ 24 structured instruction-response examples. Measured with the Qwen3.6-27B tokenizer and the actual chat template used for training, these examples contain an average of 3,622 tokens per adapter. Rationales generated during reflection are retained for analysis but are not included in the training loss.

The fixed base model is adapted using rank-16 LoRA with SVD initialization. The LoRA A matrices are initialized from cached right-singular directions of the corresponding base-model weights and remain frozen; only the B matrices are optimized. Training uses two epochs, a per-device batch size of one, gradient accumulation over eight examples, BF16 precision, and gradient checkpointing on one NVIDIA H200. Thus, an average adapter processes approximately 7,244 token presentations across both epochs.

Measurement protocol. We report synthesis costs calculated from analyzing 46 GAIA failures, 39 Claw-Eval failures, and 45 Terminal-Bench 2.1 failures (130 in total). All stages execute sequentially on one NVIDIA H200. Synthesis time covers failure reflection and example generation, optimization time is the recorded trainer runtime and end-to-end time extends from the beginning of reflection until the adapter is saved. The initial task execution and subsequent task-environment rollouts are not included and reported separately in Fig. 10.

Table 12: **Cost of constructing a modular adapter skill.** Values are averages over 130 failures across the three benchmarks. End-to-end latency includes reflection, data preparation, optimization, and adapter serialization on one NVIDIA H200. Energy denotes measured GPU energy over the same interval.

Benchmark	Adapters	Training tokens per adapter	Synthesis (min)	Optimization (s)	End-to-end (min)	GPU energy (Wh)
GAIA	46	4,059	8.51	80.9	9.89	37.0
Claw-Eval	39	3,709	5.82	80.3	7.19	26.5
Terminal-Bench 2.1	45	3,100	5.79	81.4	7.22	27.1
Overall	**130**	**3,622**	**6.76**	**80.9**	**8.16**	**30.4**

Latency and energy. Reflection and data synthesis take 6.76 minutes, on an average, on our setup, followed by 80.9 seconds (~ 1.5 minutes) of adapter optimization. The complete post-failure construction process, therefore, takes 8.16 minutes, on an average, and 6.95 minutes at the median. Average measured GPU energy is 30.4 Wh per adapter, comprising 26.6 Wh for synthesis and 3.68 Wh for optimization. Across the 130 measured adapters, this corresponds to approximately 17.7 H200 GPU-hours and 3.96 kWh of GPU energy. These measurements cover the GPU only and exclude host, storage, and datacenter overhead.

Trainable parameters and storage. The number of trainable parameters depends on the normalized failure reason and its corresponding target modules. Attention adapters that target `q_proj` and `v_proj` optimize 3.41M parameters, or 0.013% of the 26.90B-parameter base model. MLP adapters that target `down_proj` optimize 5.24M parameters, or 0.0195%. Thus, each repair trains at most approximately 0.02% additional parameters while leaving the base model fixed.

Because the frozen SVD-initialized A matrices must also be retained, the complete adapter tensors contain 6.03M parameters for attention repairs and 23.07M parameters for MLP repairs. Their serialized weight files occupy 23.0 MiB and 88.0 MiB, respectively. The recorded artifact directories occupy 42.1-107.1 MiB because each directory also contains an identical 19.1 MiB tokenizer file; this shared file can be stored once when maintaining an adapter library.

Approximate optimization FLOPs. Let $N = 26.90$B denote the number of base-model parameters and $D = 2 \times 3{,}622 = 7{,}244$ denote the average number of token presentations across two epochs. A forward pass through the base model costs approximately $2ND$. Backpropagation must propagate activation gradients through the fixed base model but does not compute updates for its weights, giving

Table 13: **Tool interfaces used by each benchmark.** The Claw-Eval row reports the union of tools across the evaluated tasks; each individual task exposes only its task-specific subset. Tool names are shown as presented to the agent.

Benchmark	Available tools
GAIA	`web_search`, `web_fetch`, `python_exec`, `shell`, `file_read`, `file_open`, and `list_dir`. These tools support web research, numerical computation, shell commands, directory inspection, and extraction from text, PDF, spreadsheet, HTML, and image files.
Claw-Eval	**Email and contacts:** `gmail_list_messages`, `gmail_get_message`, `gmail_save_draft`, `gmail_send_message`, `contacts_search`, `contacts_get`, and `contacts_send_message`. **Calendar, tasks, and scheduling:** `calendar_list_events`, `calendar_get_event`, `calendar_create_event`, `calendar_get_user_events`, `calendar_delete_event`, `todo_list_tasks`, `todo_get_task`, `todo_update_task`, `todo_create_task`, `todo_delete_task`, `scheduler_list_jobs`, `scheduler_get_job`, `scheduler_job_history`, `scheduler_create_job`, `scheduler_update_job`, and `scheduler_delete_job`. **Business services:** finance, CRM, inventory, reporting, helpdesk, and integration-management tools for listing, inspecting, updating, and creating benchmark entities. **Knowledge and content:** `kb_search`, `kb_get_article`, `kb_update_article`, `notes_list`, `notes_get`, `notes_share`, `rss_list_feeds`, `rss_list_articles`, `rss_get_feed`, `rss_get_article`, `rss_publish`, `web_search`, and `web_fetch`. **Documents and media:** `documents_extract_text`, `ocr_extract_text`, and `caption_describe_image`. The nine sandbox tasks additionally expose `Bash`, `Read`, `Write`, `Edit`, `Glob`, and `Grep` inside an isolated workspace.
Terminal-Bench 2.1	`terminal_shell`, which executes Bash commands inside a fresh, task-specific Podman container. Commands share a persistent filesystem for the duration of an attempt and return their exit code, standard output, and standard error. The shell supports inspecting files and processes, editing or creating artifacts, installing dependencies, compiling software, configuring services, and running agent-authored validation checks.

an approximate training cost of

$$C_{\text{adapter}} \approx 4ND \approx 7.8 \times 10^{14} \text{ FLOPs} = 0.78 \text{ PFLOPs}$$

per adapter, before the comparatively small LoRA and optimizer overhead. For reference, applying the conventional full-parameter training approximation of $6ND$ yields a conservative upper bound of approximately 1.17 PFLOPs. These are analytical estimates rather than hardware-counter measurements; the latency and energy values in Table 12 are measured directly.

F.4 Benchmark Tool Interfaces

Each benchmark is evaluated using its native interaction interface. The initial base execution, adapter-guided retry, and base fallback receive the same tools. Hence, improvements cannot arise from exposing additional capabilities after failure. During reflection, Self-Direction also receives the tools available in the failed execution, enabling it to generate corrective examples that refer to valid actions.

GAIA uses a fixed collection of general-purpose research and file-processing tools. Claw-Eval exposes task-specific subsets of a larger service-oriented tool collection and its nine sandbox tasks provide shell and file-manipulation tools in addition. Terminal-Bench 2.1 exposes a single persistent shell inside a fresh task container (see Table 13).

For Terminal-Bench, each adapter or base retry begins in a fresh container to prevent filesystem changes from a previous attempt from leaking into the next one. Claw-Eval similarly relies on the official benchmark services and audit logs. GAIA retries retain access to the same task attachments and general-purpose tools. Thus, Self-Direction changes how the agent uses an existing interface, rather than changing the interface itself.

F.5 Self-Direction Loop

The Self-Direction loop is based on three steps: 1) reflect to produce a lesson in text, 2) distill lesson in LoRA, and 3) load LoRA weights in model. The Python pseudo-code that implements the high-level steps is as follows:

```
skill_library = []
SIM_THRESHOLD = tau
LIBRARY_CAP = C

def agent_loop(task, base_model):
    success, trace = agent(base_model, task)
    if success:
        return True

    return self_direction(base_model, task, trace)


def self_direction(base_model, task, failure_trace):
    # Reuse a sufficiently related prior skill.
    skill, similarity = retrieve_best(
        task, skill_library
    )

    if skill is None or similarity < SIM_THRESHOLD:
        skill = None

        # Learn only while the bounded library has capacity.
        if len(skill_library) < LIBRARY_CAP:
            reason = normalize_failure(failure_trace)
            lessons = reflect(
                base_model,
                task,
                failure_trace,
                reason,
                REFLECTION_PROMPT,
            )

            adapter = train_lora(
                freeze(base_model), lessons
            )
            skill = register(
                adapter,
                retrieval_key=embed(task),
                source_task=task,
            )
            skill_library.append(skill)

    # Retry using either a retrieved or newly learned skill.
    success = False
    if skill is not None:
        adapted_model = load_adapter(
            base_model, skill.adapter
        )
        success, _ = agent(
            adapted_model, task
        )
        unload_adapter(adapted_model)

    # Fall back to the clean, fixed base model.
    if not success:
        success, _ = agent(
            base_model, task, RETRY_PROMPT
        )

    return success
```

G PROMPT TEMPLATES

Self-Direction uses the same base model for task execution and failure reflection. Prompts are specialized to each benchmark's tools and success conditions but share the common structure shown next. We present generic versions for readability first; benchmark-specific prompts are presented in the following Section G.3.

G.1 TASK-EXECUTION SYSTEM PROMPT

The execution prompt defines a ReAct-style interaction protocol. At each turn, the agent either emits one tool call and waits for its result or terminates with a final answer. Tool observations are appended to the trajectory before the next model invocation.

Task-execution system prompt

```
You are a careful agent working in [ENVIRONMENT].
Reason step by step and use the provided tools when they help.

Tool-use protocol:
- Emit one tool call per turn using the required tool-call format,
  then stop generating.
- Wait for the tool result before deciding the next action.
- Inspect tool outputs and errors before proceeding.
- Avoid repeating actions that do not advance the task.
- Respect the available step and tool-call budgets.

Completion protocol:
- Verify that the requested answer or environment state has been
  produced before stopping.
- When the task is complete, end with:
  Final Answer: [ANSWER OR DONE]
```

GAIA specializes this template for short-answer problem solving. Terminal-Bench exposes a container shell and emphasizes verifying filesystem or program state. Claw-Eval supplies task-specific tools, whereas MiniWoB++ describes accessibility-tree observations and browser actions.

G.2 FAILURE-REFLECTION PROMPT

After a verified failure, the base model acts as its own teacher. It receives the task, normalized failure reason, available tools, and a compact rendering of the failed trajectory. There is NO leakage of the correct answer or any hidden tests.

The self-reflection stage produces structured instruction–response examples rather than an unconstrained textual critique. These examples teach the procedure that was missing from the failed execution and become the training data for the modular adapter skill. Note that they are NOT full model rollouts.

Failure-reflection prompt

```
You are a teacher model for continual agent learning.

A student agent attempted a task using tools and failed. Generate
training examples that teach the PROCEDURE needed to solve this class
of task. Do not teach the student to memorize the tools or answers.

NO ANSWER LEAKAGE:
- Do not copy a task-specific answer, hidden solution, verifier rule,
  or final artifact into a response.
- Teach how to derive or verify the answer instead.
```

```
Each example should teach:
- which tools to use and in what order;
- what queries, commands, or inputs to provide;
- how to interpret observations and tool results;
- how to recover from ineffective actions;
- when enough evidence has been collected to stop.

Use the failed trajectory as evidence of the missing skill. Vary the
examples across related situations so that the lesson describes a
general procedure rather than one trajectory.

Return only a JSON array with exactly [N] objects:
[
  {
    "instruction": "...",
    "response": "...",
    "rationale": "..."
  }
]

TASK (x):
[TASK]

FAILURE REASON (r):
[FAILURE REASON]

FAILURE-SPECIFIC FOCUS:
[GUIDANCE FOR WRONG ANSWER, STEP BUDGET, TOOL BUDGET, OR MISSING
  FINAL RESPONSE]

AVAILABLE TOOLS:
[TOOL NAMES OR BENCHMARK TOOL DESCRIPTIONS]

FAILED TRAJECTORY (t):
[COMPACT SEQUENCE OF ACTIONS, OBSERVATIONS, ERRORS, AND FINAL OUTPUT]
```

G.3 Benchmark-Specific Prompt Specializations

The generic templates above are instantiated by fixed benchmark-specific system prompts. These prompts are shared across tasks and runs within each benchmark; only the benchmark task, current observation, attachment, and task-local tool schema vary. To make these differences explicit, we reproduce the principal benchmark-specific clauses below. The bracketed benchmark names are editorial labels; the prompt text itself is reproduced verbatim.

Benchmark-specific execution clauses

```
[GAIA]

You are a careful problem solver working on GAIA benchmark tasks.
Reason step by step and use the provided tools when they help.

Final answer protocol:
- When you are ready, respond with concise reasoning followed by a
  single line beginning with 'Final Answer:' on the last line.
- The answer should be in the simplest form the question asks for
  (a number, a short string, or a comma-separated list).


[Terminal-Bench 2.1]

You are completing a Terminal-Bench task inside an isolated Linux
```

```
container.

Use 'terminal_shell' to inspect the environment, create or edit
files, install or configure software when necessary, and verify
your work. Commands start in the task image's working directory and
 filesystem changes persist throughout this attempt.

- Check command exit codes and inspect relevant outputs or
artifacts.
- Do not look for benchmark tests, verifier files, reward files, or
  solutions. They are intentionally unavailable during the attempt.

When the requested state has been produced and checked, end with
"Final Answer: done".


[Claw-Eval]

You are running Claw-Eval tasks through the official Claw-Eval
harness.
Use the provided benchmark tools directly and complete the user's
request. The benchmark grades the full trace, service audit logs,
generated artifacts, and final response, not just a short final
answer.

- Tool names are case-sensitive. Use exactly the names shown in the
  tool list.
- Do not claim that you performed an action unless you actually
used
  the relevant tool or verified the artifact/state.

Stopping:
- When the task is complete, provide the requested result clearly.
If
  the task asks for a report, classification, draft, or plan, put
that
  content in your final assistant message.


[MiniWoB++]

You are controlling a browser for a MiniWoB++ task. Reason step by
step.

Each observation contains:
  goal: <what to accomplish>
  url:, reward:, terminated:, truncated:[, last\_action\_error:]
  page: an accessibility tree where every interactive element has a
 bid.

- Use bids that appear in the current 'page:' tree; never invent
bids.
  If 'last_action_error:' is set, do not repeat the same call---
choose
  a different bid or re-observe. Avoid repeated scroll/wait loops.

Stopping:
- When an observation shows 'terminated: True' with 'reward >=
0.5',
  respond with a single line: 'Final Answer: done'. Do not invent
any
  other final answer.
```

The reflection prompt is specialized in the same way. GAIA uses the generic failure-reflection system prompt shown above without an additional benchmark-specific override. The other benchmarks replace it with prompts that encode their observable state and explicitly exclude information unavailable to the agent as shown below.

Benchmark-specific reflection constraints

```
[Terminal-Bench 2.1]

Use only the task instruction and the agent's terminal trajectory
as evidence. The hidden verifier, benchmark tests, oracle solution,
 and reward implementation are unavailable and must not be guessed
or reconstructed.

Vary examples across related tasks. Do not copy task-specific
answers, secrets, exact final artifacts, hidden checks, or an
oracle command sequence.


[Claw-Eval]

Use the failed trajectory as evidence of which agent skill was
missing.

CRITICAL RULES:
- Do not copy a task's reference solution or grader rubric into the
 response.
- Do not hardcode specific fixture answers unless the failed
trajectory itself shows how to derive them from tools/files.


[MiniWoB++]

Use the trajectory as evidence of which browser skills the agent
lacked or misapplied rather than guessing bids.

CRITICAL RULES:
- Do not copy specific numeric bids, autocomplete picks, or other
seed-specific values from the trajectory into the response. Bids
change across seeds; teach the agent how to FIND the right element
by role + name, not which integer to type.
```

G.4 Adapter Retry and Base Fallback

If the retry with an adapter skill is unsuccessful, a final retry fallback is attempted with the base agent and a short instruction is appended to the original system prompt:

Retry prompt appended to the user prompt

```
You already tried this task once and produced an incorrect answer.
Please try again.
```

For Claw-Eval, the final sentence also asks the agent to verify the result before stopping. During the adapter-guided retry, the selected modular adapter is active. If that attempt fails, the adapter is fully unloaded and the same task and retry suffix are presented to the base model. No reflection examples or adapter outputs are inserted into the fallback context. This isolates the fallback as an ordinary base-model retry while preserving the same failure-conditioned instruction.

H QUALITATIVE ANALYSIS OF FORWARD TRANSFER

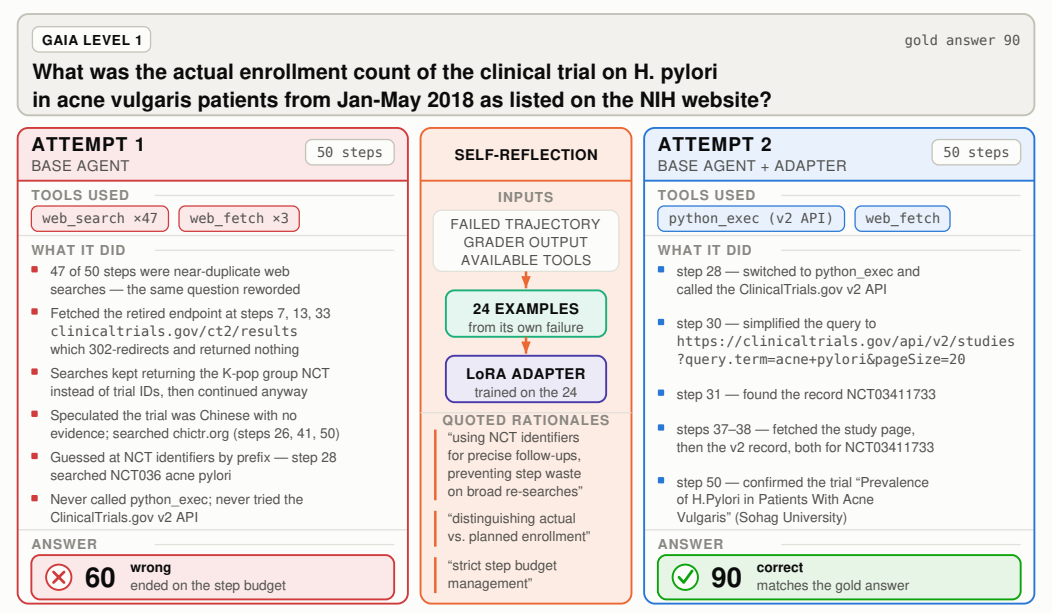


Figure 15: Self-Direction **recovers failing tasks.** A qualitative example showing how Self-Direction reflects on the failure, correctly identifies overuse of invalid tools, and generates corrective examples that ultimately help the agent succeed on a GAIA Level-1 task.

The quantitative transfer results show that stored adapter skills can improve later tasks. We now examine what these adapters appear to transfer qualitatively. Across GAIA and Claw-Eval, the strongest examples share a procedural structure – such as pagination, cross-system reconciliation, or exact-value verification – even when their topics, difficulty levels, or task categories differ. Fig. 15 shows Self-Directed recovery of a Level 1 task in GAIA. Examples of generated reflections are shown in the following section (Appendix I).

H.1 TRANSFER WITHIN AN ONLINE GAIA STREAM

The GAIA deployment experiment in Section 4.4 interleaves all 165 tasks across difficulty levels and enables the agent to retrieve only adapters learned earlier in the stream. Across the three routing thresholds, existing adapters are selected for 282 later failures and produce 41 successful adapter-conditioned retries. Thirty of these successes cross GAIA difficulty levels. Same-level and cross-level retrievals have nearly identical success rates, 14.7% and 14.5%, suggesting that benchmark difficulty is not the natural boundary of a reusable skill.

Table 14 presents representative transfers. In each case, the source and target differ in surface content but require a related sequence of actions or verification strategy.

These examples suggest that the adapters can encode procedures rather than task-specific guidance. The spreadsheet adapter, for example, transfers from tabular sales data to numerical values embedded in an image. The shared behavior is not the input modality but the discipline of extracting, checking, computing, and formatting. Similarly, the ORCID adapter transfers from publication records to Wikipedia revisions because both tasks require complete pagination and verified counting.

An adapter can also become useful even when it does not repair its source task. The first ORCID adapter fails its own adapter-guided retry, yet later produces 14 successful retries in the most permissive routing condition. Success on the source evaluation is, therefore, not a complete measure of whether the learned procedure will be useful elsewhere.

Table 14: **Representative transfers in the shuffled GAIA stream.** Every source task occurs before its target in the stream. The examples illustrate procedural reuse across topics, modalities, and difficulty levels.

Source adapter	Later target	Shared procedure	Observed recovery
ORCID publication aggregation (Level 3)	Wikipedia revision counting (Level 2)	Traverse a complete remote history, handle pagination, apply a date constraint, and verify completeness before counting.	Initial answer: 2610. Adapter answer: 2732 (correct).
BBC Earth video lookup (Level 1)	Exact quotation from another video (Level 1)	Identify the correct video and extract a localized fact using descriptions, timestamps, comments, or other supporting metadata.	The initial execution exhausted its step budget; the adapter returned the exact quotation.
Spreadsheet aggregation (Level 1)	Statistics over values extracted from an image (Level 2)	Extract structured values, inspect intermediate data, compute with a tool, verify the calculation, and format only after validation.	Initial answer: 2.404. Adapter answer: 17.056 (correct).
Caesar-cipher decoding (Level 2)	Text-grid reconstruction (Level 1)	Apply a text transformation, check linguistic coherence, and preserve exact capitalization, spacing, and output constraints.	The initial execution recovered the meaning but violated formatting; the adapter returned the exact required sentence.

H.2 Cross-Category Transfer on Claw-Eval

The Claw-Eval experiment in Section 4.4 provides a cleaner category shift. We freeze a library of three adapters learned from English operations (ops) tasks and evaluate it on 24 unseen workflow tasks in the original benchmark. The base model solves $18/24$ tasks. On the six initially encountered failures, retrieved operations adapters solve three, increasing final accuracy to $21/24$ (87.5%).

The successful transfers involve related coordination procedures despite the source and target belonging to different benchmark categories. Table 15 summarizes the three recovered workflows.

Table 15: **Qualitative cross-category transfer from Claw-Eval operations tasks to unseen workflows.** Scores are assigned by the Claw-Eval grader. No workflow adapter is trained before evaluation.

Source operations skill	Target workflow	Shared procedure	Grader Score change
Scheduled-task management	Weekly meeting tracking	Collect status across multiple records, identify missing or incomplete actions, and produce a structured progress report.	$0.413 \rightarrow 0.946$
Month-end reconciliation	Task backtracking	Reconcile evidence across emails, task lists, and meeting notes; identify discrepancies and report status, blockers, and next actions.	$0.674 \rightarrow 0.894$
Scheduled-task management	Multi-project progress reporting	Aggregate task status across several projects, identify risks, and organize the results into separate actionable reports.	$0.586 \rightarrow 0.890$

The scheduled-task adapter transfers to two workflows because all three tasks require auditing distributed state, identifying incomplete work, and producing a structured report. The reconciliation adapter similarly transfers beyond finance: Its useful behavior is gathering corresponding records from several systems, resolving inconsistencies, and summarizing the resulting status.

Interpretation. Together, these examples support the view that modular adapter skills can preserve helpful procedural lessons and make them available in new contexts. The strongest cases are those in which the source and target share an identifiable workflow, even when their topics, modalities, difficulty levels, or benchmark categories differ.

I QUALITATIVE EXAMPLES OF FAILURE-CONDITIONED Self-Direction

The examples generated during reflection do not simply reproduce the failed task, nor are they full, expensive rollouts. Instead, they express the underlying failure as a reusable procedure across varied situations. Next, we trace one task from each primary benchmark for which the base execution failed and the adapter-guided retry succeeded. Excerpts are shortened for readability but are drawn directly from the saved run artifacts.

GAIA: Structured extraction from audio.

Task. The agent was asked to transcribe an attached strawberry-pie recipe and return only the filling ingredients as an alphabetized, comma-separated list without measurements.

Failure. The base agent returned generic ingredient names such as "lemon juice" and "vanilla extract," omitting the required qualifiers "freshly squeezed" and "pure," and introduced extraneous formatting. The answer, therefore, did not match the required list and was recorded as incorrect.

Generated lesson. Self-reflection produced examples covering a general audio-to-structured-data procedure: Transcribe the recording, isolate the requested section, exclude irrelevant categories, remove quantities, normalize and sort the extracted items, and verify the required output format. One example asks the agent to extract only the dry ingredients from a baking recording while excluding measurements and returning an alphabetized list.

Outcome. The resulting adapter returned the exact expected ingredient list and recovered the task.

Claw-Eval: Completing evidence-grounded research.

Task. The agent was asked to evaluate a migration from Redis to Valkey, covering the Redis license change, Valkey's governance, technical compatibility, cloud support, migration experiences, and project health.

Failure. The base execution spent its trajectory repeatedly searching and fetching sources but ended without synthesizing the requested report.

Generated lesson. The reflective examples taught a more robust research workflow: Broaden queries when search returns no results, pivot to authoritative URLs when retrieval fails, cross-check technical claims, map collected evidence to every requested section, and reserve sufficient time for synthesis. For example, one generated instruction asks what to do when an initial technical search returns no results; its response recommends reformulating the query, trying authoritative sources directly, and avoiding unsupported conclusions.

Outcome. The adapter-guided retry completed the requested technical evaluation and passed the Claw-Eval verifier.

Terminal-Bench 2.1: Verifying a multi-service deployment.

Task. The agent had to configure a Git server over SSH, deploy the `main` and `dev` branches through a post-receive hook, and serve both through distinct HTTPS endpoints.

Failure. The base execution exhausted its step budget without establishing a working end-to-end deployment. The subsequent test encountered an invalid repository state—including "remote HEAD refers to nonexistent ref"—and failed.

Generated lesson. Self-reflection produced examples emphasizing observable verification rather than assumed success: Inspect effective SSH and Nginx configurations, confirm canonical ports, validate the bare repository with `git ls-remote`, instrument the deployment hook, and reproduce the complete clone-commit-push-fetch workflow from a clean directory. One example explicitly instructs the agent to execute the evaluator's full workflow and compare the returned page contents against the expected strings before declaring completion.

Outcome. The adapter-guided retry correctly deployed both branches and the complete evaluator test passed.

Together, these cases illustrate that Self-Direction extracts different forms of procedural knowledge from different failures: precise information extraction on GAIA, evidence-grounded task completion on Claw-Eval, and systematic end-to-end verification on Terminal-Bench 2.1.

J WHY SVD INITIALIZATION

The SVD initialization of matrix A offers two benefits: It further reduces the trainable parameters of an already lightweight adapter and anchors adaptation in a high-energy subspace of the pretrained model (Ren et al., 2026). The zero initialization of matrix B makes the adapter an exact no-op before training, while freezing the base model confines the update to a small, failure-specific parameter space, only training $\sim \mathbf{0.02}\%$ of the total additional parameters on the base model. The failure type also determines where the repair is applied: Planning and budget failures target attention projections, whereas incorrect-answer failures target MLP projections.

In our final configuration, A_ℓ is initialized from a rank-(r) high-energy right-singular subspace of the frozen base weight, $B_{i,\ell}$ is initialized to zero, and A_ℓ remains frozen. Only $B_{i,\ell}$ is optimized:

$$\mathcal{D}_i = \{(u_j, v_j)\}_{j=1}^{N_i}, \qquad \phi_i^\star = \arg\min_{\phi_i} \frac{1}{N_i} \sum_{j=1}^{N_i} [-\log p_{\theta,\phi_i}(v_j \mid u_j)], \qquad \theta \text{ fixed.}$$

Here, ϕ_i denotes the trainable parameters of the new adapter. u is a training instruction and v is its desired procedural response over N corrective examples. $p_{\theta,\phi_i}(v \mid u)$ is the probability the adapted model assigns to response v given instruction u.

SVD provides a shared coordinate system for repair. For an adapted weight matrix W, each skill takes the form

$$W_i = W + \Delta W_i, \qquad \Delta W_i = B_i A,$$

where A is initialized from a rank-16 slice of the base weight's high-energy spectral band and remains fixed during training. Only B_i is learned from the reflective examples for failure i. The selected subspaces capture approximately 16-22$\times$ more base-weight energy than random subspaces of the same dimension. Thus, different skills learn within a compact set of input features that are already prominent in the pretrained model.

Importantly, the learned update does not simply amplify the corresponding base weights. Although its input directions are inherited from the base model, its output directions are largely new. One interpretation is that the adapter recognizes familiar pretrained features but learns to route them toward behavior that would have prevented the observed failure.