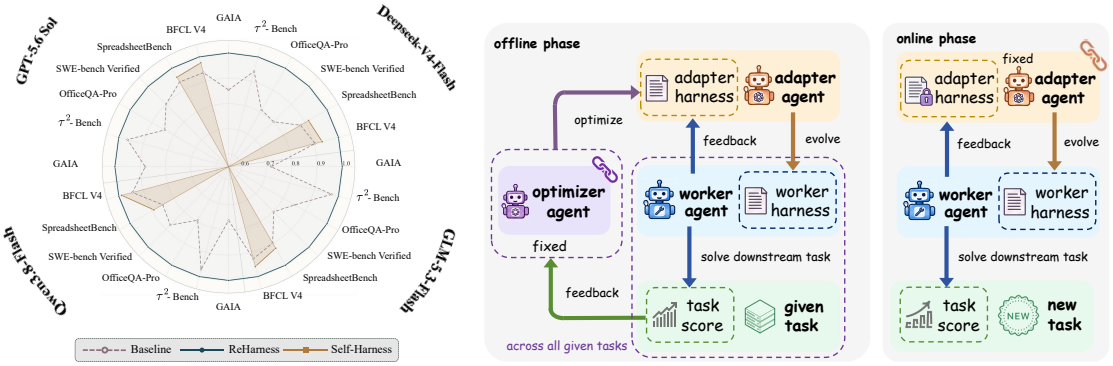
REHARNESS: LEARNING REUSABLE HARNESS EVOLUTION ACROSS TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents increasingly rely on programmable harnesses to coordinate reasoning, tool use, and execution. Existing harness evolution methods typically optimize a task-specific harness from scratch for each new task, requiring substantial task-specific data and evolution steps. In this work, we identify that harness evolution exhibits reusable structure across heterogeneous tasks: independently evolved harnesses repeatedly discover similar improvement strategies, although the resulting harnesses remain task-specific. Building on this insight, we propose ReHarness, a reusable harness evolution framework that learns how to evolve across tasks. ReHarness learns a transferable evolution strategy from multiple offline-given tasks and applies it to rapidly adapt task-specific worker harnesses to new tasks using only a few examples. Experiments across four LLMs and six benchmarks demonstrate that ReHarness consistently improves agent performance on four new tasks not used during the offline phase, while outperforming the task-specific evolution method Self-Harness on two tasks from the offline suite under the same data budget. Code and models will be made publicly available.



(a) Relative performance

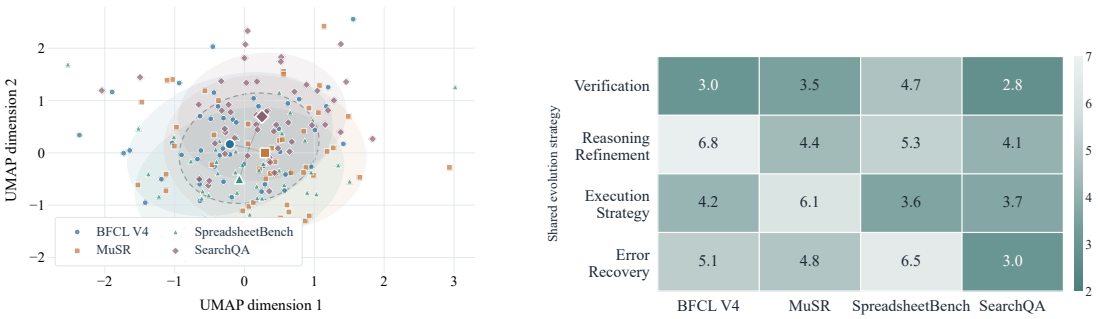
(b) Overview of the three agent roles in ReHarness

Figure 1: **Overview and Performance of ReHarness.** ReHarness consistently improves performance across models and benchmarks, while enabling rapid harness adaptation through reusable evolution strategies.

1 INTRODUCTION

Large language model (LLM) agents have demonstrated strong capabilities across diverse tasks by augmenting language models with external tools, memory, and execution mechanisms (Yao et al., 2023; Wang et al., 2024b; Xi et al., 2025). Their performance depends not only on the underlying model but also on the

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.



(a) (b)

Figure 2: **Empirical analysis of harness evolution across heterogeneous tasks.** (a) UMAP visualization of independently collected evolution trajectories from four tasks. The substantial overlap in the embedding space suggests cross-task similarity in harness evolution. (b) Average evolution step at which four shared strategies are first discovered on each task. These strategies repeatedly emerge across tasks, but are often discovered only after several task-specific evolution steps. See Appendix D.1 and D.2 for more details.

surrounding harness that organizes how the model receives instructions, utilizes tools, maintains state, and executes actions. Consequently, evolving the harness provides a way to improve agent performance without changing model parameters (Yang et al., 2024; Lee et al., 2026).

However, existing harness evolution methods (Lee et al., 2026; Lin et al., 2026; Zhang et al., 2026a) evolve a task-specific harness independently for each downstream task. Although effective on individual tasks, this task-by-task paradigm has two limitations. First, evolution on each new task starts largely from scratch, repeatedly rediscovering evolution patterns that may be shared across tasks and requiring substantial task-specific data. Second, since each evolution process is optimized for a particular task distribution, the resulting harness tends to specialize to that task, potentially limiting its generalization to tasks from other domains.

These limitations motivate us to explore whether harness evolution can be made reusable across tasks. Ideally, evolution should learn generalizable strategies from previous tasks while adapting harnesses to new task requirements. This raises a question: *Can we learn a reusable harness evolution strategy and apply it to efficiently evolve task-specific harnesses for new tasks from only a few examples?*

Key Observations. To make harness evolution reusable across tasks, we first investigate whether harness evolution for the same model exhibits shared structure across heterogeneous tasks. Specifically, we conduct a pilot study on four heterogeneous tasks using DeepSeek-V4-Flash (DeepSeek-AI et al., 2026). For each task, we independently perform Self-Harness (Zhang et al., 2026a) evolution using only its own task data, recording 50 intermediate harnesses per task. As shown in Fig. 2(a), evolution trajectories from different tasks substantially overlap in the embedding space, suggesting shared structures across task-specific evolution processes. This suggests that the shared structure may reflect reusable evolution patterns induced by the underlying model, providing a reusable foundation for adapting harnesses across different downstream tasks.

We further examine whether task-specific evolution repeatedly discovers similar strategies across tasks. To this end, we identify strategies that emerge in all four tasks and track when they are first discovered during evolution on each task. As shown in Fig. 2(b), strategies such as verification, reasoning refinement, execution strategy, and error recovery consistently emerge across all tasks, typically after several evolution steps. The recurrence of these strategies across different tasks suggests that harness evolution contains generalizable patterns intrinsic to the model beyond individual task distributions. Leveraging such reusable strategies to guide harness evolution may therefore enable better generalization to new tasks.

Our Solution. In this work, we propose ReHarness, a **re**usable **harness** evolution framework that learns how to evolve across tasks. As illustrated in Fig. 1(b), ReHarness consists of three agents with distinct roles. The *worker agent* solves the downstream task with a task-specific worker harness, while the *adapter agent* evolves this harness based on task execution feedback. The adapter agent's evolution behavior is itself governed by a programmable adapter harness, which is further evolved across tasks by the *optimizer agent*. By separating task-level harness adaptation from cross-task learning of the evolution strategy, this hierarchy enables evolution strategies themselves to be learned and reused across tasks.

We realize this hierarchy through offline and online phases that separate learning how to evolve from applying the learned evolution strategy. During the offline phase, ReHarness optimizes the adapter harness through three loops: the inner loop adapts task-specific worker harnesses and provides performance feedback, the outer loop uses this feedback to optimize the adapter harness within each batch, and the cross-batch loop carries forward the optimized adapter harness and evolution experience across batches. During the online phase, the optimizer agent is removed and the learned adapter harness is fixed, enabling the learned evolution strategy to directly evolve worker harnesses for new tasks from only a few examples.

With this design, ReHarness turns harness evolution from a task-by-task optimization process into a general adaptation capability. In a sense analogous to meta-learning (Finn et al., 2017), agents learn a reusable evolution strategy and rapidly specialize to new tasks on demand. Extensive experiments conducted on four LLMs across diverse benchmarks demonstrate the effectiveness and generalization capabilities of ReHarness. Our contributions are summarized as follows:

- We identify a shared structure in harness evolution across tasks. We reveal that similar changes repeatedly emerge when harnesses are evolved for different tasks, indicating that part of the evolution strategies can be learned and reused across tasks, laying the foundation for cross-task generalization.
- We propose ReHarness, a reusable harness evolution framework. Through a hierarchical design, ReHarness learns from harness evolution across given tasks and applies the learned evolution strategy to new tasks, enabling cross-task generalization beyond task-specific optimization.
- We conduct extensive evaluations on four LLMs, including DeepSeek-V4-Flash (DeepSeek-AI et al., 2026), GLM-5.3-Flash (GLM-5-Team et al., 2026), Qwen3.8-Flash (Qiu et al., 2026), and GPT-5.6 Sol. Results across six benchmarks, covering four new tasks not used offline and two offline-given tasks, demonstrate ReHarness's effectiveness and its ability to generalize to new tasks.

2 Background and Motivation

2.1 Preliminaries

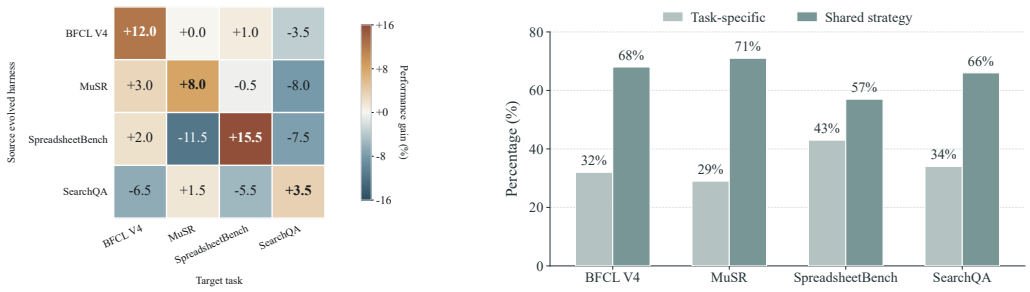
Harnessed LLM Agent. Let M denote a frozen large language model and h denote its programmable harness. Given an input x, the behavior of the agent can be represented as

$$y = f_{M,h}(x), \tag{1}$$

where $f_{M,h}$ denotes the agent composed jointly by the frozen model M and harness h. While M remains unchanged, different choices of h can induce different agent behaviors. For a task τ and an evaluation set $\mathcal{D}$, we denote the performance of harness h as

$$J_\tau(h;\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,z)\in\mathcal{D}} s_\tau\left(f_{M,h}(x), z\right), \tag{2}$$

where z is the task-specific ground truth, and s_τ is the corresponding scoring function. Harness optimization therefore aims to improve $J_\tau(h;\mathcal{D})$ by changing h while keeping M fixed.



(a) (b)

Figure 3: **Empirical analysis of harness transfer and evolution strategies across tasks.** (a) Performance gain obtained by directly transferring an evolved harness from each source task to different target tasks. (b) Proportion of shared and task-specific strategies identified during harness evolution across four tasks. See Appendix D.3 and D.4 for more details.

Harness Evolution. Harness evolution (Lee et al., 2026; Zhang et al., 2026a) performs this optimization by iteratively modifying harnesses based on execution feedback. For task τ, let h_τ^0 denote the initial harness and ξ_τ^k denote feedback collected from h_τ^k at evolution step k. An evolver π produces a harness modification

$$\delta_\tau^k = \pi\left(h_\tau^k, \xi_\tau^k\right), \tag{3}$$

where δ_τ^k specifies the modification made to current harness. The harness is then updated as h_τ^{k+1} using δ_τ^k. Repeating this procedure yields an evolution trajectory

$$h_\tau^0 \xrightarrow{\delta_\tau^0} h_\tau^1 \xrightarrow{\delta_\tau^1} h_\tau^2 \xrightarrow{\delta_\tau^2} \cdots \xrightarrow{\delta_\tau^{K-1}} h_\tau^K, \tag{4}$$

where K denotes the number of evolution steps. Conventional harness evolution aims to improve $J_\tau(h; \mathcal{D})$ by directly evolving a task-specific harness along this trajectory. Our goal is instead to learn a reusable evolver π that can efficiently evolve task-specific harnesses for new tasks.

2.2 Key Observations

Evolved harnesses cannot be directly reused across tasks. Given the shared evolution strategies observed in Fig. 2, the most straightforward approach is to directly reuse a harness evolved on one task for other tasks, without evolving a new harness for each task. However, we find that this simple approach is ineffective. Specifically, we evaluate each evolved harness on its source task and directly transfer it to the other tasks. As shown in Fig. 3(a), evolved harnesses consistently improve their source tasks, achieving an average gain of 9.8%. However, when directly transferred to other tasks, the same harnesses yield an average gain of -3.0% and frequently degrade performance. This contrast shows that, *although evolution strategies can be shared across tasks, the resulting harnesses remain highly task-specific and don't transfer directly to other tasks.*

Evolution involves both shared strategies and task-specific adaptation. The failure of direct harness transfer reveals a limitation of existing task-specific evolution. While task-specific optimization can produce a harness that performs well on its source task, the resulting harness tends to specialize to the data distribution and generalizes poorly to other tasks. We argue that an ideal harness evolution should instead learn from behavioral patterns of the model that persist across different task distributions. This view is supported by Fig. 3(b), where a substantial portion of evolution strategies are repeatedly discovered across tasks despite their different task-specific requirements. These observations suggest that harness evolution involves two

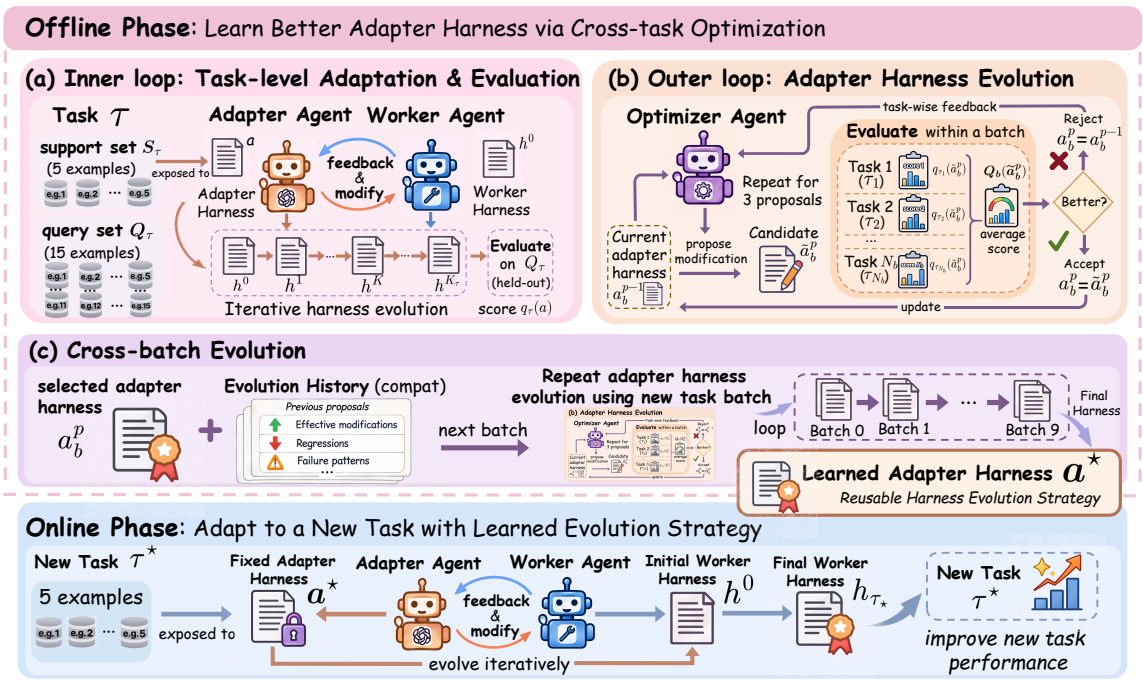


Figure 4: **Overview of ReHarness. Offline Phase** (Top): For each cross-domain task batch, *(a) Task-level Adaptation & Evaluation* evolves task-specific worker harnesses from support examples and evaluates the resulting adaptation on held-out queries in an inner loop. Based on these outcomes, *(b) Adapter Harness Evolution* iteratively modifies the adapter harness in an outer loop, while *(c) Cross-batch Evolution* carries the selected adapter harness and compact evolution history across batches, progressively learning a reusable harness evolution strategy in an adapter harness. **Online Phase** (Bottom): For a new task, the learned adapter harness $a^\star$ is fixed and directly reuses the evolution strategy acquired offline to evolve a worker harness from only a few task-specific examples, improving its performance on a new task.

distinct but related levels: *how to evolve*, which captures general ways of improving model behavior and can be learned across tasks, and *what is evolved*, which should adapt to the specific requirements of each task.

Taken together, we decompose harness evolution into two levels. At the cross-task level, we learn how to evolve by accumulating reusable evolution strategies across different task distributions. At the task level, we apply these strategies to determine what to evolve according to the specific requirements of each task. Based on this, we design ReHarness, a reusable harness evolution framework that learns evolution strategies across tasks and rapidly evolves task-specific harnesses for new tasks from only a few examples.

3 Methodology

3.1 Overview

In this section, we introduce ReHarness, which aims to learn a reusable harness evolution strategy from given tasks and applies it to rapidly evolve task-specific worker harnesses for new tasks from only a few examples. As illustrated in Fig. 1 (b) and 4, ReHarness involves three types of agents and operates in two phases.

Algorithm 1 Offline phase of ReHarness

```
Initialize adapter harness a
for each batch of tasks B do                                   ▷ Cross-batch evolution
    for each optimizer proposal do                             ▷ Outer loop: adapter harness evolution
        for each task τ ∈ B do                                 ▷ Inner loop: task-level adaptation and evaluation
            Evolve worker harness on support set S_τ and evaluate on query set Q_τ
        end for
        Accept improved adapter harness
    end for
end for
Return learned adapter harness a*
```

For the three agents, the **worker agent** directly solves a downstream task using a task-specific worker harness. The **adapter agent** serves as the harness evolver π introduced in Sec. 2.1. Given a few examples from the current task, it executes the worker agent on these examples, extracts feedback, and iteratively modifies the worker harness to better adapt it to this task. The adapter agent's behavior is governed by an adapter harness. The **optimizer agent**, in turn, evolves this adapter harness across given tasks. ReHarness thus separates task-level evolution of worker harnesses from cross-task evolution of the strategy used to evolve them.

The two phases differ in whether the adapter harness itself is evolved. During the **offline phase**, which consists of three loops, i.e., the inner loop, outer loop, and cross-batch loop, as summarized in Algorithm 1, the adapter agent evolves task-specific worker harnesses, while the optimizer agent optimizes the adapter harness based on their performance and carries forward evolution experience across batches. We detail this process in Sec. 3.2. During the **online phase**, the optimizer agent is removed and the learned adapter harness is fixed. Given a new task and only a few task-specific examples, the adapter agent directly evolves a task-specific worker harness within a few examples and adaptation tries. We describe this process in Sec. 3.3.

The details of ReHarness are presented in the following, with the complete algorithm provided in Appendix A.

3.2 Offline Phase

Inner Loop: task-level adaptation and evaluation. As shown in Fig. 4(a), the inner loop performs task-level adaptation. For each task τ, we adopt an episodic formulation inspired by meta-learning (Finn et al., 2017), where task data is divided into a support set $\mathcal{S}_\tau$ and a disjoint query set $\mathcal{Q}_\tau$. The support set provides few task-specific examples for evolving the worker harness, while the query set is held out for evaluating the resulting harness. In our experiments, we use $|\mathcal{S}_\tau| = 5$ and $|\mathcal{Q}_\tau| = 15$ (see ablations in Sec. 4.2).

Given an adapter harness a, the adapter agent first evolves an initial worker harness h_τ^0 using only $\mathcal{S}_\tau$. At each adaptation try, the worker agent runs on the support examples, and the resulting execution feedback is provided to the adapter agent to guide the next worker harness modification. This process continues until the adapter agent decides to stop or reaches the maximum tries budget $K_{\max}$. We set $K_{\max} = 5$ to provide sufficient opportunities for iterative modification while keeping task-level adaptation light. Let $K_{\tau,a} \leq K_{\max}$ denote the number of adaptation tries performed under adapter harness a, and let $h_{\tau,a}^{K_{\tau,a}}$ denote the resulting worker harness. We then evaluate the quality of a on the held-out query set as

$$q_\tau(a) = J_\tau\left(h_{\tau,a}^{K_{\tau,a}}; \mathcal{Q}_\tau\right), \tag{5}$$

where $q_\tau(a)$ denotes the query performance after adapting the worker harness with a, measuring whether the evolution strategy encoded by a generalizes from the support set to the held-out query set.

Outer loop: adapter harness evolution. The outer loop (Fig. 4(b)) operates over each offline batch $\mathcal{B}_b$, where the fixed optimizer agent sequentially proposes modifications to the current adapter harness. We define

$\mathcal{B}_b = \{\tau_1, \ldots, \tau_{N_b}\}$ as the set of task episodes included in batch b, where the tasks are from different domains and $N_b = |\mathcal{B}_b|$ denotes the number of task types. For every task $\tau \in \mathcal{B}_b$, we use the same support and query configuration. Let a_b^{p-1} denote the adapter harness before the p-th proposal in batch b, and let $\widetilde{a}_b^p$ denote the proposed candidate. To evaluate the candidate, we use it to evolve worker harnesses for all tasks in $\mathcal{B}_b$ and average their query scores across all tasks in the batch as

$$Q_b(a) = \frac{1}{|\mathcal{B}_b|} \sum_{\tau \in \mathcal{B}_b} q_\tau(a), \tag{6}$$

where $Q_b(a)$ denotes the average query performance across all tasks in batch b and serves as the outer optimization objective. The candidate adapter harness is retained only when it improves the query score:

$$a_b^p = \begin{cases} \widetilde{a}_b^p, & Q_b(\widetilde{a}_b^p) > Q_b(a_b^{p-1}), \\ a_b^{p-1}, & Q_b(\widetilde{a}_b^p) \le Q_b(a_b^{p-1}). \end{cases} \tag{7}$$

Besides the average score, we retain the performance change on each task as diagnostic feedback, allowing the optimizer agent to identify which modifications benefit multiple task types and which cause regressions on specific tasks. We use $P = 3$ sequential proposals per batch, balancing adaptation quality and costs.

Cross-batch evolution. The selected adapter harness is carried forward across offline batches,

$$a_{b+1}^0 = a_b^P, \tag{8}$$

where a_b^P is the final adapter harness in batch b. In addition, the optimizer agent retains a compact history of previous proposals, including modification mechanisms, observed gains or regressions, and recurring failures. Rather than replaying full optimization trajectories, only high-signal summaries are preserved, allowing later proposals to build on previous evolution experience without redundant context. Through repeated optimization over cross-task batches, the adapter harness progressively captures reusable evolution strategies, yielding the final adapter harness $a^\star = a_B^P$ after the last batch B. This cross-batch loop is shown in Fig. 4(c).

3.3 Online Phase

After offline optimization, the optimizer agent is removed and the final adapter harness $a^\star$ is fixed. For a new task $\tau^\star$, a worker harness is initialized following the same procedure used during the offline phase. The adapter agent with learned adapter harness $a^\star$ then operates solely on a small support set $\mathcal{S}_{\tau^\star}$ to iteratively evolve the worker harness. Neither a query set nor outer optimization is required online, allowing $a^\star$ to directly transfer the learned cross-task evolution strategy to new tasks from limited examples.

4 Experiments

To comprehensively assess the effectiveness of ReHarness, we evolve the adapter harness on four heterogeneous tasks: BFCL V4 (Patil et al., 2025), MuSR (Sprague et al., 2024), SpreadsheetBench (Ma et al., 2024), and SearchQA (Dunn et al., 2017), using $|\mathcal{S}_\tau| = 5$ support examples and $|\mathcal{Q}_\tau| = 15$ query examples for each task over 10 offline batches. We independently evolve adapter harnesses with DeepSeek-V4-Flash (DeepSeek-AI et al., 2026), GLM-5.3-Flash (GLM-5-Team et al., 2026), Qwen3.8-Flash (Qiu et al., 2026), and GPT-5.6 Sol. We then evaluate them on four new tasks, GAIA (Mialon et al., 2024), OfficeQA-Pro (Opsahl-Ong et al., 2026), τ^2-Bench (Barres et al., 2026), and SWE-bench Verified (Jimenez et al., 2024), none of which is used during the offline phase, to assess their ability to adapt worker harnesses to new tasks. We further compare our method with the task-specific evolution method Self-Harness (Zhang et al., 2026a) on SpreadsheetBench and BFCL V4 under the same evolution data budget. All reported results are averaged over 3 independent runs. Detailed experimental settings and configurations are provided in Appendix D.5 and D.7.

Table 1: **Main results of ReHarness across four benchmarks.** We compare ReHarness with the standard baseline setting across four models. We report the mean performance ± standard deviation over three runs. Numbers in green denote the absolute improvement of ReHarness over baseline.

Model	Method	GAIA	τ^2-Bench	OfficeQA-Pro	SWE-bench Verified	Avg.
DeepSeek-V4-Flash	Baseline	28.7 ± 0.8	79.5 ± 1.1	36.7 ± 2.1	26.7 ± 1.0	42.9 ± 0.7
	ReHarness (ours)	**33.6** ± 1.6 **(+4.9)**	**84.5** ± 2.8 **(+5.0)**	**45.3** ± 2.1 **(+8.6)**	**33.3** ± 1.0 **(+6.6)**	**49.2** ± 1.5 **(+6.3)**
GLM-5.3-Flash	Baseline	29.8 ± 1.1	87.1 ± 1.8	48.4 ± 0.8	32.1 ± 2.1	49.4 ± 0.8
	ReHarness (ours)	**41.5** ± 1.2 **(+11.7)**	**89.6** ± 0.4 **(+2.5)**	**56.8** ± 2.0 **(+8.4)**	**40.0** ± 1.8 **(+7.9)**	**57.0** ± 0.5 **(+7.6)**
Qwen3.8-Flash	Baseline	27.9 ± 0.9	86.9 ± 0.5	29.4 ± 1.2	29.7 ± 1.0	43.5 ± 0.4
	ReHarness (ours)	**36.6** ± 1.7 **(+8.7)**	**89.5** ± 0.8 **(+2.6)**	**37.0** ± 1.2 **(+7.6)**	**33.9** ± 1.0 **(+4.2)**	**49.3** ± 0.9 **(+5.8)**
GPT-5.6 Sol	Baseline	44.0 ± 3.0	67.8 ± 1.2	42.5 ± 1.5	44.2 ± 4.6	49.6 ± 2.6
	ReHarness (ours)	**50.0** ± 1.9 **(+6.0)**	**70.3** ± 1.0 **(+2.5)**	**50.5** ± 1.5 **(+8.0)**	**49.1** ± 3.2 **(+4.9)**	**55.0** ± 1.9 **(+5.4)**

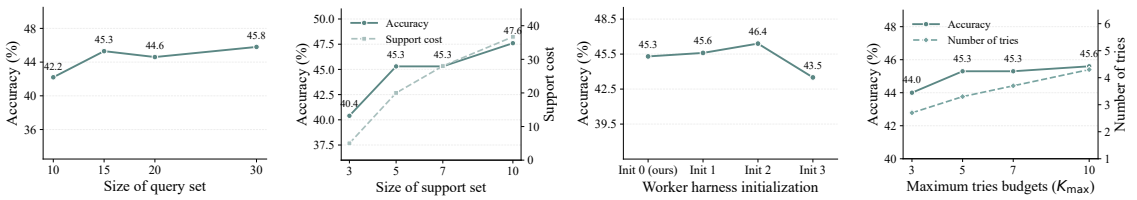
Table 2: **Main results on SpreadsheetBench and BFCL V4 benchmarks.** We compare baseline, ReHarness, and Self-Harness across four models. We report the mean performance ± standard deviation over three runs. Numbers in green denote the absolute improvements over baseline.

Model	Benchmark	Baseline	ReHarness (ours)	Self-Harness
DeepSeek-V4-Flash	SpreadsheetBench	47.7 ± 0.6	54.3 ± 0.6 (+6.6)	49.7 ± 0.6 (+2.0)
	BFCL V4	54.3 ± 0.6	60.0 ± 1.7 (+5.7)	56.0 ± 1.0 (+1.7)
GLM-5.3-Flash	SpreadsheetBench	53.3 ± 0.6	59.3 ± 0.6 (+6.0)	54.7 ± 0.6 (+1.4)
	BFCL V4	58.7 ± 1.2	63.3 ± 0.6 (+4.6)	60.7 ± 0.6 (+2.0)
Qwen3.8-Flash	SpreadsheetBench	42.3 ± 1.5	49.7 ± 0.6 (+7.4)	44.3 ± 0.6 (+2.0)
	BFCL V4	55.7 ± 0.6	59.0 ± 1.0 (+3.3)	58.3 ± 0.6 (+2.6)
GPT-5.6 Sol	SpreadsheetBench	68.7 ± 1.3	73.3 ± 2.3 (+4.6)	70.3 ± 1.3 (+1.6)
	BFCL V4	77.0 ± 1.0	82.3 ± 1.3 (+5.3)	80.3 ± 2.3 (+3.3)

4.1 Main Results

ReHarness enables effective harness adaptation to new tasks without requiring large-scale task-specific harness evolution. As shown in Table 1, all four evaluation benchmarks are unseen during the offline phase, and ReHarness requires only 5 support examples to adapt the worker harness to each target task. Under this lightweight adaptation setting, ReHarness consistently improves over the baseline across all four benchmarks and all four models, achieving an average absolute gain of **6.3%** and a maximum gain of **11.7%**. Notably, while existing harness evolution methods cannot be directly applied under our configuration (see Appendix C.1 for reasons), ReHarness transfers the learned evolution strategy to new tasks and enables effective adaptation from only a few support examples. These results demonstrate that ReHarness learns a reusable harness evolution strategy that generalizes across heterogeneous tasks and diverse LLMs.

We further compare ReHarness with Self-Harness under the same data budget and task-specific evolution setting (see Appendix D.7). As shown in Table 2, ReHarness consistently outperforms Self-Harness across both benchmarks and all four models. This demonstrates that ReHarness remains effective even when operating under the conventional task-specific evolution paradigm.



(a) (b) (c) (d)

Figure 5: **Ablation results on support set size, query set size, worker harness initialization, and maximum tries budget.** All experiments are conducted on OfficeQA-Pro with DeepSeek-V4-Flash. (a) Accuracy across different $|\mathcal{Q}_\tau|$ with $|\mathcal{S}_\tau| = 5$. (b) Accuracy and support cost across different $|\mathcal{S}_\tau|$ with $|\mathcal{Q}_\tau| = 15$. (c) Accuracy across worker harness initializations. (d) Accuracy and actual number of tries K_τ across $K_{\max}$.

4.2 Ablations and Analysis

All ablation experiments below are conducted on OfficeQA-Pro with DeepSeek-V4-Flash and all results are averaged over 3 independent runs. Additional ablations on the role of the learned adapter harness, cross-batch evolution history, and offline-given task diversity are provided in Appendix C.2, C.3 and C.4, respectively.

Effect of query set size. We first study the effect of query set size $|\mathcal{Q}_\tau|$ with fixed $|\mathcal{S}_\tau| = 5$. As shown in Fig. 5(a), using only 10 query samples reduces accuracy to 42.2%. In contrast, performance becomes stable when $|\mathcal{Q}_\tau|$ reaches 15, achieving 45.3%, 44.6%, and 45.8% with 15, 20, and 30 queries, respectively. Since the query set provides held-out feedback for evaluating adapter harness updates, an insufficient query size introduces noisy optimization signals and degrades downstream performance, while additional queries provide limited benefit. We therefore set $|\mathcal{Q}_\tau| = 15$ in our experiment.

Effect of support set size. We next vary the support set size $|\mathcal{S}_\tau|$ while fixing $|\mathcal{Q}_\tau| = 15$. As shown in Fig. 5(b), increasing support examples generally improves downstream accuracy. In particular, increasing $|\mathcal{S}_\tau|$ from 3 to 5 improves accuracy from 40.4% to 45.3%. Further increasing the support set brings smaller gains relative to additional adaptation cost: 10 support examples achieve 47.6% accuracy with higher support cost. We therefore use $|\mathcal{S}_\tau| = 5$ as the default setting, balancing adaptation performance and cost.

Sensitivity to worker harness initialization. We examine whether the learned evolution strategy is sensitive to worker harness initialization. In addition to our designed initialization, we construct three alternative initializations using an LLM, with the detailed procedure provided in Appendix D.8. As shown in Fig. 5(c), accuracies remain stable across initializations, ranging from 43.5% to 46.4%. This suggests that ReHarness doesn't critically depend on a particular worker harness initialization to learn effective evolution strategies.

Sensitivity to maximum tries budgets $K_{\max}$. Finally, we study the effect of the maximum tries budget $K_{\max}$. As shown in Fig. 5(d), increasing $K_{\max}$ from 3 to 5 improves accuracy from 44.0% to 45.3%. Further increasing the budget provides only marginal improvement while increasing the actual number of tries. This indicates that most performance gain is obtained within a small tries budget, while larger budgets mainly increase adaptation cost without substantial benefits. We therefore set $K_{\max} = 5$ throughout our experiments.

5 Concluding Remarks

This paper identifies a reusable structure in harness evolution across tasks. Building on this, our proposed framework, **ReHarness**, learns a reusable evolution strategy across tasks, enabling harness evolution on new tasks from only a few examples and improving agent performance across benchmarks and models.

AI USE STATEMENT

In this work, we used generative AI tools to assist with editing the manuscript to improve readability, clarity, and writing quality. We have not used generative AI tools for generating research ideas, formulating hypotheses, designing the methodology or experiments, implementing methods, analyzing or interpreting results, formulating mathematical claims, or other research tasks requiring disclosure. These tasks were performed by the authors. We have reviewed all AI-assisted writing modifications and verified their correctness. We take responsibility for the final content of this paper, including all text, claims, and artifacts produced with the aid of generative AI.

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, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, Changhao Zhang, Changqing Yu, Chao Wang, Cheng Ma, Cheng Zhong, Cheng Zhu, Chengjun Xiao, Chengyi Yang, Chengyu Du, Chenyang Zhang, Chi Zhang, Chuangyi Huang, Chunhao Zhang, Chunhui Du, Chunyu Zhao, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Dongyu Zhang, Enhui Yang, Fei Yu, Guang Zheng, Guodong Zheng, Guohong Li, Haichao Zhu, Haigang Zhou, Haimo Zhang, Han Ding, Hao Zhang, Haohai Sun, Haolin Lyu, Haonan Lu, Haoyu Wang, Huajie Shi, Huiyang Li, Jiacheng Chen, Jian Zhang, Jiaqi Zhuang, Jiaren Cai, Jiaxin Pan, Jiayao Li, Jiayuan Song, Jichuan Zhang, Jie Wang, Jihao Gu, Jin Zhu, Jingwei Dong, Jingyang Li, Jingyu Zhang, Jingze Zhuang, Jinhao Tian, Jinli Liu, Jinyi Hu, Jun Tao, Jun Zhang, Junbin Ruan, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kang Xu, Ke Ji, Ke Yang, Kecheng Xiao, Keyu Duan, Keyu Li, Le Han, Letian Ruan, Li Yuan, Lianfei Yu, Liheng Feng, Lijie Mo, Lin Li, Linge Du, Lingye Bao, Lingyu Yang, Lingyuan Zhou, Loki, Lu Chen, Lunbin Zeng, Ming Li, Ming Zhong, Mingliang Tao, Mingyuan Chi, Mujie Lin, Nan Hu, Ningxin Chen, Peiyin Zhu, Peng Gao, Pengcheng Gao, Pengfei Li, et al. The minimax-m2 series: Mini activations unleashing max real-world intelligence. *arXiv:2605.26494*, 2026a.

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

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu,

Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv:2606.19348*, 2026.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new qa dataset augmented with context from a search engine. *arXiv:1704.05179*, 2017.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first International Conference on Machine Learning*, 2024.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, 2017.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering. *arXiv:2602.15763*, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco

Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026.

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

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024*, 2024.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 2018.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, et al. Gpt-4 technical report. *arXiv:2303.08774*, 2024.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Josh Joseph, Ivan Zhou, Xinyi Wang, Ashutosh Baheti, Owen Oertell, Sam Havens, Jacob Portes, Michael Bendersky, Erich Elsen, and Matei Zaharia. OfficeQA pro: An enterprise benchmark for end-to-end grounded reasoning. In *Third Conference on Language Modeling*, 2026.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of qwen3.8-next architecture: Evaluation, efficiency, and training stability. *arXiv:2608.30320*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023*, 2023.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations*, 2024.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Trans. Mach. Learn. Res.*, 2024.

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. *Trans. Mach. Learn. Res.*, 2024a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 2024b.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-harness: Context-to-harness skill compilation for self-evolving agents. *arXiv:2608.15071*, 2026.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv:2505.09388*, 2025.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. In *Forty-third International Conference on Machine Learning*, 2026.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv:2606.09498*, 2026a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv:2603.19461*, 2026c.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv:2506.05176*, 2025b.

CONTENTS

A Algorithm of ReHarness . . . 16

B Related Work . . . 17

B.1 LLM Agents and Agent Harnesses . . . 17

B.2 Harness Optimization and Evolution . . . 17

C More Results and Analyses . . . 18

C.1 Analysis of Existing Harness Self-Evolution Methods and Their Applicability . . . 18

C.2 Ablation on the Role of the Learned Adapter Harness . . . 19

C.3 Ablation on the Cross-batch Evolution History . . . 20

C.4 Ablation on Offline-given Task Diversity . . . 22

D Details on Experimental Settings . . . 22

D.1 Details of Cross-Task Trajectory Embeddings Analysis . . . 22

D.2 Details of Evolution Step Analysis for Shared Strategies . . . 23

D.3 Details on Cross-Task Harness Transfer Analysis . . . 23

D.4 Details of Shared and Task-Specific Strategy Distribution . . . 23

D.5 Details on Dataset Settings . . . 23

D.6 Details on Benchmarks . . . 24

D.7 Self-Harness Baseline and Evaluation Settings . . . 25

D.8 Details of Worker Harness Initialization . . . 25

E Case Study . . . 26

E.1 Offline Phase: Learning How to Evolve Across Tasks . . . 26

E.2 Online Phase: Reusing the Learned Strategy on OfficeQA-Pro . . . 27

E.3 What Is Reused and What Remains Task-Specific . . . 28

Algorithm 2 Offline Phase of ReHarness

Require: Given task batches $\{\mathcal{B}_b\}_{b=1}^{B}$, support and query sets $\{(\mathcal{S}_\tau, \mathcal{Q}_\tau)\}$, initial adapter harness a_1^0, fixed optimizer agent $\mathcal{O}$, worker agent $\mathcal{W}$, maximum tries budget $K_{\max}$, number of proposals per batch P.

Initialize optimizer history $\mathcal{H}_{\text{opt}} \leftarrow \emptyset$
for each offline batch $b = 1, \ldots, B$ **do**
  $a_b^0 \leftarrow a_1^0$ **if** $b = 1$; otherwise $a_b^0 \leftarrow a_{b-1}^P$
  for each proposal $p = 1, \ldots, P$ **do**
    $\widetilde{a}_b^p \leftarrow \mathcal{O}(a_b^{p-1}, \mathcal{H}_{\text{opt}})$ ▷ Propose an adapter harness modification
    for each candidate $a \in \{a_b^{p-1}, \widetilde{a}_b^p\}$ **do**
      for each task $\tau \in \mathcal{B}_b$ **do**
        Initialize task-specific worker harness $h_{\tau,a}^0$
        $k \leftarrow 0$
        while $k < K_{\max}$ **do**
          $\mathcal{F}_\tau^k \leftarrow \mathcal{W}(h_{\tau,a}^k; \mathcal{S}_\tau)$ ▷ Execute worker agent on support examples
          $(h_{\tau,a}^{k+1},\ s_\tau^k) \leftarrow a(h_{\tau,a}^k, \mathcal{S}_\tau, \mathcal{F}_\tau^k)$ ▷ Adapter agent modifies worker harness
          $k \leftarrow k + 1$
          if $s_\tau^k = \textsc{Stop}$ **then**
            break
          end if
        end while
        $K_{\tau,a} \leftarrow k$
        $q_\tau(a) \leftarrow J_\tau\big(h_{\tau,a}^{K_{\tau,a}}; \mathcal{Q}_\tau\big)$ ▷ Held-out query evaluation

      end for
      $Q_b(a) \leftarrow \frac{1}{|\mathcal{B}_b|} \sum_{\tau \in \mathcal{B}_b} q_\tau(a)$ ▷ Cross-task batch score
    end for
    if $Q_b(\widetilde{a}_b^p) > Q_b(a_b^{p-1})$ **then**
      $a_b^p \leftarrow \widetilde{a}_b^p$ ▷ Accept improved adapter harness
    else
      $a_b^p \leftarrow a_b^{p-1}$ ▷ Reject non-improving proposal
    end if
    $\Delta_\tau^p \leftarrow q_\tau(\widetilde{a}_b^p) - q_\tau(a_b^{p-1}), \quad \forall \tau \in \mathcal{B}_b$ ▷ Per-task diagnostic feedback
    $\mathcal{H}_{\text{opt}} \leftarrow \text{Summarize}\big(\mathcal{H}_{\text{opt}}, \widetilde{a}_b^p, Q_b(\widetilde{a}_b^p) - Q_b(a_b^{p-1}), \{\Delta_\tau^p\}_{\tau \in \mathcal{B}_b}\big)$ ▷ Retain compact high-signal evolution history
  end for
end for
$a^\star \leftarrow a_B^P$
return learned adapter harness $a^\star$

A ALGORITHM OF REHARNESS

The pseudocode of ReHarness is shown in Algorithm 2 and Algorithm 3. Algorithm 2 describes how the adapter harness is progressively evolved across batches of given tasks, while Algorithm 3 shows how the learned adapter harness is used to rapidly evolve a task-specific worker harness for a new task.

Algorithm 3 Online Phase of ReHarness

Require: New task $\tau^\star$, few-shot support set $\mathcal{S}_{\tau^\star}$, learned adapter harness $a^\star$, worker agent $\mathcal{W}$, maximum tries budget $K_{\max}$.

Initialize task-specific worker harness $h^0_{\tau^\star}$
$k \leftarrow 0$
while $k < K_{\max}$ **do**
  $\mathcal{F}^k_{\tau^\star} \leftarrow \mathcal{W}(h^k_{\tau^\star}; \mathcal{S}_{\tau^\star})$ ▷ Execute worker agent on a few examples
  $(h^{k+1}_{\tau^\star},\ s^k_{\tau^\star}) \leftarrow a^\star(h^k_{\tau^\star}, \mathcal{S}_{\tau^\star}, \mathcal{F}^k_{\tau^\star})$ ▷ Evolve task-specific worker harness
  $k \leftarrow k + 1$
  if $s^k_{\tau^\star} = \textsc{Stop}$ **then**
    break
  end if
end while
$K_{\tau^\star} \leftarrow k$
$h^\star_{\tau^\star} \leftarrow h^{K_{\tau^\star}}_{\tau^\star}$
return $h^\star_{\tau^\star}$ ▷

B Related Work

B.1 LLM Agents and Agent Harnesses

LLM agents. Recent advances in large language models (LLMs) (Grattafiori et al., 2024; OpenAI et al., 2024; DeepSeek-AI et al., 2026; GLM-5-Team et al., 2026; Yang et al., 2025; Chen et al., 2026a) have substantially improved their capabilities in language understanding, reasoning, and generation. However, solving complex real-world tasks often requires capabilities beyond language generation, such as interacting with external environments, invoking tools, maintaining memory, and executing actions. LLM agents (Wang et al., 2024b; Xi et al., 2025) address this limitation by augmenting LLMs with these external capabilities, enabling iterative reasoning and interaction with the environment. ReAct (Yao et al., 2023) interleaves reasoning traces with environment actions, while Toolformer (Schick et al., 2023) enables language models to invoke external tools through learned API calls. Reflexion (Shinn et al., 2023) incorporates verbal feedback and episodic memory to improve subsequent decisions, and Generative Agents (Park et al., 2023) maintain memory and reflection for persistent interaction. Voyager (Wang et al., 2024a) further builds an executable skill library from previous experience, allowing acquired skills to be reused on new tasks. CoALA (Sumers et al., 2024) organizes language agents around modular memory, structured action spaces, and decision-making processes.

Agent harnesses. Beyond the underlying model, agent behavior is also shaped by the surrounding system that controls instructions, tools, memory, state, and execution. SWE-agent (Yang et al., 2024) shows that agent interface design can substantially affect performance under a fixed language model. Recent work formalizes these surrounding components as an agent harness (Lee et al., 2026; Lin et al., 2026; Zhang et al., 2026a), providing a programmable layer for improving agent capabilities without modifying model parameters.

B.2 Harness Optimization and Evolution

Recent works increasingly automate the design of the programmable components surrounding LLM agents, ranging from prompts and workflows to complete agent harnesses.

Prompt and workflow optimization. Promptbreeder (Fernando et al., 2024) evolves both task prompts and the mutation prompts used to improve them, while DSPy (Khattab et al., 2024) automatically optimizes prompts and demonstrations in modular LLM pipelines. GEPA (Agrawal et al., 2026) further uses execution trajectories and natural-language reflection to evolve prompts from trial-and-error experience. Beyond prompts, AFlow (Zhang et al., 2025a) searches over code-represented agent workflows using execution feedback, while Automated Design of Agentic Systems (Hu et al., 2025) employs a meta agent to iteratively discover improved agent architectures. Darwin Gödel Machine (Zhang et al., 2026b) extends this direction by allowing agents to modify and evaluate their own implementations.

Harness evolution. Recent approaches directly optimize the complete agent harness. Meta-Harness (Lee et al., 2026) evolves task-specific harnesses using previous implementations, execution traces, and evaluation scores. Self-Harness (Zhang et al., 2026a) improves an agent's operating harness through weakness identification, harness proposal, and regression-based validation. Agentic Harness Engineering (Lin et al., 2026) similarly uses execution evidence to guide iterative harness improvement. Evo-Harness (Wei et al., 2026) instead continually updates a structured harness online by compiling single-shot executions into reusable skills accumulated across sequential tasks.

Reusable Evolution Strategies. Several recent studies further explore whether the improvement process itself can be learned or reused. Meta Context Engineering (Ye et al., 2026) jointly evolves context-engineering skills and the contexts produced by these skills. Hyperagents (Zhang et al., 2026c) make meta-level improvement procedures editable, allowing improvements to accumulate and transfer across domains. Different from them, our work focuses specifically on learning a reusable harness evolution strategy across heterogeneous tasks, which is reused to evolve task-specific worker harnesses for new tasks from only a few examples.

C More Results and Analyses

C.1 Analysis of Existing Harness Self-Evolution Methods and Their Applicability

Existing harness self-evolution methods are designed for target-task harness optimization, where the harness is repeatedly evolved and evaluated using feedback from the same task distribution. In contrast, ReHarness focuses on learning a reusable evolution strategy from given tasks and transferring it to new tasks, where only a few support examples are available for task-level adaptation. As illustrated in Fig. 6, existing methods optimize each task-specific harness from scratch, whereas ReHarness learns how to evolve across tasks. Therefore, a direct comparison on new tasks would conflate two different capabilities: target-task harness optimization and cross-task transfer of an evolution strategy.

Meta-Harness (Lee et al., 2026) performs outer-loop search over task-specific harnesses using evaluation scores and execution traces from the target search setting. However, directly adapting Meta-Harness to our setting faces several challenges. First, Meta-Harness relies on Claude Code with Opus 4.6 as an external optimizer to propose harness modifications, whereas ReHarness learns the evolution strategy using the same underlying model agents. Replacing this external optimizer would require redesigning its original optimization pipeline. Second, Meta-Harness uses benchmark-specific SKILL.md specifications to guide harness optimization for each evaluated benchmark. Since these task-specific specifications are manually designed for the original benchmarks and are unavailable for our evaluation tasks, creating new SKILL.md files would introduce additional benchmark-specific supervision and make the comparison dependent on extra human effort. Third, directly comparing on new tasks with target-task evolution would give Meta-Harness access to additional task-specific optimization data, conflicting with our transfer setting. Although one possible solution is to compare both methods on offline tasks under matched evolution budgets, the above external optimizer and benchmark-specific skill requirements prevent a faithful reproduction of Meta-Harness under our setting.

Self-Harness (Zhang et al., 2026a) provides a more compatible comparison because it uses the same underlying model to improve its own harness without requiring an external optimizer or benchmark-specific skill specification. It mines failure patterns from execution traces, proposes candidate harness modifications, and evaluates candidates on held-in and held-out examples. However, similar to Meta-Harness (Lee et al., 2026), Self-Harness directly optimizes the harness on the target task and therefore cannot be directly compared with ReHarness in the unseen-task transfer setting (Table 1). To enable a controlled comparison, we instead evaluate both methods on given tasks from our offline suite, where Self-Harness and ReHarness perform task-level harness evolution under the same data budget. Specifically, we match the held-in and held-out examples used for evolution and introduce an additional disjoint test set for final evaluation (Table 2), as detailed in Appendix D.7.

AHE (Lin et al., 2026) evolves a coding-agent harness through repeated execution and modification on Terminal-Bench 2. Although its cross-benchmark experiment transfers the resulting harness unchanged to SWE-bench Verified without further evolution, both benchmarks remain within the coding domain. Therefore, similar to other task-specific evolution methods, AHE optimizes a harness for a specific task distribution rather than learning a reusable evolution strategy across heterogeneous tasks. Moreover, AHE is built on the NexAU coding-agent harness framework, which exposes coding-specific components such as tools, middleware, and memory for evolution. In contrast, ReHarness evolves task-specific worker harnesses represented as a single Python file across diverse task types. Thus, directly comparing AHE with ReHarness would involve differences in both task domains and harness substrates.

These comparisons highlight the distinction between task-specific harness optimization and reusable evolution strategy learning. Existing methods are effective at improving harnesses when task-specific optimization feedback is available, whereas ReHarness aims to learn the optimization process itself and transfer it across tasks. Therefore, the Self-Harness comparison under matched task-level evolution evaluates the benefit of learning a reusable evolution strategy over directly optimizing each task independently, while the new task evaluation measures the cross-task transfer capability that existing methods are not designed to address.

C.2 Ablation on the Role of the Learned Adapter Harness

To understand the role of the learned adapter harness in ReHarness, we compare it with two ablation settings that examine whether the gains come from (1) the five support examples alone without harness evolution, or (2) iterative harness adaptation with an adapter harness that has not been optimized across tasks. Both settings use the same task-specific examples as ReHarness, while removing the evolution strategy learned through cross-task optimization.

Table 3 reports the results on the four tasks using DeepSeek-V4-Flash. In all settings, we use the same five support examples provided to ReHarness and the same worker harness initialization. For Baseline + 5-shot, the examples are directly provided in the task context without any harness evolution. For Initial Adapter Harness, we use the initial adapter harness a_1^0 before cross-task optimization and perform iterative worker-harness adaptation under the same maximum tries budget $K_{\max}$.

We first examine whether the five support examples alone can account for the improvements of ReHarness. Compared with the baseline, Baseline + 5-shot improves performance on GAIA and OfficeQA-Pro, but slightly decreases performance on τ^2-Bench and SWE-bench Verified, resulting in nearly unchanged average performance. This indicates that simply providing additional task-specific examples is insufficient to explain the gains achieved by ReHarness.

We next investigate whether iterative harness adaptation can be effective without a cross-task optimized adapter harness. Although Initial Adapter Harness follows the same iterative adaptation procedure and uses the same five support examples as ReHarness, it doesn't consistently improve over the baseline. While it improves GAIA from 28.7 to 30.3, it decreases performance on the other three tasks and reduces the average

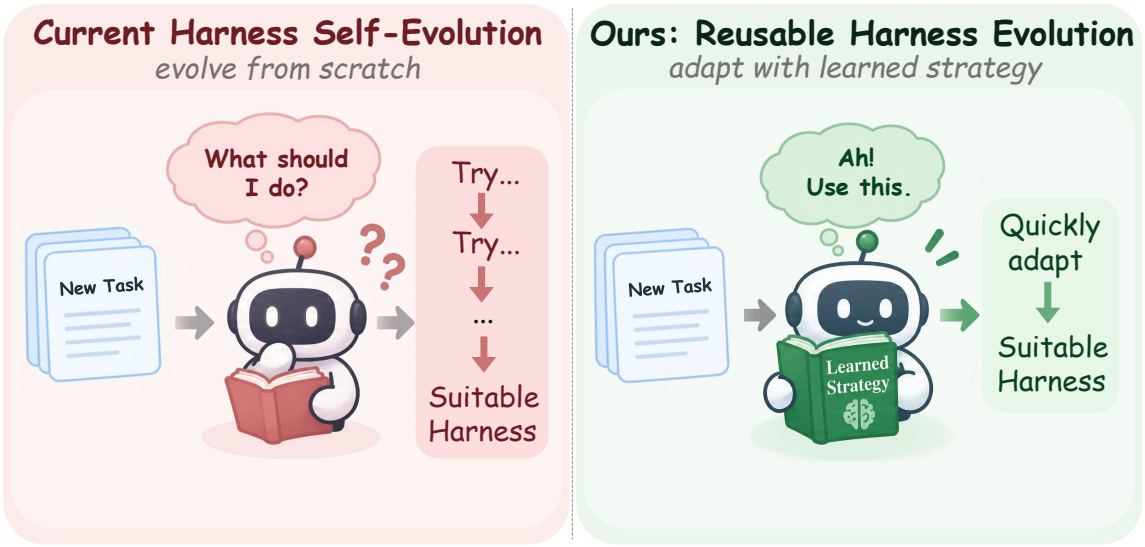


Figure 6: **Comparison between task-specific harness self-evolution and reusable harness evolution.** Existing self-evolution methods optimize each task-specific harness from scratch using task-specific feedback, while ReHarness learns a reusable evolution strategy from given tasks and transfers it to new tasks for rapid adaptation with limited examples.

score from 42.9 to 41.5. This suggests that iterative modification of the worker harness alone is insufficient, and that effective adaptation requires an adapter harness optimized across the given tasks.

In contrast, ReHarness consistently outperforms Initial Adapter Harness across all four tasks. Since both settings share the same support examples, worker harness initialization, adaptation budget, and iterative adaptation procedure, these results demonstrate that the cross-task optimization of the adapter harness is crucial for enabling effective harness evolution on new tasks.

Together, these results show that the improvements of ReHarness cannot be explained by simply providing more examples or applying iterative adaptation with an unoptimized adapter. Instead, the adapter harness learned through cross-task optimization enables more reliable and effective evolution of the worker harness across diverse unseen tasks. This supports the design principle of ReHarness that the strategy for evolving harnesses can be learned and reused across tasks.

C.3 Ablation on the Cross-batch Evolution History

Effect of cross-batch evolution history. We further investigate the role of the compact cross-batch evolution history H_{opt} using DeepSeek-V4-Flash. ReHarness carries information across offline batches in two ways: the selected adapter harness is inherited by the next batch, $a_{b+1}^0 = a_b^P$, while H_{opt} summarizes previous adapter proposals, their observed improvements or regressions, and recurring failure patterns. To isolate the contribution of the latter, we remove H_{opt} from the optimizer agent while retaining cross-batch inheritance of the selected adapter harness. Thus, the adapter harness can still preserve evolution strategies learned through accepted updates, but the optimizer agent cannot explicitly reuse experience from previous proposals. As shown in Table 4, removing the evolution history reduces the average performance on the four unseen tasks from 49.2% to 46.2%. Nevertheless, ReHarness without history summary still consistently improves over the

Table 3: **Ablation on the role of the learned adapter harness in ReHarness.** We compare ReHarness with two controls that remove the cross-task optimization of the adapter harness while preserving the task-specific information or adaptation procedure. *Baseline + 5-shot* uses the same five support examples as ReHarness without performing harness evolution. *Initial Adapter Harness* replaces the optimized adapter harness $a^\star$ with the initial adapter harness a_1^0 and follows the same iterative adaptation procedure and maximum adaptation budget $K_{\max}$ as ReHarness. Results are reported as mean $\pm$ standard deviation over three independent runs.

Method	GAIA	τ^2-Bench	OfficeQA-Pro	SWE-bench Verified	Avg.
Baseline	28.7 $\pm$ 0.8	79.5 $\pm$ 1.1	36.7 $\pm$ 2.1	26.7 $\pm$ 1.0	42.9
Baseline + 5-shot	29.8 $\pm$ 1.2	78.6 $\pm$ 1.9	39.1 $\pm$ 1.3	24.3 $\pm$ 2.1	43.0
Initial Adapter Harness	30.3 $\pm$ 1.4	74.0 $\pm$ 1.0	35.7 $\pm$ 2.0	26.1 $\pm$ 1.0	41.5
ReHarness (ours)	**33.6** $\pm$ 1.6	**84.5** $\pm$ 2.8	**45.3** $\pm$ 2.1	**33.3** $\pm$ 1.0	**49.2**

Table 4: **Ablation on the cross-batch evolution history using DeepSeek-V4-Flash.** We compare ReHarness with and without H_{opt} on four new tasks without using in the offline phase. Results are mean $\pm$ standard deviation over three independent runs.

Method	GAIA	τ^2-Bench	OfficeQA-Pro	SWE-bench Verified	Avg.
Baseline	28.7 $\pm$ 0.8	79.5 $\pm$ 1.1	36.7 $\pm$ 2.1	26.7 $\pm$ 1.0	42.9
ReHarness w/o History	32.2 $\pm$ 1.3	80.7 $\pm$ 1.3	42.7 $\pm$ 1.0	29.1 $\pm$ 1.8	46.2
ReHarness	33.6 $\pm$ 1.6	84.5 $\pm$ 2.8	45.3 $\pm$ 2.1	33.3 $\pm$ 1.0	49.2

baseline on all four tasks, increasing the average score from 42.9% to 46.2%. This shows that the cross-batch evolution of the adapter harness remains effective even without explicit optimization history. These results indicate that H_{opt} mainly improves the efficiency of adapter harness optimization. By retaining the outcomes of previous successful and unsuccessful proposals, it helps the optimizer agent avoid repeatedly exploring ineffective modification directions and more efficiently discover useful adapter updates.

Evolution history improves optimization efficiency. We further inspect the adapter harness optimization trajectories to understand why H_{opt} improves downstream performance. Without the history, only 4 adapter updates are accepted throughout the offline phase, compared with 9 under the full ReHarness configuration. Moreover, many unsuccessful proposals in the ReHarness without history revisit failure modes that had already appeared in earlier batches. This occurs because the selected adapter harness preserves the modifications that have been inherited, but doesn't explicitly record why previous candidate modifications were rejected.

In contrast, H_{opt} retains compact information about previous proposal mechanisms and their observed improvements or regressions. This allows the optimizer agent to avoid repeating similar unsuccessful modifications and focus subsequent proposals on evolution directions that have been effective across tasks. As a result, the history increases both the number of accepted adapter updates ($4 \rightarrow 9$) and final generalization performance ($46.2\% \rightarrow 49.2\%$ on average across new tasks). These results show that H_{opt} primarily improves the efficiency of discovering transferable evolution strategies rather than being required for learning them.

Table 5: **Ablation on offline-given task diversity using DeepSeek-V4-Flash.** We increase the number of heterogeneous given tasks from two to four progressively and evaluate the performance on the four new tasks, while all online settings remain unchanged. Results are the mean ± standard deviation over three independent runs. Best performances are bolded.

Given Tasks	GAIA	τ^2-Bench	OfficeQA-Pro	SWE-bench Verified	Avg.
2 tasks	29.2 ± 1.7	82.8 ± 0.7	39.2 ± 0.8	27.9 ± 1.7	44.8
3 tasks	**34.2** ± 1.7	80.5 ± 0.6	42.7 ± 2.0	30.9 ± 1.8	47.1
4 tasks (ReHarness)	33.6 ± 1.6	**84.5** ± 2.8	**45.3** ± 2.1	**33.3** ± 1.0	**49.2**

C.4 Ablation on Offline-given Task Diversity

A hypothesis of ReHarness is that reusable evolution strategies can be learned from evolution experience shared across heterogeneous tasks. We therefore investigate how the diversity of offline-given tasks affects generalization to new tasks. We construct variants of ReHarness on DeepSeek-V4-Flash, using two, three, and all four given tasks while keeping the remaining offline and online settings unchanged. We start with two given tasks because our goal is to study cross-task learning. With only one task, the outer objective would no longer capture cross-task variation and would instead optimize for task-specific evolution behavior. As shown in Table 5, increasing offline-given task diversity leads to higher average performance on new tasks. Optimization on two tasks achieves an average score of 44.8%, which increases to 47.1% with three tasks and 49.2% with all four tasks. The improvement is observed across all evaluation benchmarks rather than being driven by a single task. Compared with the two-task one, optimization on all four tasks improves GAIA from 29.2% to 33.6%, τ^2-Bench from 82.8% to 84.5%, OfficeQA-Pro from 39.2% to 45.3%, and SWE-bench Verified from 27.9% to 33.3%.

The improvement in average performance is monotonic with respect to offline-given task diversity, although individual benchmarks exhibit small non-monotonic variations. This demonstrates that adding an offline task doesn't guarantee improvement on every individual benchmark, but improves overall generalization across tasks. More heterogeneous task feedback exposes the optimizer agent to a broader range of evolution patterns. This increases the chance of identifying strategies that remain useful across different task distributions. This suggests that diverse offline-given tasks provide the cross-task evidence needed to identify more reusable evolution strategies, further demonstrating ReHarness's ability to generalize to new tasks.

D Details on Experimental Settings

D.1 Details of Cross-Task Trajectory Embeddings Analysis

To analyze whether existing task-specific harness evolution exhibits similar evolution patterns across different tasks, we collect evolution trajectories produced by Self-Harness (Zhang et al., 2026a) on four tasks: BFCL V4 (Patil et al., 2025), SpreadsheetBench (Ma et al., 2024), SearchQA (Dunn et al., 2017), and MuSR (Sprague et al., 2024). We use the same evolution configuration for all four tasks. For each task, we randomly sample 200 instances and partition them into five disjoint subsets of 40 instances. Within each subset, 20 instances are used as held-in examples and the remaining 20 as held-out examples. Beginning with the initial harness, we run Self-Harness for 10 evolution steps on each subset. We record the harness modification produced at every evolution step, resulting in $5 \times 10 = 50$ step-level evolution harnesses per task.

To compare these harnesses at the semantic level rather than through their raw implementation details, we construct a trajectory abstraction. Specifically, after each evolution step, we prompt an LLM to summarize

the intent of the resulting harness modification, focusing only on what behavior the modification is intended to improve. Each abstracted trajectory is then encoded using Qwen3-Embedding-8B (Zhang et al., 2025b), yielding a vector representation in which semantically similar evolution intents are expected to be located closer in the embedding space.

Finally, we apply UMAP (McInnes et al., 2018) to project the high-dimensional trajectory embeddings into two dimensions for visualization. The resulting representation, shown in Fig. 2(a), allows us to examine whether evolution trajectories from different benchmarks form task-specific clusters or exhibit shared semantic structures across tasks.

D.2 Details of Evolution Step Analysis for Shared Strategies

Following the evolution setup in Appendix D.1, we further analyze when semantically shared and beneficial evolution strategies emerge during independently task-specific harness evolution. For each task, we first identify evolution steps that yield a positive improvement on the held-out set relative to the harness before the corresponding modification. We then associate each such beneficial modification with its abstracted evolution intent and embedding obtained using the procedure described in Appendix D.1.

We compare these beneficial evolution intents across BFCL V4, MuSR, SpreadsheetBench, and SearchQA. Evolution intents that are semantically similar in the embedding space and repeatedly correspond to held-out improvements across different benchmarks are grouped into a shared evolution strategy. We select four representative strategies that emerge independently across heterogeneous tasks.

To characterize when these shared strategies are discovered through task-specific evolution, we record the first discovery step of each strategy in each independent evolution run. Specifically, for a given strategy, we identify the earliest evolution step whose beneficial modification is assigned to that strategy. We then aggregate the first discovery steps across independent runs for each benchmark. Fig. 2(b) reports the resulting average first discovery step for each shared strategy and task.

D.3 Details on Cross-Task Harness Transfer Analysis

For each task, we randomly sample 100 instances, with 50 held-in and 50 held-out examples, and evolve the harness for 10 steps using Self-Harness. We report the overall performance on all 100 instances. To evaluate cross-task transfer, we then transfer the evolved harness to each of the other benchmarks on the target task. We evaluate the resulting harness on the same 100 target instances and report its performance change relative to the initial harness. As shown in Fig. 3(a), this analysis measures whether improvements obtained through task-specific harness evolution can be effectively reused on different tasks.

D.4 Details of Shared and Task-Specific Strategy Distribution

Following the evolution setup in Fig. 2, we use the same procedure as in Fig. 2(b) to identify shared evolution strategies across tasks. Beneficial evolution strategies that exhibit semantic similarity across different tasks are categorized as shared strategies, while the remaining strategies are treated as task-specific strategies. We then calculate the proportion of these two categories among the beneficial strategies discovered during harness evolution for each task. The resulting distributions are reported in Fig. 3(b).

D.5 Details on Dataset Settings

Dataset configurations during the offline phase. The offline given set consists of 4 types of tasks organized into 10 batches. For each batch, we use 5 support examples and 15 held-out query examples. Specifically, we optimize the adapter harness on four heterogeneous tasks covering different agent capabilities: SearchQA for

retrieval-based question answering, SpreadsheetBench for executable spreadsheet manipulation, MuSR for long-context multi-step reasoning, and BFCL V4 for function calling and tool use. This diverse benchmark suite evaluates whether the learned evolution strategy improves general worker agent behavior across task types rather than optimizing for a single task format.

Dataset configurations during the online phase. As described in Sec. 4, we evaluate ReHarness on four benchmarks: GAIA, OfficeQA-Pro, τ^2-Bench, and SWE-bench Verified. For each benchmark, we directly apply the learned final adapter harness to evaluate our method without any additional offline optimization.

For **GAIA** (Mialon et al., 2024), we follow the official GAIA task format and evaluate on the text-only subset of the validation split, as ground-truth answers for the official test split are not publicly available. Tasks requiring external attachments are excluded. The resulting evaluation set contains 127 tasks, including 5 support examples exposed to the adapter agent and 122 held-out query examples used to evaluate both baseline and our method.

For **OfficeQA-Pro** (Opsahl-Ong et al., 2026), we restrict evaluation to the 133 hard instances, with a fixed split of 5 support examples and 128 query examples. We use the official OfficeQA-Pro reward implementation with the numerical tolerance set to zero, requiring predictions to satisfy the benchmark's exact answer-matching and normalization criteria.

For **τ^2-Bench** (Barres et al., 2026), we use the official simulation environment and evaluate across three domains: airline, retail, and telecom. The evaluation follows the official multi-turn protocol, such that each episode covers the complete interaction trajectory from user simulation and tool execution to state transitions, termination, and task scoring. Across the three domains, we use 5 support examples and 273 query examples.

For **SWE-bench Verified** (Jimenez et al., 2024), we follow the official SWE-bench Verified dataset and its canonical evaluation metadata. The SWE-bench evaluation setting follows that of HarnessX (Chen et al., 2026b), including the query evaluation protocol. Each run uses 5 support examples and a sampled set of query examples from the benchmark.

D.6 Details on Benchmarks

BFCL (Patil et al., 2025): The Berkeley Function Calling Leaderboard (BFCL) is a comprehensive benchmark for evaluating function-calling capabilities, covering diverse real-world tool-use scenarios including parallel calls and multi-turn, stateful interactions.

MuSR (Sprague et al., 2024): A multistep soft reasoning benchmark built around natural-language narratives, designed to test robust reasoning over complex scenarios that require combining multiple pieces of implicit and explicit evidence.

SpreadsheetBench (Ma et al., 2024): A real-world spreadsheet manipulation benchmark consisting of complex user instructions and associated spreadsheet files, evaluating agents' ability to understand, reason over, and modify spreadsheets through executable operations.

SearchQA (Dunn et al., 2017): A search-based question answering benchmark that evaluates agents' ability to iteratively retrieve relevant information and synthesize evidence through search and reasoning.

GAIA (Mialon et al., 2024): A benchmark for general AI assistants consisting of real-world questions that require a combination of reasoning, web browsing and tool use across multiple levels of difficulty.

OfficeQA-Pro (Opsahl-Ong et al., 2026): An enterprise-oriented benchmark for grounded reasoning over large document collections, requiring agents to perform precise retrieval, document parsing, and analytical reasoning across both textual and tabular information.

τ^2-Bench (Barres et al., 2026): A conversational agent benchmark featuring dual-control environments in which both the agent and user can take actions, testing agents' ability to reason, coordinate, and use tools in realistic service interactions.

SWE-bench Verified (Jimenez et al., 2024): A human-validated subset of SWE-bench containing 500 real-world software engineering tasks derived from GitHub issues, evaluating agents' ability to modify codebases and resolve repository-level problems.

D.7 Self-Harness Baseline and Evaluation Settings

Self-Harness (Zhang et al., 2026a) improves the harness of an agent through an iterative proposal and validation procedure. Starting from an initial harness, it first identifies failure patterns from execution traces on a set of held-in tasks, proposes candidate harness modifications based on the observed weaknesses, and then validates these modifications through regression testing. In its original evaluation setting, the tasks are divided into held-in and held-out sets. Held-in execution traces are used to drive harness evolution, while held-out traces are hidden from the proposer and their scores serve as a regression gate for accepting or rejecting candidate modifications. The final harness is then evaluated and reported on both splits. Consequently, the held-out set participates in harness selection, and no additional test split is used for final evaluation.

Unlike ReHarness, Self-Harness requires target-specific evolution using execution feedback from the target task distribution. It therefore doesn't directly support our setting in which a learned adapter harness is transferred to previously unseen benchmarks. For this reason, we don't include Self-Harness in the cross-benchmark generalization comparison in Table 1. Instead, we compare the two methods under a matched evolution setting with the same data budgets.

To ensure a fair comparison, we evaluate Self-Harness under the same data and evolution budget as ReHarness. Specifically, for each target benchmark, we run 10 evolution batches. In each batch, Self-Harness uses 5 held-in support examples to identify failure patterns and propose three candidate harness modifications. Each candidate is then evaluated on the same 15 held-out query examples, and the resulting held-out performance is used to validate the proposal following the original Self-Harness protocol. Across the 10 batches, this gives Self-Harness access to 50 held-in examples and 150 held-out examples in total, matching the examples used in the corresponding ReHarness evolution setting.

Importantly, to evaluate the generalization of the resulting harness rather than its performance on examples involved in evolution or selection, we introduce an additional disjoint test split for final evaluation. After evolution, the final harness is evaluated with 5 support examples and 100 previously unseen test examples. The 100 test examples remain completely hidden during evolution and model selection. This additional test split provides an independent estimate of harness generalization and enables an fair comparison between Self-Harness and ReHarness under the same data and evaluation settings.

D.8 Details of Worker Harness Initialization

Our default worker harness is manually designed to be simple and general, such that it can serve as a common initialization across different tasks. To construct alternative initializations for the sensitivity analysis in Sec. 4.2, we provide this default worker harness to an LLM as a template and ask it to generate three alternative worker harnesses with different design emphases. We explicitly instruct the LLM to keep each alternative simple and general. The resulting three worker harnesses are used as alternative initializations, while all other settings remain unchanged. The prompt used for generation is shown below.

Table 6: **A representative adapter harness update during cross-task optimization in batch 5.** The selected update improves held-out query performance across several heterogeneous tasks rather than benefiting only a single benchmark.

Task	Parent Adapter	Selected Adapter	Δ
BFCL V4	0.733	0.800	+0.067
MuSR	0.467	0.733	+0.267
SearchQA	0.956	1.000	+0.044
SpreadsheetBench	0.267	0.267	+0.000
Average	0.5185	0.7000	+0.1815

Prompt for Alternative Worker Harness Initialization

Given the following worker harness, which is designed to be simple and general enough to apply across different tasks, generate three alternative worker harnesses with different design emphases. Make the alternatives substantially different from the provided harness and from one another, rather than making only minor modifications. Each alternative should remain simple, general, and applicable across tasks, rather than being specialized to a particular task.

E Case Study

We provide a case study to illustrate what ReHarness learns during cross-task offline optimization and how the learned evolution strategy is used when adapting a worker harness to a new task. The case study highlights the design principle of ReHarness: how to evolve can be learned and reused across tasks, whereas what is evolved remains task-specific. Specifically, we first inspect the evolution of the adapter harness offline, and then trace how the learned adapter harness adapts a worker harness to the new OfficeQA-Pro benchmark online.

E.1 Offline Phase: Learning How to Evolve Across Tasks

We first inspect a representative offline run with DeepSeek-V4-Flash. The adapter harness is evolved across four given heterogeneous tasks: BFCL V4, MuSR, SpreadsheetBench, and SearchQA. For each task episode, the adapter agent observes five support examples and iteratively modifies the task-specific worker harness, while performance on 15 held-out query examples is used to evaluate the resulting adaptation. The optimizer agent then updates the adapter harness according to its average query performance across tasks.

From failure feedback to targeted worker adaptation. Early in the offline phase, worker harness modifications are often ineffective. Some edits leave the support behavior unchanged, while others primarily modify general instructions without reliably transferring to held-out queries. As it proceeds, the adapter harness develops a more structured use of execution feedback. In particular, observed failures are mapped to candidate worker capabilities and corresponding edit surfaces, such as tool handling, answer extraction, context construction, and execution control. The adapter harness additionally retains information about previous adaptation attempts, allowing subsequent steps to avoid repeatedly modifying edit surfaces that have produced little behavioral change.

Table 7: **Online worker harness evolution on OfficeQA-Pro.** The learned adapter harness remains fixed throughout the process and generates task-specific worker modifications from the five support examples. A support-specific proposal is rejected, while the subsequent five proposals are accepted to form the final worker harness.

Proposal	Worker harness modification	Decision
1	Initial Treasury-data grounding proposal	Reject
2	Extract value from `FINAL_ANSWER`	Accept
3	Add data grounding and numeric-output constraints	Accept
4	Normalize scalar and list answers	Accept
5	Add arithmetic, rounding, sign, and formatting guidance	Accept
6	Add question-dependent calculation strategies	Accept

This evolution changes the role of the adapter from producing generic worker modifications to deciding where and how the worker harness should be modified given the observed task feedback. Importantly, the resulting modification remains task-specific: the adapter agent doesn't attempt to reuse a worker harness evolved for another task, but instead reuses the strategy for constructing an appropriate modification.

Cross-task feedback favors reusable adaptation behavior. A representative transition occurs in offline batch 5. The parent adapter harness obtains an average held-out query score of 0.5185, whereas the selected adapter reaches 0.7000 after two accepted updates. The improvement is distributed across multiple heterogeneous tasks: BFCL V4 improves from 0.733 to 0.800, MuSR from 0.467 to 0.733, and SearchQA from 0.956 to 1.000, while SpreadsheetBench remains unchanged at 0.267. The corresponding results are summarized in Table 6. This example illustrates the role of the cross-task objective in Eq. 7. An adapter modification is evaluated through the task-specific worker harnesses it induces across different tasks. Consequently, the optimizer agent receives evidence about whether a proposed evolution behavior is useful beyond a single task.

Selection prevents harmful evolution. The evolution process doesn't require every proposed adapter modification to be inherited. This is important because a plausible modification can improve behavior on the small support set while failing to transfer to held-out queries. For example, in offline batch 9, the current adapter obtains an average query score of 0.5264. The three optimizer proposals produce scores of 0.3417, 0.4978, and 0.4322, respectively. Since none improves upon the parent, all three proposals are rejected and the current adapter harness is retained.

This example shows the role of held-out query feedback. Support examples provide the execution evidence needed to construct worker-harness modifications, whereas query performance determines whether the corresponding adapter harness's behavior should be inherited across offline batches. The resulting evolution is therefore not necessarily monotonic across different batches, but harmful adapter updates need not propagate to later offline stages.

E.2 Online Phase: Reusing the Learned Strategy on OfficeQA-Pro

We next examine how the learned adapter harness is used on a new task. We use a representative DeepSeek-V4-Flash run on OfficeQA-Pro, which is not used during adapter harness offline optimization. Following the main experimental protocol, the learned adapter harness $a^\star$ is fixed and receives only five support examples from OfficeQA-Pro. It then evolves a task-specific worker harness without any further outer-loop optimization.

OfficeQA-Pro requires grounded reasoning over document collections. In the examined run, the five support examples expose recurring failures involving answer boundaries, numerical formatting, and multi-step

calculations. Starting from the initial worker harness, the fixed adapter produces a sequence of task-specific modifications that address these observed failures.

Task-specific worker harness evolution. The accepted modifications progressively introduce four types of behavior. First, the worker harness is instructed to ground its answer in the supplied Treasury Bulletin data while preserving the benchmark's retrieval interface. Second, the worker harness's answer extraction is modified to isolate the value inside the designated `FINAL_ANSWER` boundary, preventing auxiliary reasoning text from being submitted to the evaluator. Third, the worker harness normalizes common numerical answer forms, including thousands separators, signs, whitespace, and list formatting. Finally, the evolved worker harness introduces question-dependent calculation guidance for operations such as absolute differences, ratios, averages, and multi-step numerical computations.

These modifications are specific to the failures exposed by the OfficeQA-Pro support examples. They are not stored as an OfficeQA-specific strategy during cross-task offline optimization. Instead, they are generated online by applying the learned evolution strategy to the current task feedback and worker implementation. Table 7 summarizes this evolution process. The first proposal is rejected because it contains support-specific content, whereas the subsequent proposals progressively refine the worker harness through task-specific but reusable forms of adaptation, including answer extraction, output normalization, and calculation guidance.

Improvement on held-out queries. We evaluate the initial and evolved worker harnesses using the same fixed support and query split. The direct worker solves 44 of 128 held-out queries (34.38%), whereas the evolved worker harness solves 55 of 128 (42.97%), corresponding to an improvement of 11 queries, or 8.59 percentage points. Both conditions use the same five support examples and the same 128 held-out queries, demonstrating effectiveness.

Notably, the recorded support scores in this run are not monotonically improved by the accepted edits. This suggests that cross-task offline optimization teaches the adapter agent to go beyond simply optimizing performance on the few support examples. Instead, it learns to treat support executions as diagnostic evidence, using observed failures to construct task-specific worker modifications that can generalize beyond the examples to avoid simply overfitting to them.

E.3 What Is Reused and What Remains Task-Specific

The two cases together illustrate the separation underlying ReHarness. During the offline phase, cross-task optimization shapes the adapter harness toward a reusable procedure for interpreting execution feedback, selecting useful worker edit surfaces, and avoiding modifications that fail to transfer. During the online phase, this procedure is reused without further optimization, but the resulting worker modifications are determined by the failures of the new task.

For OfficeQA-Pro, this produces task-specific mechanisms for numerical document question answering, including answer extraction, numerical normalization, and calculation guidance. These mechanisms need not resemble the worker harnesses evolved for BFCL V4, MuSR, SpreadsheetBench, or SearchQA. What transfers is therefore not a fixed downstream harness, but the strategy used to construct and refine one from limited task-specific feedback.

This case study provides a concrete instance of the observation in Sec. 2. How to evolve can exhibit reusable structure across heterogeneous tasks, while what is evolved should remain specialized to the downstream task. ReHarness operates on this distinction by learning the former during the cross-task offline phase and applying it to rapidly construct the latter online.