LEARNING FROM SINGLE-FEEDBACK DEPLOYMENT EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

The success of LLM agents has attracted growing interest in deploying small-size, self-hosted models to serve as user assistants for everyday work. After deployment, users would like these assistants to improve as they work. In practice, however, tasks arrive sequentially, and users cannot be expected to repeatedly evaluate different attempts at the same task. A common approach is to adapt memories or skills in the agent harness while keeping model parameters fixed. Parameter-based learning provides another promising direction, yet how to effectively train models from such single-feedback deployment experience remains under-explored. In this paper, we study this problem and introduce DEAL, an executable testbed with related task streams in general workplace assistance and AI research. Using DEAL, we find that single-rollout reinforcement learning suffers from an exploration bottleneck, where one evaluated trajectory may leave better behaviors unexplored. Based on this observation, we propose Self-Rewarded Exploration (SRE), which expands additional trajectories from isolated task snapshots and uses the live feedback as an anchor to assign self-rewards. We further provide a theoretical analysis of when the benefit of rollout expansion can outweigh self-reward errors. Experiments on Gemma4-E4B-IT demonstrate the effectiveness of SRE in improving deployment-time performance, generalizing to unseen tasks, and preserving general capabilities.

1 INTRODUCTION

The success of Large Language Model (LLM) agents has attracted growing interest in deploying small, self-hosted models as user assistants for everyday work (Belcak et al., 2025; MiniCPM Team, 2025). Through an agent harness, these assistants interact with files and tools to analyze data, modify code, and prepare reports (Steinberger et al., 2026). After deployment, users may also expect them to improve as they work (Ren et al., 2026). Task outcomes and user feedback provide useful experience about recurring mistakes or user requirements (Wang et al., 2026b; Zhang et al., 2026a).

Learning after deployment differs from the gym-based agentic training before deployment. In a training gym, agents can revisit tasks and receive feedback on new attempts. During deployment, tasks arrive along the user's workflow, and users may move on after evaluating one delivered result. The agent must therefore use the limited experience and feedback from completed tasks to improve its performance on subsequent tasks (Guo et al., 2026). We study this process of *deployment-time self-evolution*, in which agents adapt their model parameters or persistent harness state through continued use. Specifically, there exists a task stream where the agents can only complete one live execution and receive one external outcome feedback per task. Existing work explores different ways to use this experience. One direction updates memories or reusable skills while keeping the model fixed (Chhikara et al., 2025; Agrawal et al., 2026; Alzubi et al., 2026; Ma et al., 2026), while another updates model parameters through live interactions (Wang et al., 2026b; Xia et al., 2026). Both approaches aim to turn deployment experience into transferrable signals on subsequent tasks.

Evaluating deployment-time self-evolution requires both realistic assistant tasks and opportunities to reuse experience. Agents should complete work through a harness, while different tasks should share materials, requirements, or working procedures (Asawa et al., 2026). Existing benchmarks provide useful task resources (Bai et al., 2026; Ji et al., 2026; Guo et al., 2026), but further task construction is needed to support long streams of related tasks under a common deployment-time protocol. We

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.

therefore introduce DEAL (Deployment-time Evaluation of Agent Learning), an executable testbed covering *general workplace assistance* and *AI research*. We construct related tasks from workspace files and research materials, and refine them through execution and quality checks. In DEAL, shared abilities and work contexts connect the tasks, while rubric-based user simulation provides feedback on the execution artifacts. DEAL adopts a single-feedback protocol to study learning with limited user supervision, where the agent receives one user feedback for each online task.

Under DEAL, we implement a range of baselines covering both training-free methods and parameter-based learning. We further study parameter-based learning, as recent work shows that parameter updates can help models internalize experience and improve task execution (Xia et al., 2026; Harrington et al., 2026). Under the single-feedback constraint, an intuitive approach is to train the model from live trajectories and their rewards through single-rollout reinforcement learning (RL) (Hou et al., 2026; Hu et al., 2026). However, the limited feedback may reveal problems in the delivered result without providing a better trajectory to learn from. Whether the observed result is good or poor, alternative behaviors that may provide advantage signals remain unexplored. We refer to this limitation as an *exploration bottleneck*.

In this paper, we point out that one user feedback need not limit learning to one trajectory, and the agent can asynchronously get more rollouts in free time. Based on this observation, we propose Self-Rewarded Exploration (SRE). After live execution, SRE generates additional trajectories in isolated copies of the saved environment snapshots. These candidates receive no new user evaluations. Instead, SRE uses the live trajectory and its feedback as an anchor to construct a shared rubric and assign self-rewards. The internal trajectories and their self-rewards are then used for GRPO-style updates (Shao et al., 2024). Each update group uses only self-rewards, as mixing them with external scores may distort the advantages. The additional rollouts provide learning experience for future tasks without replacing the current delivery. We further theoretically analyze when additional rollouts yield better candidates despite biased self-rewards.

We conduct experiments with Gemma4-E4B-IT (Gemma Team, 2026b), comparing SRE with CASCADE (Guo et al., 2026), EvoSkill (Alzubi et al., 2026), and REINFORCE++ (Hu et al., 2025), on DEAL. SRE obtains the largest improvement of deployment-time performance and task-solving abilities in all scenarios except one case, while largely preserving general capabilities. Further diagnostics show that additional rollouts provide better candidates and that the feedback anchor reduces selection errors. We also find that mixing external scores with self-rewards can distort group-relative advantages, supporting our use of internal-only reward groups.

Our contributions are summarized as follows:

- We introduce DEAL, a testbed for deployment-time self-evolution on related workplace and AI research tasks, combines harness-based workflow-based execution, shared task structure, and a single-feedback protocol.
- We identify an *exploration bottleneck* in learning from a single live trajectory and propose SRE, which combines internal rollout expansion with feedback-conditioned self-rewards to support parameter learning without additional user evaluations.
- We characterize the trade-off between exploration gain and selection error, and bound the effect of self-reward errors on group-centered policy gradients.

2 Agent Deployment Scenario and Learning Protocol

In this section, we introduce the background of this paper, i.e., deployed agents that assist users with everyday work. These agents interact with files and tools through a harness, and learn from completed tasks to improve their future performance, which we term *deployment-time self-evolution*. Figure 1 shows the difference between deployment-time self-evolution and traditional gym-based evolution, where tasks can be repeatedly executed and evaluated. During deployment, tasks instead follow the user's workflow, and users may move on after providing feedback, prohibiting further live attempts (Guo et al., 2026). Below, we introduce the deployment-time self-evolution protocol and formalize its objective.

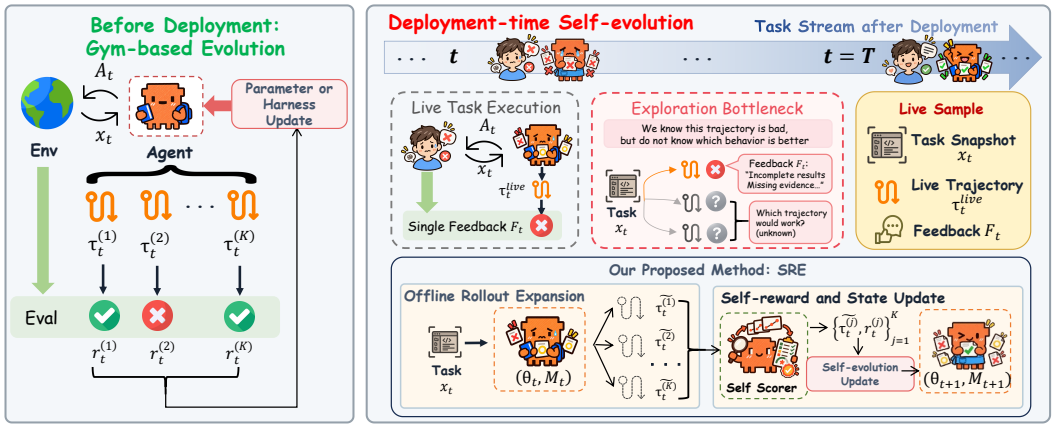

Figure 1: Deployment-time self-evolution from a stream of user tasks. For each task, the agent completes the single live execution and receives user feedback. It then uses the accumulated experience to update its states. The objective is to maxmize the user's satisfaction continually.

2.1 Deployment-time Self-evolution

Let $\mathcal{X} = (x_1, x_2, \cdots, x_T)$ be the task stream where x_t is the t-th arriving task. Denote s_t as the initial environment state, and $(\boldsymbol{\theta}_{t-1}, M_{t-1})$ as the agent's state before that task, consisting of model weights and persistent harness state (e.g., memories or skills). The initial agent state is $(\boldsymbol{\theta}_0, M_0)$. For each task, the agent completes one live trajectory and then receives one outcome feedback:

$$\tau_t \sim \pi_{\boldsymbol{\theta}_{t-1}}(\cdot \mid x_t, s_t, M_{t-1}), \quad F_t \leftarrow \text{ExternalFeedback}(x_t, \tau_t).$$

Here, τ_t contains the complete multi-turn execution, including model actions, tool observations, and the delivered result. The execution history available after task x_t is $\mathcal{H}_t = \{(x_k, s_k, \tau_k, F_k)\}_{k=1}^{t}$. After receiving F_t, the agent can adapt its state using $\mathcal{H}_t$:

$$(\boldsymbol{\theta}_t, M_t) = \mathcal{U}(\boldsymbol{\theta}_{t-1}, M_{t-1}, \mathcal{H}_t),$$

where $\mathcal{U}$ denotes the self-evolution procedure. Self-evolution $\mathcal{U}$ can change an agent's persistent context and harness, i.e., updating M_t while keeping the model parameters fixed (Karten et al., 2026; Zhang et al., 2026b), its parameters, i.e., updating $\boldsymbol{\theta}_t$ (Wang et al., 2026b; Lei et al., 2026), or both its harness and weights (Hebbar et al., 2026; Xia et al., 2026).

2.2 Deployment-time Learning Objective

To measure the value of deployment-time self-evolution, we compare the learning agent with a frozen base agent over the same task stream $\mathcal{X}$. The comparison quantifies the benefit of learning throughout deployment, rather than only the performance of the final agent.

Let $u_t(\tau)$ denote the utility of the trajectory τ in the task x_t. We first collect the base trajectory $\tau_t^{\text{base}} \sim \pi_{\boldsymbol{\theta}_0}(\cdot \mid x_t, s_t, M_0)$ using the fixed initial agent state. Motivated by Guo et al. (2026); Asawa et al. (2026), the objective is to design a self-evolution method $\mathcal{U}$ that maximizes the expected average utility gain over the frozen base agent:

$$J_T(\mathcal{U}) = \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{\tau_t, \tau_t^{\text{base}}} \left[u_t(\tau_t) - u_t(\tau_t^{\text{base}}) \right]. \tag{1}$$

A positive $J_T(\mathcal{U}) > 0$ indicates an average gain over the frozen base during deployment, and a higher value of $J_T(\mathcal{U})$ represents a better gain. To rigorously evaluate the self-evolving agents after deployment, in the next section, we introduce a testbed within two everyday scenarios, specifying the task environments and evaluation protocol.

3 DEAL: A DEPLOYMENT-TIME SELF-EVOLUTION TESTBED

In this section, we introduce DEAL (Deployment-time Evaluation of Agent Learning), a testbed for studying deployment-time self-evolution on two domains, *workplace assistance* and *AI research*.

3.1 MOTIVATION

Evaluating deployment-time self-evolution requires a testbed that reflects how assistants work and learn during use. Users typically deploy models through a harness for tool use and task execution (OpenAI, 2025; Steinberger et al., 2026). Their needs also extend beyond question answering to completing tasks in a workspace. Meanwhile, feedback from completed tasks can only benefit subsequent work. The task stream therefore needs opportunities to reuse earlier experience (Alquier et al., 2017; Asawa et al., 2026). These considerations lead to three key requirements for testbeds:

(1) The agent should be wrapped in a harness system like Codex (OpenAI, 2025);

(2) The tasks should reflect realistic assistant workflows, requiring agents to interact with files and tools in a workspace to produce artifacts, rather than merely answering questions;

(3) The task stream should share structural biases to transfer previous knowledge to later tasks (Guo et al., 2026; Shu et al., 2026; Harrington et al., 2026).

Existing benchmarks or testbeds provide useful foundations for evaluation, yet we find that no single one clearly captures these factors. DTLBench (Guo et al., 2026), ClawArena (Ji et al., 2026) and MetaClaw (Xia et al., 2026) provide promising examples of task stream, while the task form still lies in typical QAs. ClawGym (Bai et al., 2026) offers everyday tasks at scale, but constructs self-contained requests with task-specific resources.

Thus, to mitigate the gap, we further introduce DEAL, which is constructed to bring these requirements together in related task streams.

3.2 TASK CONSTRUCTION AND OVERVIEW

Inspired by prior work on agentic data construction (Kimi Team, 2026; Bai et al., 2026; Hua et al., 2026), we combine grounded task synthesis with execution-based refinement. Figure 2 summarizes our three-stage pipeline, with all construction agents operating through a harness.

We begin with **Task and Evidence Preparation**. The resource curator agent organizes source materials into an ability pool of reusable capabilities and an evidence pool of files and dependencies. Abilities are shared and recombined across tasks to support cross-task learning. In **Grounded Task Synthesis**, the curator turns selected abilities into task scenarios and retrieves supporting evidence. The task generator agent then uses these inputs to create candidate tasks and evaluation rubrics. Finally, in **Iterative Quality Refinement**, the task tester executes candidates for rubric-based evaluation. The task revisor uses execution and judging traces to revise the tasks and rubrics for retesting. Cases with invalid or insufficient evidence return to the resource curator. Tasks are accepted for four criteria: clear scope and evidence, feasibility and skill alignment, no answer or process leakage, and verifiable rubrics. We also manually review tasks and rubrics against these criteria and return identified problems for revision. Construction details and model configurations are provided in Appendix C.1.

We instantiate DEAL in two assistant-work scenarios: *General Workplace Assistance* (DEAL-WS) and *AI Research* (DEAL-RS).

DEAL-RS. We screen all papers posted to arXiv's `cs.AI` category between September 2025 and June 2026 and retain the 252 papers with associated open-source code repositories. The initial ability pool is built from AI research skills selected from ClawHub and SkillsHub. We combine these abilities with evidence to construct 1,800 online tasks.

DEAL-WS. We use company-level workspace files from Workspace-Bench (Tang et al., 2026) as source materials. We build its ability pool from workspace-based benchmarks (Tang et al., 2026; Bai et al., 2026; Z.ai, 2026; Qwen Team & Alibaba Data, 2026; Ding et al., 2026). We combine these abilities with the workspace files and dependencies to construct 2,000 online tasks.

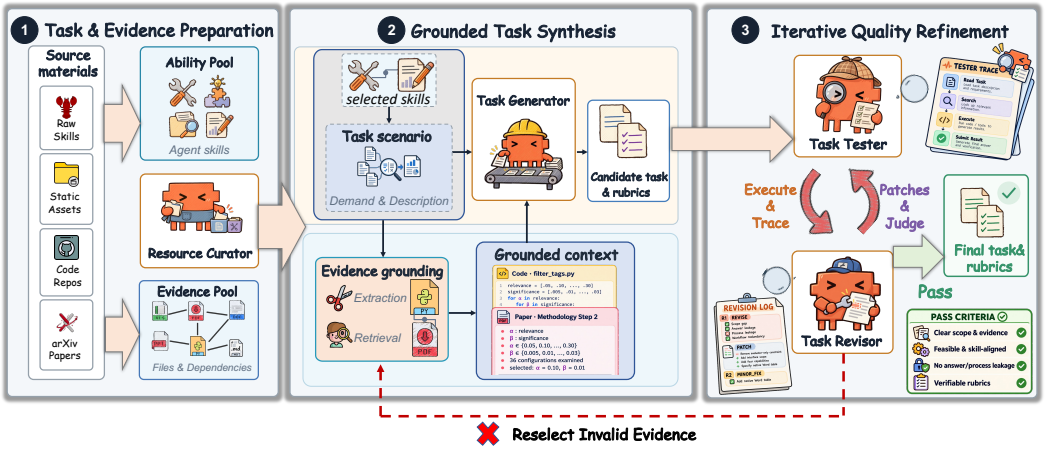

Figure 2: The shared task construction pipeline. Source materials are matched with capability requirements and evaluation rubrics to generate tasks with the resources needed for execution and the criteria used to evaluate the work.

For each scenario, we additionally filter 200 high-quality tasks for held-out evaluation. We provide a detailed overview of DEAL-WS and DEAL-RS in Appendix C.2.

3.3 Execution, Feedback, and Evaluation Protocol

In DEAL, all methods use `nanobot` harness (HKUDS, 2026) to execute tasks and record trajectories and artifacts. We employ a user agent equipped with task rubrics to evaluate the delivered work and simulate user feedback. For ease of analysis, we structure this feedback as $F_t = (u_t, c_t)$, where $u_t \in [0, 1]$ is the utility score and c_t is textual feedback. Both are returned once per task, while the rubrics and reference answers remain hidden. We compute online gains from live outcomes using Equation (1). Held-out evaluation uses the final agent state after the full task stream.

3.4 Implemented Baselines

We implement both training-free and parameter-based learning methods in DEAL. The training-free baselines use past experience to guide later tasks without updating model parameters. CASCADE retrieves successful cases as examples (Guo et al., 2026), while EvoSkill turns feedback into reusable skills (Alzubi et al., 2026). For parameter-based learning, we use REINFORCE++ (Hu et al., 2025) to update the model from live trajectories and scalar utilities.

We further study parameter-based learning, as recent work shows its potential to internalize experience and enhance ability (Harrington et al., 2026; Xia et al., 2026). However, a single rollout may leave better behaviors undiscovered, even when its outcome is accurately evaluated. In the next section, we analyze this exploration bottleneck and introduce our method, Self-Rewarded Exploration.

4 SRE: Feedback-conditioned Self-rewarded Exploration

In this section, we first discuss the exploration bottleneck of single-rollout RL in Section 4.1, motivating us to introduce SRE for learning from additional rollouts with imperfect self-rewards in Section 4.2. In Section 4.3, we provide theoretical support for SRE.

4.1 Exploration Bottleneck of Single-rollout RL

An intuitive approach to parameter-based learning is single-rollout RL (Hu et al., 2025; Hou et al., 2026; Hu et al., 2026). It has shown success in gym-based agentic training before deployment, as it avoids group synchronization and supports asynchronous updates (Hou et al., 2026). These environ-

ments still allow repeated attempts and feedback on the same task, providing further opportunities to discover different behaviors (Xu & Ding, 2026).

However, such opportunities cannot be assumed in deployment, as users may move on after one outcome feedback. Whether the observed result is good or poor, alternative behaviors may remain unexplored. We refer to this limitation as an *exploration bottleneck*. A learned value model can support advantage estimation, but value pretraining and critic updates require careful configuration and add to the demands of local deployment (Hou et al., 2026; Pan et al., 2026; Qi et al., 2026).

We instead use additional rollouts to support both exploration and advantage estimation. In deployment, self-evolution can run asynchronously with task execution (Wang et al., 2026b; Xia et al., 2026; Tiwari et al., 2026). From a saved task snapshot, the agent can generate additional offline rollouts in isolated copies when the inference server is free. Once these candidates receive cheap surrogate rewards, they can also provide signals to identify informative behaviors and estimate advantage. This motivates Self-Rewarded Exploration (SRE), which views the live feedback as privileged information to assign rewards to the agent's extra rollouts.

4.2 Learning with Additional Rollouts

SRE consists of three components: offline rollout expansion, feedback-conditioned self-reward estimation, and Group-Relative Policy Optimization (GRPO; Shao et al., 2024)-style update. The complete procedure is summarized in Appendix A.

Offline rollout expansion. For each task x_t, we save the initial environment state s_t before live execution. After receiving feedback F_t, we restore isolated copies of this snapshot and generate K additional trajectories $\tilde{\tau}_t^{(k)} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid x_t, s_t, M_{t-1})$, $k = 1, \ldots, K$. Here, each rollout uses the original task input and initial environment, and the live feedback is not added to the actor's input to keep the input conditions consistent with live execution (Hübotter et al., 2026). Each trajectory is generated in the forked snapshot, without modifying the user's original workspace.

Feedback-conditioned self-reward estimation. We estimate the surrogate rewards of these additional rollouts by having the agent score its own internal rollouts. The detailed procedure has two stages. First, a self-rubric agent uses the live trajectory and its feedback as an anchor to construct a task-specific rubric $\mathcal{Q}_t$. The rubric is shared across all internal candidates. Second, a self-scoring agent evaluates each candidate's execution and artifacts against $\mathcal{Q}_t$ to assign a reward $\hat{r}_t^{(k)}$. The score reflects both task requirements and user preferences expressed in the feedback.

GRPO-style update. Having the surrogate self-rewards in hand, we can use GRPO-style update to train the model (Shao et al., 2024). In implementation, we exclude the live response user reward to calculate the group-level advantage, as we observe a systematic offset from these external scores.

Figure 3 shows that self-rewards tend to be higher than ground-truth rewards on the evaluated Gemma4-E4B-IT trajectories. Mixing them with the live reward can shift the group mean and distort the advantages. We therefore use only the K offline trajectories and their self-rewards in each rollout group.

Formally, for each task, let $\bar{\hat{r}}_t = K^{-1} \sum_{k=1}^{K} \hat{r}_t^{(k)}$ denote the mean self-reward. We compute the group-relative advantages as $\hat{A}_t^{(k)} = (\hat{r}_t^{(k)} - \bar{\hat{r}}_t)/[\mathrm{Std}(\{\hat{r}_t^{(j)}\}_{j=1}^{K}) + \varepsilon]$, where ε is a small constant for numerical stability.

We apply a clipped GRPO-style policy update using these advantages, where only agent-generated tokens contribute to the policy loss, and tool observations serve as context.

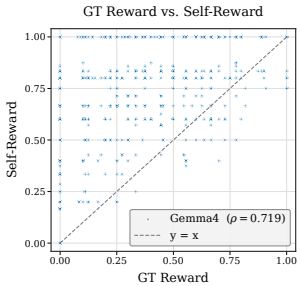

Figure 3: GT vs. self-rewards for Gemma4 on DEAL-RS.

4.3 Theoretical Justification: A Multi-armed Bandit View

In this subsection, we analyze rollout expansion with biased self-rewards from a Multi-Armed Bandit (MAB) perspective. For a fixed task x and a fixed policy $\pi_{\boldsymbol{\theta}}$, we view each complete trajectory

as an arm with return $R_x(\tau) = u_x(\tau) \in [0,1]$. Let $\mathcal{T}_K$ contain K internal trajectories sampled independently from this policy, and let $R_x^* = \sup_\tau R_x(\tau)$. Let $\hat{R}_x(\tau)$ denote its self-reward. Considering the candidate selected by self-reward, i.e., $\hat{\tau}_K \in \arg\max_{\tau \in \mathcal{T}_K} \hat{R}_x(\tau)$, its expected simple regret can be decomposed as

$$\textsc{Reg}_K(x) = R_x^* - \mathbb{E}[R_x(\hat{\tau}_K)] = \underbrace{R_x^* - \mathbb{E}\left[\max_{\tau \in \mathcal{T}_K} R_x(\tau)\right]}_{C_K(x)\text{: coverage gap}} + \underbrace{\mathbb{E}\left[\max_{\tau \in \mathcal{T}_K} R_x(\tau) - R_x(\hat{\tau}_K)\right]}_{S_K(x)\text{: selection error}}.$$

The first term $C_K(x)$ measures the gap from failing to discover higher-return trajectories. The second term $S_K(x)$ measures the loss when self-rewards fail to select the best available candidate. For a single rollout, there is no candidate selection, so $S_1(x) = 0$ and $\textsc{Reg}_1(x) = C_1(x)$. Increasing K can reduce $C_K(x)$ through broader exploration, but self-reward errors may offset this gain. The following proposition bounds this selection error.

Proposition 1 (Exploration with biased rewards)**.** *Suppose the estimated returns satisfy*

$$\hat{R}_x(\tau) = R_x(\tau) + b_x + e_x(\tau), \qquad |e_x(\tau)| \le \epsilon_r$$

for every candidate almost surely, where b_x is shared across the group. Then

$$S_K(x) \le 2\epsilon_r,$$
$$\textsc{Reg}_1(x) - \textsc{Reg}_K(x) \ge C_1(x) - C_K(x) - 2\epsilon_r.$$

Thus, expanded rollouts achieve lower expected selection regret when $C_1(x) - C_K(x) > 2\epsilon_r$.

With Proposition 1, we can compare group-centered gradients from self-rewards and true returns.

Corollary 1 (Group-relative gradient error)**.** *Under the assumptions of Proposition 1, let $R_k = R_x(\tilde{\tau}^{(k)})$ and $\hat{r}_k = \hat{R}_x(\tilde{\tau}^{(k)})$, with group means $\bar{R}$ and $\bar{\hat{r}}$. Suppose $z_k = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tilde{\tau}^{(k)} \mid x)$ satisfies $\|z_k\|_2 \le G$ almost surely. Treating rewards as fixed weights, define the GT and self-reward gradient estimates on the same candidate group as*

$$g_K = \frac{1}{K}\sum_{k=1}^{K}(R_k - \bar{R})z_k, \qquad \hat{g}_K = \frac{1}{K}\sum_{k=1}^{K}(\hat{r}_k - \bar{\hat{r}})z_k.$$

Then $\|\hat{g}_K - g_K\|_2 \le 2\epsilon_r G$ almost surely.

Both proofs are provided in Appendix B. Proposition 1 shows that exploration yields better candidates in expectation when coverage gains exceed selection errors. Corollary 1 shows that small scoring errors keep group-centered gradients close to their GT counterparts. Together, these results support learning from additional rollouts with feedback-conditioned self-rewards.

5 Experiments

In this section, we conduct experiments on DEAL to study how agents learn from deployment experience. We compare different adaptation methods and examine the effectiveness of SRE.

5.1 Experimental Settings

Evaluation settings. We evaluate online utility gains and held-out performance after deployment on DEAL-RS and DEAL-WS. We also use AARRI-Bench (Wang et al., 2026a) for DEAL-RS, and PinchBench (PinchBench, 2026) and ClawGym-Bench (Bai et al., 2026) for DEAL-WS. Users also use these assistants for everyday conversations, so task improvements should not come at the cost of general capabilities. We therefore evaluate MMLU-Pro (Wang et al., 2024), BBEH (Kazemi et al., 2025), GPQA-D (Rein et al., 2023), AIME 2026 (Balunović et al., 2025), MATH-500 (Lightman et al., 2023), HumanEval+ (Liu et al., 2023), and IFBench (Pyatkin et al., 2025). These tests are run without harnesses, and their mean score is reported as *General Capabilities*. Online gain follows Equation (1); benchmark metrics and settings are given in Appendix D.1.

Table 1: Results for Gemma4-E4B-IT on DEAL-RS and DEAL-WS. Online gains are relative to the base agent as in Equation (1). Task abbreviations are defined in Appendix C.2. Frozen models share the base model's general-capability scores ("–"). Best and second-best results within each domain are **bold** and underlined, respectively.

Method	Online Gain	DEAL-RS (ID)						OOD Evaluation	General Capabilities
		Comp.	Align.	Verify.	Workflow	Audit	Avg.	AARRI-Bench	
Base	0.00	16.68	27.98	33.86	14.09	18.39	21.30	35.38	**57.04**
EvoSkill	-1.07	26.04	29.90	34.61	19.71	24.70	26.90	30.88	–
CASCADE	+2.49	25.52	23.93	37.30	16.38	19.59	24.93	35.25	–
REINFORCE++	+4.91	25.84	**40.27**	39.75	24.19	22.43	29.93	37.97	56.10
SRE (Ours)	**+7.07**	**30.96**	36.13	**50.88**	**26.16**	**37.69**	**35.28**	**38.52**	55.11

Method	Online Gain	DEAL-WS (ID)						OOD Evaluation		General Capabilities
		Ops.	Log.	Res.	Dev.	Prod.	Avg.	PinchBench	ClawGym-Bench	
Base	0.00	48.52	44.78	38.34	37.02	32.55	40.24	**62.98**	46.61	**57.04**
EvoSkill	-6.50	47.58	49.22	35.35	38.61	33.74	40.90	50.87	**46.86**	–
CASCADE	+0.92	**52.50**	**51.08**	32.70	**54.84**	36.84	**45.59**	56.18	46.26	–
REINFORCE++	-2.06	36.30	39.56	42.20	41.87	30.86	38.16	56.78	46.36	54.93
SRE (Ours)	**+0.99**	41.56	36.97	**45.17**	48.24	**38.28**	42.04	51.24	44.03	55.25

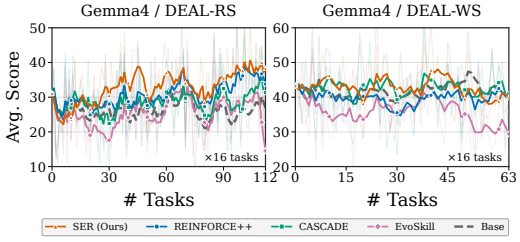

Figure 4: Live task scores during deployment. Methods share the task order within each panel.

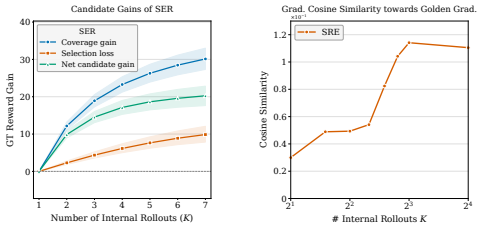

Figure 5: Effect of K on DEAL-RS. Left: candidate gains. Right: gradient alignment with a 64-rollout GT reference.

Baselines and training. We use Gemma4-E4B-IT (Gemma Team, 2026b) for the main experiments. We compare SRE with the frozen base agent, CASCADE (Guo et al., 2026), EvoSkill (Alzubi et al., 2026), and REINFORCE++ (Hu et al., 2025), using the implementations in Section 3.4. Within each domain, methods share the task order and feedback interface. SRE uses $K = 7$ internal rollouts per task, with rollout generation and model updates running asynchronously. Full configurations are given in Appendix D.1.

5.2 Main Results

Table 1 and Figure 4 show that SRE achieves the highest online gains on both Gemma4 streams: $+7.07$ on DEAL-RS and $+0.99$ on DEAL-WS, with the latter close to CASCADE's $+0.92$. SRE also exceeds REINFORCE++ by 5.35 and 3.88 points on the respective held-out sets, although CASCADE remains best on workspace ID evaluation.

The research gains extend to AARRI-Bench, but SRE underperforms Base and REINFORCE++ on both workplace OOD benchmarks. Its General Capabilities score also decreases after adaptation in both domains. Thus, continuous online improvement without forgetting in model weight perspective remains challenging.

5.3 Analysis of Self-Rewarded Exploration

We analyze 100 randomly selected Gemma4 task groups from DEAL-RS, each with seven internal trajectories and paired Ground-Truth (GT) and self-reward scores. Internal GT scores are used only for offline diagnostics. Detailed experiemental settings can be found in Appendix E.1.

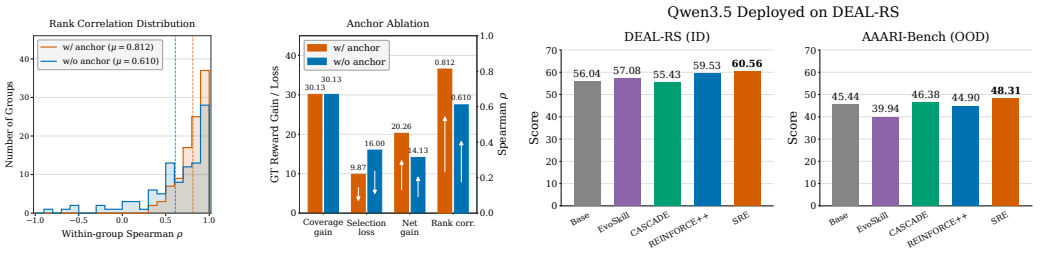

Figure 6: Feedback-anchor ablations on Gemma4: reward ranking (left) and candidate selection (middle). Right: generalization to Qwen3.5-9B after deployment on DEAL-RS.

Effect of the rollout count. Figure 5 (left) shows increasing net candidate gain with K; at $K = 7$, self-reward selection outperforms uniform selection by 20.26 GT points. Since SRE learns from the whole group, we also compare its gradient with a 64-rollout GT-scored GRPO reference using cosine similarity. Alignment levels off around $K = 7$–8 (right), supporting $K = 7$ as a practical balance between alignment and rollout cost.

Role of the feedback anchor. With fixed candidates, anchor-conditioned scoring has higher mean within-task Spearman correlation (0.812 versus 0.610) and net candidate gain (20.26 versus 14.13; Figure 6, left and middle). The subset favors larger no-anchor ranking losses, and some scoring reruns used a different endpoint; these comparisons are descriptive (Appendix E.1).

Generalization to another model family. We further examine deployment-time self-evolution with Qwen3.5-9B (Qwen Team, 2026) on DEAL-RS. SRE exceeds REINFORCE++ by 1.03 ID points and 3.41 AARRI-Bench points (Figure 6, right), suggesting great generalization.

Additional analyses. Further analyses examine reward grouping and its effect on advantages (Appendix E.1), candidate coverage and selection errors (Appendix E.2), and limitations of training-free adaptation (Appendix E.4). Case studies illustrate memory retrieval, skill use, and feedback-conditioned scoring (Appendix F).

6 Related Work

Deployment-time self-evolution studies how agents improve on subsequent tasks using experience collected during use (Guo et al., 2026; Asawa et al., 2026). Existing methods retain experience through memories or reusable skills while keeping the model fixed (Chhikara et al., 2025; Alzubi et al., 2026), or update model parameters through live interactions (Wang et al., 2026b; Xia et al., 2026). Single-rollout RL supports learning from individual trajectories and avoids group synchronization in asynchronous agentic training (Hu et al., 2025; Hou et al., 2026; Hu et al., 2026). DEAL provides related workspace task streams for comparing these adaptation methods with a common single-feedback protocol. SRE further addresses the exploration bottleneck by generating internal trajectories and judge them with the anchor feedback, without requesting additional user evaluations.

7 Discussion and Conclusion

We study deployment-time self-evolution with limited user feedback and introduce DEAL for general workplace and AI research tasks. SRE expands internal rollouts and uses the original feedback to construct self-rewards for learning. Experiments show positive online gains and improved research-task generalization, although gains vary across settings and some general capability loss remains. Our evaluation uses simulated user feedback, while SRE requires restorable snapshots and extra computation and remains sensitive to self-reward quality. Future work will study real-user task streams, adaptive rollout budgets, and more reliable self-rewards. Preserving general capabilities during continued adaptation is another important direction.

AI STATEMENT

In this work, we used generative AI tools for translation and language-related assistance, including improving grammar, clarity, concision, and the presentation of author-written text. We have not used generative AI tools for formulating research questions or hypotheses, developing the conceptual or theoretical framework, designing the methodology or experiments, implementing the proposed method, generating or processing data, or analyzing and interpreting the experimental results. We have reviewed all AI-assisted work. All AI-assisted revisions were checked against the authors' intended meaning and reviewed for technical accuracy. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our code is available at https://anonymous.4open.science/r/SER-DEE0/. We provide the pseudocode of SRE in Appendix A and the proofs of our theoretical results in Appendix B. The task construction pipeline and dataset statistics of DEAL are described in Appendix C. Appendix D documents the training infrastructure, rollout and evaluation settings, and baseline implementations. The group selection, reward comparison, and candidate subset analysis procedures used in our offline diagnostics are detailed in Appendices E.1 and E.2.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Pierre Alquier, The Tien Mai, and Massimiliano Pontil. Regret bounds for lifelong learning. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 261–269. PMLR, 20–22 Apr 2017.

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

Anomaly. OpenCode: The open source coding agent. https://github.com/anomalyco/opencode, n.d. Accessed September 26, 2026.

Anthropic. Claude Code: Overview. https://code.claude.com/docs/en/overview, n.d. Accessed September 26, 2026.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments, 2026. URL https://arxiv.org/abs/2606.05661.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, Wayne Xin Zhao, and Ji-Rong Wen. ClawGym: A scalable framework for building effective Claw agents, 2026. URL https://arxiv.org/abs/2604.26904.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. MathArena: Evaluating LLMs on uncontaminated math competitions, 2025. URL https://arxiv.org/abs/2505.23281.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic AI, 2025. URL https://arxiv.org/abs/2506.02153.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

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

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, et al. WildClawBench: A benchmark for real-world, long-horizon agent evaluation. *arXiv preprint arXiv:2605.10912*, 2026.

Gemma Team. Gemma 4 technical report, 2026a. URL https://arxiv.org/abs/2607.02770.

Gemma Team. Gemma 4 technical report, 2026b. URL https://arxiv.org/abs/2607.02770.

GLM-5-Team. GLM-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Siyuan Guo, Yali Du, Hechang Chen, Yi Chang, and Jun Wang. CASCADE: Case-based continual adaptation for large language models during deployment. *arXiv preprint arXiv:2605.06702*, 2026. URL https://arxiv.org/abs/2605.06702.

Anne Harrington, Nayan Saxena, Michael Murphy, Anastasia Borovykh, Zeyu Yun, Sridhar Kamath, Ara Eindra Kyi, Trevor Darrell, Jitendra Malik, and Yutong Bai. When does continual learning require learning, 2026. URL https://arxiv.org/abs/2607.07847.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. SIA: Self improving AI with harness & weight updates, 2026. URL https://arxiv.org/abs/2605.27276.

HKUDS. nanobot: Ultra-lightweight, open-source, self-hosted personal AI agent framework. https://github.com/HKUDS/nanobot, 2026.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026. URL https://arxiv.org/abs/2607.07508.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization. *arXiv preprint arXiv:2501.03262*, 2025. URL https://arxiv.org/abs/2501.03262.

Jian Hu, Yifan Zhang, Hao Zhang, Binfeng Xu, Shaokun Zhang, Hongqing Peng, Zhiding Yu, Pavlo Molchanov, Jan Kautz, and Yi Dong. FlashREINFORCE: Critic-free single-rollout asynchronous RL for agentic language models, 2026. URL https://github.com/yifanzhang-pro/FlashREINFORCE.

Zhanbo Hua, Yifan Yao, Weihao Xie, Yongchi Zhao, Minghao Liu, Ruizhi Qiu, Zhewei Huang, Zun Wang, Yiyan Ji, Yunhai Ye, Letian Zhu, Xinping Lei, Han Li, Zhiyuan Ma, Zili Wang, Zhaoxiang Zhang, and Jiaheng Liu. CLI-Universe: Towards verifiable task synthesis engine for terminal agents, 2026. URL https://arxiv.org/abs/2606.22883.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

Haonian Ji, Kaiwen Xiong, Siwei Han, Peng Xia, Shi Qiu, Yiyang Zhou, Jiaqi Liu, Jinlong Li, Bingzhou Li, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. ClawArena: Benchmarking AI agents in evolving information environments, 2026. URL https://arxiv.org/abs/2604.04202.

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

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. BIG-Bench Extra Hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://aclanthology.org/2025.acl-long.1285/.

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

Jingdi Lei, Di Zhang, Junxian Li, Weida Wang, Kaixuan Fan, Xiang Liu, Qihan Liu, Xiaoteng Ma, Baian Chen, and Soujanya Poria. δ-mem: Efficient online memory for large language models, 2026. URL https://arxiv.org/abs/2605.12357.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.01210.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. SkillClaw: Let skills evolve collectively with agentic evolver, 2026. URL https://arxiv.org/abs/2604.08377.

MiniCPM Team. MiniCPM4: Ultra-efficient LLMs on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

NVIDIA. NeMo RL: A scalable and efficient post-training library. https://github.com/NVIDIA-NeMo/RL, 2025. GitHub repository.

OpenAI. Introducing Codex. https://openai.com/index/introducing-codex/, May 2025. Accessed: 2026-09-19.

OpenAI. GPT-5.5 system card. https://openai.com/index/gpt-5-5-system-card/, 2026a.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026b.

Haoxuan Pan, Chuyue Zhou, Xin Li, et al. JustRL II: Scaling small LLMs to 128k reasoning with a critic. Technical blog, 2026. URL https://panhaoxuan.notion.site/justrl-ii-scaling-small-llms-to-128k-reasoning-with-a-critic.

PinchBench. PinchBench: Real-world benchmarks for AI coding agents. https://github.com/pinchbench/skill, 2026. Accessed: 2026-09-20.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025. URL https://arxiv.org/abs/2507.02833.

Penghui Qi, Xiangxin Zhou, and Wee Sun Lee. Best practice critic optimization, 2026. URL https://arxiv.org/abs/2608.23566.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team and Alibaba Data. QwenClawBench: Real-user-distribution benchmark for OpenClaw agents, April 2026. URL github.com/SKYLENAGE-AI/QwenClawBench.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, Li Nanbo, Yibo Yang, Mingchen Zhuge, and Jürgen Schmidhuber. Self-improvements in modern agentic systems: A survey, 2026. URL https://arxiv.org/abs/2607.13104.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. URL https://arxiv.org/abs/1909.08053.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. AgentCL: Toward rigorous evaluation of continual learning in language agents, 2026. URL https://arxiv.org/abs/2606.02461.

Peter Steinberger, OpenClaw Foundation, and contributors. OpenClaw: Your assistant, on your devices, in your chats, 2026. URL https://github.com/openclaw/openclaw.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-Bench 1.0: Benchmarking AI agents on workspace tasks with large-scale file dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

Rishabh Tiwari, Kusha Sareen, Lakshya A Agrawal, Joseph E. Gonzalez, Matei Zaharia, Kurt Keutzer, Inderjit S Dhillon, Rishabh Agarwal, and Devvrit Khatri. Learning, fast and slow: Towards LLMs that adapt continually, 2026. URL https://arxiv.org/abs/2605.12484.

Jiayu Wang, Weijiang Lv, Bowen Fu, Jing Fu, Jiayi Song, Lingyu Zhang, Lanxuan Xue, Luodi Chen, Zepeng Xin, Kaiyu Li, et al. Act as a real researcher: A suite of benchmarks evaluating frontier LLMs and agentic harnesses in the research lifecycle, 2026a. URL https://arxiv.org/abs/2606.07462.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026b. URL https://arxiv.org/abs/2603.10165.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.01574.

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. MetaClaw: Just talk – an agent that meta-learns and evolves in the wild, 2026. URL https://arxiv.org/abs/2603.17187.

Binfeng Xu, Hao Zhang, Shaokun Zhang, Songyang Han, Mingjie Liu, Jian Hu, Shizhe Diao, Zhenghui Jin, Yunheng Zou, Michael Demoret, Jan Kautz, and Yi Dong. Polar: Agentic RL on any harness at scale. *arXiv preprint arXiv:2605.24220*, 2026. URL https://arxiv.org/abs/2605.24220.

Zhongwen Xu and Zihan Ding. Single-stream policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=b61UW62K7W.

Z.ai. ZClawBench: A benchmark for OpenClaw-style agent tasks. https://huggingface.co/datasets/zai-org/ZClawBench, 2026.

Haoran Zhang, Luxin Xu, Zhilin Wang, Runquan Gui, Shunkai Zhang, Haodi Lei, Zihao He, Bingsu He, Chicheng Qin, Tong Zhu, Xiaoye Qu, Yang Yang, Yu Cheng, and Yafu Li. π-bench: Evaluating proactive personal assistant agents in long-horizon workflows, 2026a. URL https://arxiv.org/abs/2605.14678.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026b. URL https://arxiv.org/abs/2510.04618.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An LLM post-training framework for RL scaling. https://github.com/THUDM/slime, 2025. GitHub repository.

A Pseudocode of Self-Rewarded Exploration

Algorithm 1 summarizes SRE under the single-feedback protocol. The actor receives the same task information during live and internal execution. Only the self-rubric and self-scoring stages use the live feedback. We show the logical dependencies in order; internal rollout generation and policy updates can run asynchronously with later live tasks.

Algorithm 1 Self-Rewarded Exploration (SRE)

Require: Initial policy $\pi_{\boldsymbol{\theta}}$, task stream $(x_t)_{t=1}^{T}$, internal rollout count $K = 7$, numerical constant $\varepsilon > 0$
Ensure: Updated policy and one delivered outcome per live task
for $t = 1, \ldots, T$ **do**
  Save the initial task environment s_t and the available harness state M_{t-1}
  Execute $\tau_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid x_t, s_t, M_{t-1})$ in the live workspace
  Deliver its outcome and receive the single external feedback $F_t = (u_t, c_t)$
  Construct the shared rubric $\mathcal{Q}_t$ from (x_t, τ_t, F_t)
  for $k = 1, \ldots, K$ **do**
    Restore an isolated copy of s_t
    Generate $\tilde{\tau}_t^{(k)} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid x_t, s_t, M_{t-1})$ without F_t in the actor input
    Score the candidate trace and artifacts using $\mathcal{Q}_t$ to obtain $\hat{r}_t^{(k)}$
  end for
  Form the internal-only group $\{(\tilde{\tau}_t^{(k)}, \hat{r}_t^{(k)})\}_{k=1}^{K}$
  Compute $\bar{\hat{r}}_t = K^{-1} \sum_{k=1}^{K} \hat{r}_t^{(k)}$ and $\sigma_t = \mathrm{Std}(\{\hat{r}_t^{(k)}\}_{k=1}^{K})$
  Set $\hat{A}_t^{(k)} = (\hat{r}_t^{(k)} - \bar{\hat{r}}_t)/(\sigma_t + \varepsilon)$ for each internal trajectory
  Update $\boldsymbol{\theta}$ with the clipped GRPO-style objective on agent-generated tokens
  Make the updated policy available for subsequent tasks
end for

The live trajectory provides the feedback anchor, but neither its tokens nor its GT reward enter the internal-only update group. Tool observations provide context and are masked from the policy loss. No internal candidate is submitted for another user evaluation. Algorithm 1 describes a valid scored group; missing self-scores are not treated as observed GT labels.

B Proofs for Section 4.3

We use the notation introduced in Section 4.3.

B.1 Proof of Proposition 1

Proposition 1 (Exploration with biased rewards)**.** *Suppose the estimated returns satisfy*

$$\hat{R}_x(\tau) = R_x(\tau) + b_x + e_x(\tau), \qquad |e_x(\tau)| \le \epsilon_r$$

for every candidate almost surely, where b_x is shared across the group. Then

$$S_K(x) \le 2\epsilon_r,$$
$$\mathrm{REG}_1(x) - \mathrm{REG}_K(x) \ge C_1(x) - C_K(x) - 2\epsilon_r.$$

Thus, expanded rollouts achieve lower expected selection regret when $C_1(x) - C_K(x) > 2\epsilon_r$.

Proof. Consider any realization of $\mathcal{T}_K$ on which the assumed error bounds hold. Let

$$\tau_K^* \in \arg\max_{\tau \in \mathcal{T}_K} R_x(\tau), \qquad \hat{\tau}_K \in \arg\max_{\tau \in \mathcal{T}_K} \hat{R}_x(\tau).$$

By the optimality of $\hat{\tau}_K$ under the estimated returns,

$$R_x(\hat{\tau}_K) + b_x + e_x(\hat{\tau}_K) \ge R_x(\tau_K^*) + b_x + e_x(\tau_K^*).$$

The common offset b_x cancels, giving

$$0 \le R_x(\tau_K^*) - R_x(\hat{\tau}_K) \le e_x(\hat{\tau}_K) - e_x(\tau_K^*) \le 2\epsilon_r.$$

Taking expectations yields

$$S_K(x) = \mathbb{E}[R_x(\tau_K^*) - R_x(\hat{\tau}_K)] \le 2\epsilon_r.$$

For a single candidate, $S_1(x) = 0$ and $\textsc{Reg}_1(x) = C_1(x)$. Applying the regret decomposition, we obtain

$$\begin{aligned}\textsc{Reg}_1(x) - \textsc{Reg}_K(x) &= C_1(x) - C_K(x) - S_K(x) \\ &\ge C_1(x) - C_K(x) - 2\epsilon_r.\end{aligned}$$

The right-hand side is positive whenever $C_1(x) - C_K(x) > 2\epsilon_r$, which proves the strict comparison. □

Remark. Under a fixed sampling policy, couple the candidate sets as $\mathcal{T}_{K+1} = \mathcal{T}_K \cup \{\tilde{\tau}^{(K+1)}\}$, where the additional trajectory is sampled independently from the same policy. Then

$$C_K(x) - C_{K+1}(x) = \mathbb{E}\left[\left(R_x(\tilde{\tau}^{(K+1)}) - \max_{\tau \in \mathcal{T}_K} R_x(\tau)\right)_+\right],$$

where $(z)_+ = \max\{z, 0\}$. Thus, the coverage gap decreases strictly whenever the additional rollout has a positive probability of outperforming the current best candidate. The proposition specifies when this coverage gain exceeds the possible loss from reward estimation.

B.2 Proof of Corollary 1

Corollary 1 (Group-relative gradient error)**.** *Under the assumptions of Proposition 1, let* $R_k = R_x(\tilde{\tau}^{(k)})$ *and* $\hat{r}_k = \hat{R}_x(\tilde{\tau}^{(k)})$*, with group means* $\bar{R}$ *and* $\bar{\hat{r}}$*. Suppose* $z_k = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tilde{\tau}^{(k)} \mid x)$ *satisfies* $\|z_k\|_2 \le G$ *almost surely. Treating rewards as fixed weights, define the GT and self-reward gradient estimates on the same candidate group as*

$$g_K = \frac{1}{K}\sum_{k=1}^{K}(R_k - \bar{R})z_k, \qquad \hat{g}_K = \frac{1}{K}\sum_{k=1}^{K}(\hat{r}_k - \bar{\hat{r}})z_k.$$

Then $\|\hat{g}_K - g_K\|_2 \le 2\epsilon_r G$ *almost surely.*

Proof. Fix a realization of the candidate group on which the reward-error and score-norm bounds hold. Write $e_k = e_x(\tilde{\tau}^{(k)})$ and $\bar{e} = K^{-1}\sum_{k=1}^{K} e_k$. Since $\hat{r}_k = R_k + b_x + e_k$, the group mean satisfies $\bar{\hat{r}} = \bar{R} + b_x + \bar{e}$. Hence,

$$(\hat{r}_k - \bar{\hat{r}}) - (R_k - \bar{R}) = e_k - \bar{e}.$$

The shared offset b_x cancels. Moreover, $|\bar{e}| \le \epsilon_r$, so

$$\left|(\hat{r}_k - \bar{\hat{r}}) - (R_k - \bar{R})\right| \le |e_k| + |\bar{e}| \le 2\epsilon_r.$$

Using the same trajectories and policy parameters for both gradient estimates gives

$$\begin{aligned}\|\hat{g}_K - g_K\|_2 &= \left\|\frac{1}{K}\sum_{k=1}^{K}(e_k - \bar{e})z_k\right\|_2 \\ &\le \frac{1}{K}\sum_{k=1}^{K}|e_k - \bar{e}|\,\|z_k\|_2 \\ &\le 2\epsilon_r G.\end{aligned}$$

This holds almost surely and completes the proof. □

Remark. The bound compares gradient estimates on the same candidate group, with rewards treated as fixed weights. It allows biased and dependent scoring errors. The result concerns the group-centered policy-gradient component without standard-deviation normalization or clipping, rather than the complete asynchronous GRPO update.

C DEAL DETAILS

C.1 TASK CONSTRUCTION DETAILS

Our construction pipeline combines grounded task synthesis with execution-based refinement (Kimi Team, 2026; Bai et al., 2026; Hua et al., 2026). Figure 2 shows the three stages. Each construction agent uses a harness to inspect files, call tools, and record its actions.

Task and evidence preparation. The resource curator organizes the source materials into an ability pool and an evidence pool. The ability pool describes reusable operations, such as comparing benchmark results or reconciling a spreadsheet. These abilities can be combined across tasks, so earlier experience can remain useful as the task context changes. The evidence pool contains the files, code, assets, and dependencies needed to carry out those operations. Before synthesis, we remove skills that require network access, unsupported multimodal inputs or outputs, or capabilities outside the two target domains.

Grounded task synthesis. The curator selects one or more abilities and defines a concrete work scenario. It then retrieves the evidence needed for that scenario. Given these inputs, the task generator writes the user request and its evaluation rubric together. The request specifies the required work and deliverables, while the rubric checks requirements that can be verified from the available evidence and final artifacts. Hidden criteria and reference answers are kept separate from the actor-visible files.

Execution-based refinement and human review. The task tester executes each candidate and records its trajectory and artifacts. The result is evaluated against the candidate rubric. The task revisor inspects both the execution trace and the judging trace, then revises the request or rubric for another test. If the evidence is missing or unsuitable, the case returns to the curator. We also manually review the constructed tasks and rubrics for clear scope, sufficient evidence, feasibility, skill alignment, and verifiable evaluation criteria. The review checks that task materials do not reveal the answer or prescribe the solution process. Problems identified during review are returned for revision.

Models and harnesses. The resource curator and task generator use GLM-5.2-FP8 (GLM-5-Team, 2026). For these two roles, DEAL-RS uses the nanobot harness (HKUDS, 2026), while DEAL-WS uses the Claude Code harness (Anthropic, n.d.). All task testers use nanobot. The tester pool includes GLM-5.2-FP8, DeepSeek-V4-Flash (DeepSeek-AI, 2026), Gemma4-31B-IT (Gemma Team, 2026a), Qwen3.5-4B/9B (Qwen Team, 2026), and Qwen3-4B-Thinking (Qwen Team, 2025). The task revisor uses GPT-5.5 or GPT-5.6 (OpenAI, 2026a;b). All GPT-based construction agents run through the Codex harness (OpenAI, 2025). All non-GPT models are self-hosted. These construction harnesses are separate from the common nanobot executor used in the learning experiments.

C.2 DATASET OVERVIEW

DEAL-RS. We screen papers posted to arXiv's `cs.AI` category from September 2025 to June 2026 and retain 252 papers with associated open-source code repositories. We build the ability pool from research skills collected from ClawHub[1] and SkillsHub[2]. Combining these abilities with local paper and repository evidence gives 1,800 online tasks and 200 held-out tasks. The five task families are benchmark comparison, method-to-code alignment, paper claim verification, research workflow package, and experiment design audit. In Table 1, they are abbreviated as **Comp.**, **Align.**, **Verify.**, **Workflow**, and **Audit**, respectively.

DEAL-WS. We use the company-level files and dependencies in Workspace-Bench (Tang et al., 2026) as source materials. The ability pool draws on Workspace-Bench, ClawGym (Bai et al., 2026), ZClawBench (Z.ai, 2026), QwenClawBench (Qwen Team & Alibaba Data, 2026), and Wild-ClawBench (Ding et al., 2026). We combine these abilities with grounded workspace materials to

[1] https://clawhub.ai
[2] https://www.skillhub.club/

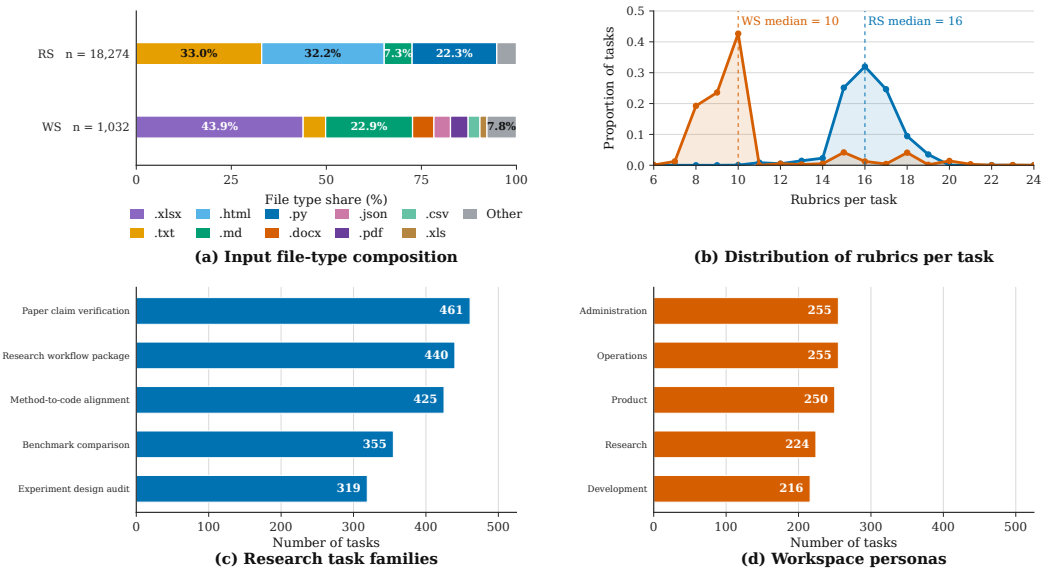

Figure 7: Dataset overview. (a) File-type shares among the inventoried input files in each domain; n denotes files. (b) Distribution of the number of rubric criteria per task. (c) Research task-family counts across 1,800 online and 200 held-out tasks. (d) Workspace persona counts across 1,000 online and 200 held-out tasks.

construct 1,000 online tasks and 200 held-out tasks. The five role categories in Table 1 are operations manager (**Ops.**), logistics manager (**Log.**), researcher (**Res.**), backend developer (**Dev.**), and product manager (**Prod.**). For both domains, **Avg.** denotes the overall task score, **ID** denotes in-domain evaluation, and **OOD** denotes evaluation on external benchmarks.

Dataset characteristics. Figure 7(a) summarizes 18,274 research input files and 1,032 workspace input files. Research inputs mainly consist of text (33.0%), HTML (32.2%), and Python files (22.3%). Workspace inputs are dominated by spreadsheets (43.9% `.xlsx`) and Markdown files (22.9%). These counts describe the input-file inventory, not the number of tasks. Panel (b) shows that research tasks typically require more evaluation criteria: the median is 16, compared with 10 for workspace tasks. Panel (c) contains 461 claim-verification tasks, 440 workflow tasks, 425 alignment tasks, 355 comparison tasks, and 319 audit tasks. Panel (d) reports the workspace persona breakdown shown in the figure: administration (255), operations (255), product (250), research (224), and development (216). Together, the five categories cover all 1,200 workspace tasks.

D Implementation and Experimental Details

D.1 Experimental Settings

Training infrastructure. We build our training infrastructure on slime (Zhu et al., 2025). Our asynchronous design draws on the infrastructure designs of Polar (Xu et al., 2026), OpenClaw-RL (Wang et al., 2026b), and ClawGym (Bai et al., 2026). We decouple rollout execution and reward computation from policy optimization, allowing rollout collection and training to overlap. For Gemma4-E4B-IT, we integrate the Megatron backend adapted in NeMo RL (NVIDIA, 2025; Shoeybi et al., 2019) into this training pipeline.

Training settings. We use Gemma4-E4B-IT (Gemma Team, 2026b) as the deployed model. Table 2 lists the rollout and reward settings for both domains. SRE uses seven internal rollouts per

task and forms advantage groups from their self-rewards only. The self-scoring procedure aggregates three evaluations by their median when all three are available. Rollout generation and model updates run asynchronously; tool observations are masked from the policy loss.

Table 2: Gemma4-E4B-IT rollout and reward configuration for DEAL-WS and DEAL-RS.

Setting	Workspace tasks	Research tasks
Agent harness	nanobot	nanobot
Context window	64K	72K
Maximum output	8K	16K
Temperature	0.7	0.7
Top-p	1.0	1.0
Top-k	-1	-1
Maximum tool iterations	200	200
Reward judge	Gemma4-31B-IT	Gemma4-31B-IT
Judge temperature	0.7	0.7
Judge maximum output	8K	8K
Judge timeout	2,700 s	900 s
Agentic judge timeout	–	1,800 s
Maximum judge attempts	3	3

Evaluation settings. For **evaluation on DEAL**, we evaluate each saved system state on the workplace and research evaluation sets, each containing 200 tasks. Workplace evaluation uses a 64K context window and an 8K output limit, while research evaluation uses an 80K context window and a 16K output limit. The agent harness and tool configuration otherwise match those used during deployment. We keep the evaluated state fixed and do not learn from evaluation feedback.

For **evaluation beyond DEAL**, we use ClawGym-Bench (Bai et al., 2026) and PinchBench (PinchBench, 2026) for workspace tasks, and AARRI-Bench (Wang et al., 2026a) for research tasks. We filter only tasks that are incompatible with the available execution environment. For ClawGym-Bench, we remove 13 tasks that require network access and retain 187 tasks. For PinchBench[3], we remove 20 tasks: 14 require live web access, external repositories, or video processing; two require visual understanding or generation; and four depend on an integrated `fws` and `gh/gws` environment. The remaining split contains 127 tasks. For AARRI-Bench[4], we exclude one task requiring web search and one requiring SVG evaluation, leaving 80 tasks. For AARRI-Bench, we generate one evaluated response per task. Table 3 lists the shared evaluation budgets.

Table 3: Evaluation settings for external benchmarks.

Setting	ClawGym-Bench / PinchBench	AARRI-Bench
Agent harness	nanobot	nanobot
Context window	64K	64K
Maximum output	8K	8K
Temperature	0.7	0.7
Top-p	1.0	1.0
Top-k	-1	-1
Maximum task attempts	3	3
Overall task timeout	4,200 s	4,200 s
Agent execution timeout	3,600 s	3,600 s
Evaluator	See text	pytest verifier
Evaluator timeout	300 s	–
Maximum evaluator attempts	3	–

For ClawGym-Bench, we combine code-based checks with rubric-based assessment by Gemma4-31B-IT (Gemma Team, 2026a). The code checks evaluate mechanically verifiable requirements,

[3] https://github.com/pinchbench/skill
[4] https://github.com/AARR-bench/AARRI-bench

while the rubric judge assesses the delivered work against the task criteria. PinchBench also uses Gemma4-31B-IT for judging. AARRI-Bench uses its deterministic `pytest` verifier. Missing or failed tasks receive zero reward; we do not exclude them from benchmark averages.

General capability evaluation. We evaluate MMLU-Pro (Wang et al., 2024), BBEH (Kazemi et al., 2025), GPQA-D (Rein et al., 2023), AIME 2026 (Balunović et al., 2025), MATH-500 (Lightman et al., 2023), HumanEval+ (Liu et al., 2023), and IFBench (Pyatkin et al., 2025) without tools or web access. We enable reasoning for Gemma4-E4B-IT and use an 80K context limit. For AIME 2026 and MATH-500, we sample four responses per problem and report both average accuracy and pass@4; all other benchmarks use one response per item. For MMLU-Pro and GPQA-D, we report accuracy. For BBEH, we report the harmonic mean and micro average. For HumanEval+, we report pass@1. For IFBench, we report prompt level loose accuracy.

Gemma4-E4B-IT uses a 16K output limit, except on MATH-500, where the limit is 32K. The default decoding parameters are temperature 0.7, top-p 1.0, and top-k -1. MATH-500 instead uses temperature 0.6 and top-p 0.95. We keep these settings fixed across the base and adapted model states.

Parameter-frozen baselines. For CASCADE (Guo et al., 2026), we keep the retrieval encoder fixed as the memory grows during deployment. For EvoSkill (Alzubi et al., 2026), we follow its original skill evolution procedure within our ordered deployment stream. Each update uses only the trajectories and feedback observed up to that point, and newly evolved skills become available only for subsequent tasks. Concretely, we retain EvoSkill's official OpenCode-based skiller (Anomaly, n.d.) and evolution logic, while replacing only its task-execution component with the same Nanobot executor harness used by the other methods. The executor model remains fixed for both baselines.

E Additional Experiments

E.1 Feedback Anchoring and Reward-Source Mixing

We use the saved diagnostic data for the same 100 Gemma4 groups studied in Section 5.3. Each group has one live GT reward and seven internal trajectories with GT and self-reward pairs. The comparison uses historical anchor-conditioned scores and saved scores from a rerun without the anchor, with the candidates and GT labels held fixed. No additional rollout or judge call is needed for this analysis. The recorded self-scores use the median of successful repeated evaluations; we retain recorded zeros and do not replace missing scores with zero. All 100 selected groups have complete pairs.

Selection and uncertainty. The parent no-anchor study selected 112 groups, one per available step, using historical ranking diagnostics. Of these, 106 have defined correlations under both scoring variants. We use the same 100-group subset as the figures in the main text: the 100 lowest values of $\rho_{\mathrm{w/o}} - \rho_{\mathrm{w/}}$. This selection retains groups with larger no-anchor degradation, so the results describe this subset rather than the full task stream. Some no-anchor results come from retries with a changed inference endpoint. The comparison therefore also includes scoring variability; it is not an isolated causal estimate of the anchor's effect. We compute within-group Spearman correlation using average ranks for ties. Its mean is 0.812 with the anchor and 0.610 without it. Intervals use 4,000 paired group-bootstrap samples with seed 20260926 and are conditional on the selected groups.

Advantage comparison. For a reward vector r, we compute $A_i = (r_i - \bar{r})/(\mathrm{Std}(r) + 10^{-8})$ using the population standard deviation. *Internal-only* normalizes the seven internal self-rewards and compares them with the normalized GT rewards of those same seven trajectories. *Mixed* normalizes the live GT reward together with the seven internal self-rewards. Its reference group contains the live GT reward and the seven internal GT rewards. We count a sign flip when the estimated and reference advantages have strictly opposite signs. We exclude internal trajectories with $|A_i^{\mathrm{GT}}| \leq 0.05$ and report the excluded fraction separately. An estimated advantage of zero is reported separately and is not counted as a strict sign flip.

False suppression of the live trajectory. A live trajectory is falsely suppressed when its all-GT advantage exceeds 0.05 but its Mixed advantage is negative. Of the 54 groups with a clearly pos-

Table 4: Reward-source mixing on the fixed 100-group diagnostic subset. Flip rates are over eligible internal trajectories. Exclusion rates use all 700 internal trajectories. Live suppression is reported over all 100 groups.

Self-reward	Grouping	Sign flips	GT near-zero	Live suppression
w/ anchor	Internal-only	$108/681$ (15.86%)	2.71%	–
w/ anchor	Mixed	$127/684$ (18.57%)	2.29%	$29/100$
w/o anchor	Internal-only	$157/681$ (23.05%)	2.71%	–
w/o anchor	Mixed	$164/684$ (23.98%)	2.29%	$38/100$

itive GT-reference live advantage, this occurs in 29 groups with anchor-conditioned self-rewards (53.70%) and 38 groups without the anchor (70.37%). The corresponding rates over all groups are 29% and 38%. The Mixed-minus-Internal-only increase in internal sign flips is 2.71 percentage points with the anchor (95% bootstrap interval: $[0.23, 5.30]$) and 0.92 points without it ($[-0.78, 2.78]$). Thus, the latter increase is not clearly separated from zero in this sample. Estimated zero advantages account for $12/681$ and $6/684$ eligible trajectories in the two with-anchor groups, and $11/681$ and $6/684$ without the anchor. These results motivate keeping externally judged rewards and self-rewards out of the same normalized update group. They do not imply that the anchor should be removed from self-reward estimation.

E.2 Coverage Gain and Selection Loss

We use the same seven internal candidates per group to examine the decomposition in Section 4.3. For each $K \in \{1, \ldots, 7\}$, we enumerate all $\binom{7}{K}$ subsets. For a subset S, let

$$V_{\text{random}}(S) = \frac{1}{K} \sum_{k \in S} R_k, \qquad V_{\text{best}}(S) = \max_{k \in S} R_k, \qquad V_{\text{self}}(S) = \frac{1}{|J(S)|} \sum_{k \in J(S)} R_k,$$

where $J(S) = \arg\max_{k \in S} \hat{r}_k$. The last expression is the expected GT reward under uniform tie-breaking among self-reward maximizers; GT scores are never used to break the tie. We average subsets within each group and then average the 100 groups equally. Coverage gain is $V_{\text{best}} - V_{\text{random}}$, selection loss is $V_{\text{best}} - V_{\text{self}}$, and net candidate gain is $V_{\text{self}} - V_{\text{random}}$. Net gain therefore equals coverage gain minus selection loss. This evaluates a fixed candidate pool and does not rerun learning at different rollout budgets.

Table 5: Exact subset analysis on the fixed 100 groups. All values are GT score points on the 0–100 scale. Coverage is shared because both scoring variants use the same candidates.

		w/ anchor		w/o anchor	
K	Coverage gain	Selection loss	Net gain	Selection loss	Net gain
1	0.00	0.00	0.00	0.00	0.00
2	12.17	2.29	9.88	4.53	7.64
3	18.91	4.37	14.55	8.18	10.73
4	23.28	6.15	17.13	10.94	12.34
5	26.29	7.65	18.64	13.05	13.23
6	28.46	8.88	19.58	14.69	13.77
7	30.13	9.87	20.26	16.00	14.13

Table 5 shows that more candidates expose better GT-scored trajectories. At $K = 7$, $V_{\text{random}} = 29.93$ and $V_{\text{best}} = 60.06$. Selection by self-reward achieves $V_{\text{self}} = 50.19$ with the anchor and 44.06 without it. Scoring errors offset part of the coverage gain, but the remaining net gain is positive. Both variants attain their highest observed net gain at $K = 7$ within the examined range. This supports the current budget in this diagnostic; it does not establish an optimal training budget. In particular, V_{best} is an offline oracle upper bound and is not deployed performance.

E.3 General Capabilities Results

In Table 6, we provide the full results of the general capabilities.

Table 6: Basic abilities of Gemma4-E4B-IT before and after deployment training. All scores are reported on a 0–100 scale. Avg. is the unweighted mean over the seven benchmarks; IFBench uses prompt-level loose accuracy. CASCADE and EvoSkill share Base under weight-only evaluation. Missing results and averages requiring them are shown as "–".

Training setting	Method	MMLU-Pro	BBEH	GPQA-D	AIME 2026	MATH-500	HumanEval+	IFBench	Avg.
Shared	Base / CASCADE / EvoSkill	67.57	32.28	54.55	45.83	69.25	90.85	44.67	57.86
DEAL-RS	REINFORCE++	68.13	32.96	59.60	45.00	68.65	91.46	45.00	56.10
DEAL-RS	SRE (Ours)	67.87	32.57	56.06	47.50	69.25	89.63	43.00	55.11
DEAL-WS	REINFORCE++	67.00	33.16	55.05	48.33	70.60	89.63	41.67	54.93
DEAL-WS	SRE (Ours)	67.87	32.17	56.06	49.17	68.80	90.24	42.33	55.25

E.4 Failure Analysis for Parameter-Frozen Methods

We examine where stored experience fails to improve execution. Memory methods must retrieve relevant cases, while skill methods must produce, access, and follow reusable guidance. We use Qwen3.5-9B and GLM-5.2 as the lower- and higher-performing executors in this diagnostic. This comparison concerns these two models on the evaluated tasks, rather than a general classification of model capability.

E.4.1 Memory-Based Methods

The memory case in Appendix F shows a mismatch between project relevance and procedural relevance. The retrieved case uses the same paper and repository as the new task, but asks for a different operation. It therefore provides little guidance for the numerical checks required by the incoming task. This example illustrates a retrieval limitation; it does not establish how often the limitation occurs across the stream.

E.4.2 Skill-Based Methods

For skill methods, we separate three possible failure points: writing reusable skills, accessing the relevant skills, and executing them correctly. Table 7 summarizes the configurations used in this analysis.

Table 7: Skill diagnostic results on DEAL-RS. Scores use the 0–100 scale. Checkmarks indicate settings that evolve skills; crosses indicate settings without skill evolution. In A–D, the agent autonomously decides whether to read the mounted skills. E and G use no skills. F always uses a fixed set of skills generated by B, with their complete contents injected directly into the prompt; therefore, its file-read rate is not applicable.

ID	Executor	Skiller	Evol.	Read rate	Score
A	Qwen3.5-9B	Qwen3.5-9B	✓	0.04%	57.08
B	Qwen3.5-9B	GLM-5.2	✓	0.51%	56.75
C	GLM-5.2	Qwen3.5-9B	✓	34.70%	87.01
D	GLM-5.2	GLM-5.2	✓	80.00%	87.52
E	Qwen3.5-9B	None	×	0.00%	56.04
F	GLM-5.2	Fixed skills	×	N/A (inline)	86.80
G	GLM-5.2	None	×	0.00%	81.13

Skill generation. In the inspected cases, the lower-performing skiller often retains details of a single task instead of writing a reusable procedure. Its skills may omit verification steps or leave their application conditions unclear. Replacing this skiller with GLM-5.2 produces clearer instructions under manual inspection, but the Qwen executor's score changes only from 57.08 in A to 56.75 in B. The same executor without skills scores 56.04 in E. Better-written skills alone therefore do not explain the final task score.

Skill access. Qwen reads the mounted skills in only 0.04% of executions in A and 0.51% in B. The rates rise to 34.70% and 80.00% when GLM-5.2 is the executor in C and D. The read-rate difference suggests that accessing the guidance is itself a bottleneck. A skill cannot affect an execution through its contents if the executor does not read it. Configuration F removes this access step by placing a fixed set of skills from B directly in the prompt.

Skill execution. Access does not guarantee that the executor follows the guidance correctly. In the paired diagnostic, both models receive the same skill. On the 83 tasks completed by both, GLM-5.2 scores 91.4 on average, compared with 63.0 for Qwen. It scores higher on 65 tasks, ties on 14, and scores lower on four. These averages are conditional on completion by both models. The fixed-skill comparison is consistent with this observation: GLM-5.2 scores 86.80 with the injected skills in F and 81.13 without skills in G. Together, these results separate skill generation, skill access, and skill execution. Improving the stored guidance does not remove the executor's need to collect evidence, follow task constraints, and verify its output.

F Qualitative Case Studies

We present representative traces showing how agents use prior experience and feedback. The first cases examine memory retrieval and the writing, reading, and execution of skills; the final cases illustrate how live feedback can improve self-rubric construction and help select stronger internal trajectories. Each box reports the task, the available experience or feedback, the observed behavior, and the evaluator's diagnosis.

F.1 Memory-Based Case

Same Project, Wrong Operation

Task. Audit the quantitative methodology claims in *Unveiling Ruby*, including survey counts, Cohen's kappa, sampling assumptions, and derived approval and relevance statistics.

Retrieved memory. The retrieved successful case uses the same paper and repository, but solves a different problem: it aligns the paper's data acquisition, filtering, preprocessing, and topic extraction pipeline with the codebase.

Execution trace. The Executor follows the shared paper and repository context, identifies the reported kappa values, and notes several missing local inputs. However, it does not reconstruct the approval and rejection arithmetic, check the rounding of the relevance rate, or inspect the Boolean label conversion in the local kappa script.

Evaluator feedback. The response receives 0.333. The evaluator credits the reported kappa values and broad evidence gaps, but marks the required arithmetic and implementation checks as missing.

Comment and takeaway. This case illustrates why shared entities are not enough for useful memory retrieval. The reusable unit must include the operation and its checks, not only the paper or repository name.

F.2 Skill-Based Cases

A Narrow Skill with Little Reuse

Task. The Executor must reconstruct an ablation table, compare MAE gaps across four datasets, and assess whether those gaps support the paper's claim.

Available skill. The Skiller has produced an `error-diagnoser` skill for a refused connection to the judge. It recognizes a specific network error, retries with exponential backoff, and falls back to an offline judge. These instructions do not address evidence collection, numerical reconstruction, or output validation.

Access trace. The skill is mounted and announced in 33 executions but read in only 2. In this example, the Executor does not read it.

Evaluator feedback. The response reconstructs the four MAE values correctly but omits the paper's claim, the comparison across datasets, and the caveats about statistical significance and other metrics, resulting in a score of 0.30.

Comment and takeaway. This case links skill quality to later access: a record of one infrastructure failure offers little help with a benchmark audit. It illustrates limited reuse, but does not by itself explain every low skill-read rate.

Relevant Guidance, Incomplete Execution

Task. Audit nine TSGBench quality measures and distinguish qualitative observations within a measure family from a formal consistency test across measure families.

Available skill. Both the weaker and stronger Executors receive the same task-specific benchmark-audit guidance.

Weaker Executor. It creates the requested CSV and lists the four shared weaknesses of the model-based measures. However, it omits the better direction of the measures, does not explain which weaknesses are addressed by ED and DTW, and conflates the analyses in Sections 6.1 and 6.4. It satisfies only $2/11$ criteria and scores 0.182.

Stronger Executor. It covers all $11/11$ criteria and scores 1.0.

Comment and takeaway. The same guidance can lead to different outcomes because execution requires capabilities beyond reading the skill. Parameter-frozen adaptation remains limited by the executor's ability to ground and verify the instructions.

F.3 Self-Reward Cases

Single Feedback, Better Rubric

Task. Audit whether a repository consistently implements the paper's three-part INR continuous function, including the residual architecture's hidden dimension, global and group layers, and clustering-derived groups. The output must provide structured paper-to-code comparisons with explicit source paths and alignment verdicts.

Live feedback. The live response scores 0.50. The feedback reveals requirements that are not explicit in the task wording: the three components are a trend polynomial, a seasonal Fourier term, and a residual adaptive group-based MLP; the first two must be connected to `block_func.py`; and architectural claims must cite concrete local paper sources rather than an "implied paper specification."

Without anchor. The task-only rubric asks for three generic components but does not identify the trend and seasonal decomposition or require component-level source grounding. More importantly, it treats repository conformity as the evaluation target: the implementation must use hidden dimension 16, one global layer, one group layer, and K groups. An answer can therefore lose credit for correctly reporting that a value is configurable or that the repository does not match the paper.

With anchor. The feedback-conditioned rubric explicitly checks the trend and seasonal components against `block_func.py`, requires each architectural claim to be tied to a local paper source, and evaluates whether the answer compares the paper and repository values. It therefore permits a correct verdict to be a match, mismatch, or configuration caveat, consistent with the ground-truth rubric.

Why the rubric is better. The difference is not merely that the feedback-conditioned rubric contains more criteria; it corrects what the rubric is intended to measure. On the same seven rollout trajectories, Spearman correlation with ground-truth scores increases from -0.502 to 0.917, pairwise accuracy from 0.313 to 0.875, and top-1 agreement from 0 to 1. The task-only rubric assigns its highest score to two rollouts whose ground-truth scores are only 0.25 and 0.125. The feedback-conditioned rubric instead ranks the three ground-truth-best rollouts, each scoring 0.50, highest. In this example, the single feedback signal improves both criterion coverage and evaluation semantics.

Comment and takeaway. The live feedback corrects the evaluation target: an audit should report whether code matches the paper, rather than reward conformity to one assumed implementation. The anchor can improve candidate ranking by clarifying what counts as a correct answer.

Weak Live Trace, Strong Offline Rollout

Task. Audit whether LogosQ's QFT comparison against PennyLane, Qiskit, Yao.jl, and Q# is fairly designed and whether the reported 120–900× and 6–22× speedup ranges are supported by the paper's benchmark evidence.

Live response. The live response shifts the analysis toward VQE and Heisenberg results and omits the 1–24 qubit range, the claim of identical circuit configurations, the backend switch at 11 qubits, the absence of per-qubit baseline timings, and the relationship between the speedup ranges and the reported algorithmic complexities. It receives a ground-truth score of 0.30.

Feedback-derived evaluation signal. The live feedback identifies these omissions. The resulting self-rubric checks the reported execution times and speedup ranges, the switch from an FFT/state-vector implementation at 1–10 qubits to an MPS backend at 11–24 qubits, the associated complexity claims, and the traceability of conclusions to local paper evidence. The feedback thus converts the failure of one trajectory into criteria that can be applied to new trajectories.

Best additional rollout. One of seven additional rollouts produces a substantially stronger audit. It identifies the 11-qubit backend switch as a comparability confound, distinguishes aggregate speedup ranges from the missing per-qubit and per-baseline timing data, connects the 120–900× claim to the MPS complexity argument, and preserves the caveat that the Q# claim is qualitative rather than quantitative. Its only ground-truth omission is failing to use the required methodology source for the baseline list and the 1–24 qubit range. It receives a ground-truth score of 0.90.

Self-reward selection. For this rollout, the three self-reward evaluations are 0.50, 0.75, and 0.75, giving a median score of 0.75. This is the highest aggregate self-reward among the seven additional rollouts. The selected rollout is also the one with the highest ground-truth score. Across the full rollout pool, self-reward achieves Spearman correlation 0.875, top-1 agreement 1, and zero top-1 regret.

Why the rollout is better. The improvement comes from correcting the scope of the audit and completing its evidence chain. Unlike the live trajectory, the selected rollout remains focused on QFT, identifies the central backend-switching confound and connects the numerical claims, missing evidence, and theoretical explanation into a coherent fairness judgment. Offline expansion supplies the stronger trajectory, while self-reward supplies the selection mechanism that recovers it from the rollout pool.

Comment and takeaway. A weak live outcome can still supply useful criteria for judging new trajectories. This example illustrates both coverage gain and successful self-reward selection. The internal score of 0.90 is an offline diagnostic, not a replacement for the delivered live score of 0.30.