RSIE: RECURSIVE SELF-IMPROVEMENT THROUGH INTERACTION WITH EXPERT ENVIRONMENTS

Anonymous authors
Paper under double-blind review

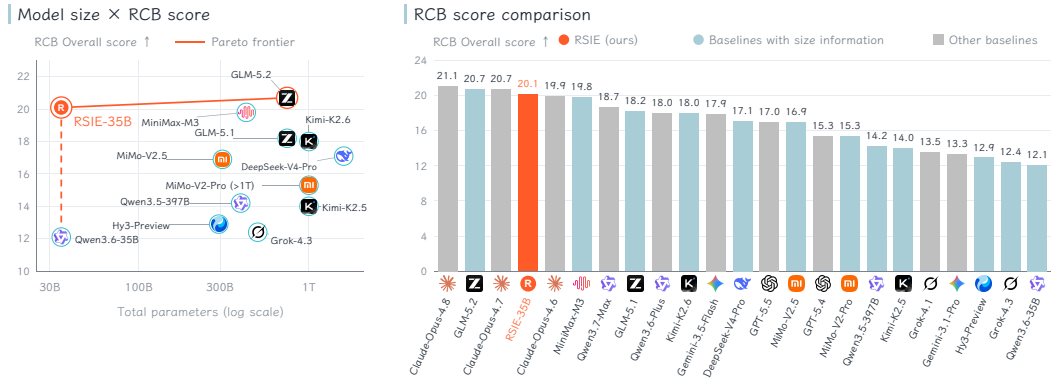


Figure 1: **Results on ResearchClawBench**.Left: score versus model size and Pareto frontier; right: score comparison with RSIE highlighted.

ABSTRACT

Recent work on recursive self-improvement (RSI) has demonstrated that agents can improve through repeated interaction with their tasks and environments. Yet it remains unclear whether such improvement can persist in expert environments, where executions are long-horizon and costly, and feedback is sparse, delayed, and domain-specific. Research and professional tasks naturally exhibit these characteristics, making them a demanding testbed for recursive self-improvement. We introduce **RSIE**, a system for recursive self-improvement through interaction with expert environments. RSIE repeatedly executes expert tasks, extracts reusable experience from completed trajectories and their evaluation feedback, and retains candidate experience only when its reuse leads to measurable improvement. Confirmed experience is accumulated through two complementary paths: externalized guidance for subsequent executions and parameter internalization via on-policy self-distillation. As the agent improves, saturated tasks are replaced by new ones, continually exposing the system to its evolving capability frontier. We evaluate RSIE with Qwen3.6-35B and Qwen3.5-9B on ResearchClawBench, SGIBench, and GDPVal, spanning scientific research and professional work. RSIE consistently improves over the corresponding base agents, showing that recursive self-improvement can extend to expert environments with sparse task-level feedback.

1 INTRODUCTION

Recursive self-improvement (RSI) studies how an agent can learn from its own interactions and improve its future behavior. The central idea is to turn experience collected during interaction into improvements that persist across subsequent tasks.

A growing body of work has explored this idea under different forms of feedback and evaluation. Some systems rely on automatically verifiable outcomes, such as exact answers, sampled agreement, or executable tests (Huang et al., 2026; Xia et al., 2025; Zhou et al., 2026). Others use benchmark scores or model-based evaluation to select which behaviors and experiences to preserve (Ouyang et al., 2026; Wang et al., 2024; Guo et al., 2026; Zhang et al., 2026a). Recent work also extends

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.

self-improvement to the surrounding agent program or harness (Fan & Huang, 2026; Zhang et al., 2026b). Despite these differences, most existing methods are designed for settings where feedback is frequent, inexpensive, and repeatable. These conditions support repeated trial and error, but leave open how an agent should improve when each execution is costly, meaningful evaluation arrives only after a complete deliverable, and the effects of early decisions are difficult to trace.

We study this question in expert environments derived from research and professional work. In these environments, an agent must carry out a domain-specific, long-horizon workflow—such as analyzing data, conducting experiments, using specialized tools, or producing a professional report or presentation—and is evaluated only after the resulting artifact is complete. Feedback is therefore sparse, delayed, and task-specific, while each interaction may be expensive. In our evaluation, ResearchClawBench and GDPVal provide examples of this setting through artifact-level or rubric-based assessment. We refer to this setting as recursive self-improvement in expert environments (RSE). This setting raises a basic question: can an agent improve from a small number of costly expert interactions, and can experience extracted from one execution improve later behavior?

We introduce **RSIE**, a system for recursive self-improvement in expert environments. It has three components. First, FrontierWorld constructs executable expert environments from completed research and professional deliverables, providing each environment with its own workspace and evaluator. Second, an experience extraction and admission mechanism analyzes completed executions into candidate experience and retains a candidate only when reusing it improves the score under the same evaluator. Third, confirmed experience is fed back through two complementary routes: it is externalized into the agent's harness to guide subsequent executions and internalized into the model parameters through on-policy self-distillation.

Our contributions are as follows:

- **A recursive self-improvement system for expert environments.** We introduce RSIE, which extends recursive self-improvement to costly, long-horizon tasks with sparse, domain-specific feedback by combining executable expert environments, reuse-based experience admission, and feedback to both the harness and the model parameters.
- **Executable environments derived from expert deliverables.** We introduce FrontierWorld, which constructs environments from published research, professional reports, and presentations rather than from problems the system poses to itself, so that each task is grounded in an existing deliverable and its associated evaluation criterion.
- **An empirical study of RSI in expert environments.** On ResearchClawBench, SGIBench, and GDPVal with Qwen3.6-35B and Qwen3.5-9B, we show that both base agents achieve performance comparable to mainstream open- and closed-source reference models across the expert-task benchmarks.

2 Related Work

Environment and Task Synthesis for Agent Learning. A growing line of work treats task and environment construction as part of the agent learning pipeline rather than as a fixed source of training data. NexForge (Zhao et al., 2026) compiles capability requirements into executable tasks together with the files, dependencies, and runtime configurations needed for execution, while CUA-Gym (Wang et al., 2026) jointly synthesizes task instructions, environment states, and verifiable reward functions. Agent-World (Dong et al., 2026) discovers real-world tool ecosystems to construct tasks with controllable difficulty, while GenEnv (Guo et al., 2025) and DreamGym (Chen et al., 2026b) adapt generated environments and curricula to the agent's evolving capabilities. RSIE extends this direction to expert research and professional tasks with costly execution and task-specific evaluation.

Recursive Self-Improvement. Existing RSI systems improve agents through multiple surfaces. R-Zero (Huang et al., 2026) and Agent0 (Xia et al., 2025) couple task generation with policy improvement, while ReasoningBank (Ouyang et al., 2026), AWM (Wang et al., 2024), WikiSkill (Tang et al., 2026), and KOPE (Chen et al., 2026a) accumulate reusable experience outside model parameters. PEAM (Guo et al., 2026) studies experience internalization, whereas DGM (Zhang et al., 2026a), HELIX (Fan & Huang, 2026), and Hyperagents (Zhang et al., 2026b) expand improvement to agent programs or runtime harnesses. Recent work further explores meta-level self-improvement

Table 1: Representative recursive self-improvement systems, ordered by the cost of a single execution.

Method	Improvement Mechanism				Feedback Regime	
	Task / World	Model Update	Experience Reuse	Verified Retention	Judged by	Cost-scale per run
R-Zero (2026)	✓	✓	✗	✗	Sampled agreement	Seconds
Agent0 (2025)	✓	✓	✗	✗	Executable test	Seconds
ReasoningBank (2026)	✗	✗	✓	✗	Executable test	Seconds
AWM (2024)	✗	✗	✓	✗	Benchmark score	Seconds
PEAM (2026)	✗	✓	✓	✗	Benchmark score	Seconds
DGM (2026a)	✗	✗	●	●	Benchmark score	Minutes
MetaRSI (2026)	●	✓	●	✗	Benchmark score	Minutes
Meta^n (2026)	✗	✗	●	✗	Benchmark score	Minutes
WikiSkill (2026)	✗	✗	✓	✗	Benchmark score	Minutes
KOPE (2026a)	✗	✗	✓	●	Measured outcome	Minutes
Dream-RSI (2026)	●	✗	✓	✗	Replayed outcome	Minutes
RSIAgent (2026)	●	✗	✓	✗	System's verifier	Minutes
Ours	●	✓	✓	✓	**External rubric**	**Hours**

and adaptive exploration (Tan et al., 2026; Kim et al., 2026; Zheng et al., 2026; Zhu et al., 2026).Table 1 summarizes representative systems by their improvement mechanisms and feedback regimes. It highlights the distinction between inexpensive, repeatable feedback and the costly, externally evaluated interactions targeted by RSIE. RSIE focuses on how these ideas can operate under costly expert-task interaction with externally evaluated experience.

Agents for Expert Research and Professional Work. A parallel line of work pushes agents toward increasingly realistic scientific and professional tasks. The AI Scientist and AI Scientist-v2 (Lu et al., 2024; Yamada et al., 2025) automate substantial parts of the scientific workflow, while ResearchClawBench (Xu et al., 2026), SGI-Bench (Xu et al., 2025), and SciAgentArena (Liu et al., 2026) evaluate agents on expert-level scientific tasks. GDPval (Patwardhan et al., 2026) extends this setting to economically valuable professional work. RSIE uses such expert tasks not only for evaluation, but as the substrate for recursive self-improvement.

3 RSIE: Recursive Self-Improvement in Expert Environments

We formulate RSIE as a closed loop over a stream of expert environments. At cycle t, the system maintains an agent A_t, an expert-task pool $\mathcal{T}_t$, an active task set $\mathcal{C}_t \subseteq \mathcal{T}_t$, and an experience memory $\mathcal{M}_t$. Each task $\tau \in \mathcal{T}_t$ provides a task-specific workspace, supporting assets and tools, and a native evaluator E_τ. The pool capacity and the per-cycle active-task budget are fixed, while the task identities and active set may change across cycles.

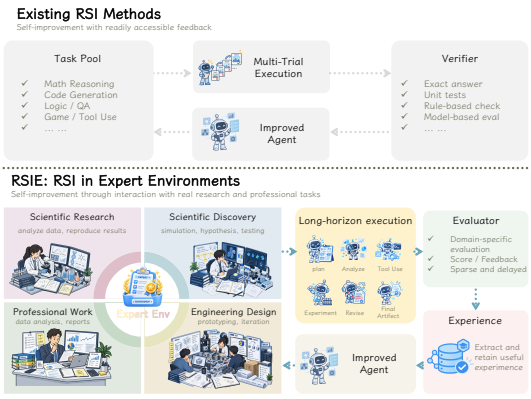


Figure 2: **From readily verifiable tasks to expert environments.** Existing RSI typically relies on frequent, inexpensive feedback, whereas expert tasks involve costly long-horizon execution and sparse, domain-specific evaluation.

3.1 System Overview

Figure 3 summarizes the closed loop. FrontierWorld supplies executable research and professional tasks through the expert-task pool. During each cycle, A_t executes the active tasks and produces trajectories and artifacts that are scored by their native evaluators. The resulting

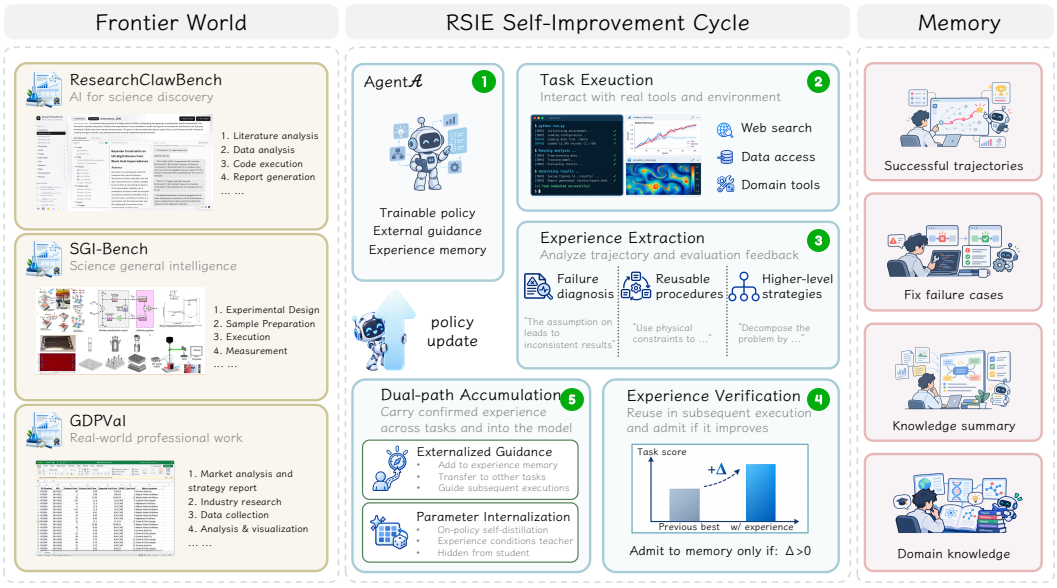


Figure 3: Overview of RSIE. RSIE repeatedly executes real research and professional tasks, extracts and verifies useful experience from completed executions, and accumulates confirmed experience through externalized guidance and parameter internalization. As the agent improves, saturated tasks are replaced with new ones.

evidence is used to form and verify candidate experience. Verified experience is accumulated in $\mathcal{M}_t$ and reused through two paths: it provides guidance during subsequent executions and supplies teacher-side information for the parametric update of the agent. The updated agent then provides the evidence used by FrontierWorld to refresh the task frontier.

The resulting transition is

$$(A_t, \mathcal{T}_t, \mathcal{M}_t) \xrightarrow{\text{execute and evaluate}} \mathcal{E}_t \xrightarrow{\text{verify and accumulate}} (A_{t+1}, \mathcal{T}_{t+1}, \mathcal{M}_{t+1}), \tag{1}$$

where $\mathcal{E}_t$ denotes the execution evidence collected during cycle t. Externalized experience remains available through $\mathcal{M}_t$, whereas parametric accumulation changes the agent parameters and can therefore be evaluated without the experience memory in context. The following sections describe the three components of this transition.

3.2 FrontierWorld: Expert Task Frontier

FrontierWorld provides the expert environments used by RSIE. Each environment is represented as

$$e = (x_e, W_e, R_e), \tag{2}$$

where x_e is the task instruction, W_e is the workspace together with its supporting assets and tools, and R_e is a task-specific evaluator that scores the submitted artifact and records the corresponding evaluation feedback. The evaluator and any reference material used to construct it remain hidden from the agent.

The environments are derived from completed research and professional deliverables—published studies that release the data and code behind a reported result, and occupational work products such as clinical notes, financial models, and engineering reports. Finished work fixes three properties a self-posed problem cannot guarantee: the task is solvable, the workspace holds real artifacts, and the reported outcome supplies a target for a hidden criterion. No stage of the three offline steps below queries the agent, so no property of a task depends on it.

Step 1: source candidate deliverables. From the semantics of existing seed tasks, FrontierWorld issues retrieval queries, deduplicates by document identity, screens titles and abstracts, and parses the survivors for links to released data and code. A final ranking scores task formability, artifact completeness, and checkability of the reported outcome; the three are necessary rather than compensatory, so one cannot offset another.

Step 2: construct and register the package. A construction agent writes x_e, assembles W_e from the released artifacts and supporting literature, and derives R_e from the outcome the deliverable itself reports. Only the public subset—instruction, data, literature—enters the execution container, so the agent is scored against the reported outcome but never reads it. A static pass then checks this contract—file structure, artifact presence, criterion weighting, and the public/hidden boundary—though it runs no rollout and so certifies the contract rather than solvability. Admitted packages enter an immutable, content-addressed pool.

Step 3: refresh the frontier. Each task is assigned one of four states: *reserve*, *active*, *mastered*, or *revisit*. A task enters *mastered* when it reaches the benchmark-defined maximum score, and enters *revisit* after three consecutive valid evaluations without a strict improvement over its own best score. These transitions are task-local: they depend only on the task's evaluation history under its unchanged evaluator and never on comparisons or rankings across tasks. Whenever an active task leaves the frontier, its slot is filled by a package registered before the run begins. The active set therefore evolves across cycles, and its score is not used as a direct measure of progress; all reported performance numbers are obtained on held-out tasks.

Appendix A.3 details the offline task-construction pipeline and frontier-refresh policy.

3.3 Experience Memory: Extraction, Verification, and Reuse

For each completed execution, RSIE analyzes the trajectory, produced artifact, and evaluator feedback to extract candidate experience. A candidate may capture a reusable procedure, a failure and its repair, or a domain-specific insight. Candidates remain pending until they are tested in a subsequent execution of the same task, separating experience proposal from experience admission.

Let ℓ_e^t denote the experience loaded when task e is executed at cycle t, and let b_e^{t-1} be the best valid score previously obtained on that task. The experience loaded at cycle t is retained only when its reuse improves the task score:

$$k_e^t = \begin{cases} \ell_e^t, & \text{if } s_e^t > b_e^{t-1}, \\ k_e^{t-1}, & \text{otherwise,} \end{cases} \tag{3}$$

where s_e^t is returned by the unchanged task evaluator. Invalid executions caused by infrastructure failures are excluded from this comparison. A candidate written in one cycle is therefore evaluated in a later cycle, and only verified candidates enter $\mathcal{M}_t$ or the parametric update.

Verified experience supports two forms of reuse. During externalized reuse, relevant entries from $\mathcal{M}_t$ are retrieved to guide subsequent task execution; entries can therefore be tested beyond the task from which they were extracted. During parametric reuse, the verified experience is available to the teacher during on-policy distillation, while the student receives the ordinary task context. This provides a route for evaluating whether the behavior survives after the explicit experience is removed. The detail in appendix A.5

3.4 Recursive Self-Improvement Cycle

At cycle t, RSIE executes A_t on the active tasks in $\mathcal{C}_t$. The trajectories, artifacts, and evaluator feedback form $\mathcal{E}_t$, from which candidate experience is extracted and subsequently verified. Let $\mathcal{Z}_t$ denote the candidates that pass this verification. The memory and the agent are then updated as

$$\mathcal{M}_{t+1} = \mathrm{UpdateMemory}(\mathcal{M}_t, \mathcal{Z}_t), \tag{4}$$

$$A_{t+1} = \mathrm{UpdateModel}(A_t, \mathcal{Z}_t). \tag{5}$$

The updated agent is used to refresh the task frontier:

$$(\mathcal{T}_{t+1}, \mathcal{C}_{t+1}) = \mathrm{RefreshFrontier}(A_{t+1}, \mathcal{T}_t, \mathcal{E}_t), \tag{6}$$

where FrontierWorld supplies task packages for released slots. Repeating this transition couples what the agent learns with what it encounters next: verified experience changes the agent and its memory, while the resulting execution evidence changes the subsequent expert-task pool.

Additional implementation details for these components, including task-generation, experience representation, verification protocol and parameter-update settings are provided in the appendix A.5

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate RSIE on three expert-task benchmarks covering scientific research, scientist-aligned workflows, and professional deliverables. All in-house runs use *ResearchHarness* as the common execution harness. We preserve each benchmark's native task format, rubric, and score aggregation: ResearchClawBench uses checklist scoring, SGIBench reports separate scores for its DeepResearch and wet-experiment subsets, and GDPVal uses item-level pairwise comparisons for professional deliverables. Scores are compared within each benchmark and are not averaged across benchmarks.

We initialize all runs from two Qwen base models at two scales, denoted by Qwen3.6-35B and Qwen3.5-9B, and run each system for ten cycles. The active pools contain 600 tasks for ResearchClawBench, 5,000 high-quality generated tasks for SGIBench, and 1,000 active tasks for GDPVal throughout the experiment. Every task is executed through *ResearchHarness* with 4 CPU cores and 8 GB of memory. ResearchClawBench uses long-horizon task budget of two hours, SGIBench uses a maximum 2-hour task budget, and GDPVal uses a maximum 4-hour agent timeout, a 10-minute limit for each model call, and at most 250 interaction steps over its evaluation set. For LLM-judged components, we use GPT-5.4: ResearchClawBench applies it as a checklist judge, SGIBench (DeepResearch) combines it with exact-match checks, and GDPVal uses Elo ratings fitted from pairwise win/loss/tie outcomes as its headline metric.. The SGIBench wet-experiment subset uses its deterministic action rubric. GDPVal reports continuous scores together with win/loss/tie outcomes and Elo ratings. Unless explicitly enabled, experience retrieval is disabled, allowing us to distinguish improvement retained in the model parameters from gains that depend on retrieved experience. Detailed protocol tables and judge settings are provided in Appendix A.2.

4.2 MAIN RESULTS

Table 2 summarizes reference-model results together with runs initialized from the two Qwen base agents. The reference models provide context for the performance range on each expert-task benchmark. The experimental rows compare SFT, DPO, recursive improvement without experience retrieval, and the complete RSIE configuration. The two SGIBench subsets are reported separately because they use different evaluators. The GDPVal column reports Elo ratings, whereas ResearchClawBench and SGIBench retain their native scores.

For the smaller base agent, SFT decreases the RCB score from 8.83 to 6.02 and the SGI wet-experiment score from 17.80% to 16.34%. DPO provides a stronger training control, reaching 9.22 on RCB, 33.97% on SGI wet experiments, 47.20% on SGI (DeepResearch), and 744 Elo on GDPVal. Recursive improvement without experience retrieval reaches 13.1 on RCB, 46.32% on SGI (DeepResearch), and 713 Elo on GDPVal. Enabling experience retrieval further improves these results to 14.9, 48.56%, and 825 Elo, respectively.

For the larger base agent, SFT provides a modest improvement over the base agent, whereas recursive improvement without experience retrieval reaches 19.2 on RCB, 55.70% on SGI (DeepResearch), and 1153 Elo on GDPVal. The complete RSIE configuration reaches 20.1, 59.50%, and 1339 Elo, respectively. The gains across scientific research, deep research, and professional deliverables show that interaction with expert tasks can produce improvements that transfer across task types and evaluation protocols. Experience retrieval provides a further, consistent benefit beyond the parametric improvement obtained through recursive interaction.

Figure 4 reports the best score observed up to each improvement cycle for ResearchClawBench, SGIBench (DeepResearch), and GDPVal. The curves are obtained through self-improvement without additional parameter training. Short-horizon tasks are often resolved in early cycles, whereas more demanding tasks continue to improve over multiple cycles. When a later raw score decreases, the plotted curve remains at the previous best, separating durable progress from cycle-level evaluation variation. Across the three benchmarks, the resulting trajectories show how recursive interaction accumulates improvements over time rather than relying on a single successful cycle.

Table 2: **Main results on expert-task benchmarks.** Reference models are included for context. In-house rows compare base agents, SFT, DPO, recursive improvement without experience, and the complete RSIE configuration. The two SGIBench subsets are reported separately because their evaluators differ. ResearchClawBench reports its native checklist score, SGIBench reports percentages, and GDPVal reports Elo ratings.

Method	Research ClawBench	SGIBench		GDPVal
		Wet experiments	(DeepResearch)	
Closed-source				
Claude Opus 4.8	21.1	67.83	51.57	1490
Claude Opus 4.6	19.9	58.23	50.21	1220
GPT-5.5	17.0	52.81	59.12	1336
Gemini 3.5 Flash	17.9	41.58	46.54	1184
Qwen3.7-Max	18.7	50.62	51.89	1114
Grok-4.3	13.2	52.88	38.36	922
Open-source				
Kimi K2.6	18.0	50.21	38.36	1026
DeepSeek-V4-Pro	17.1	60.21	50.94	1374
MiMo-V2.5	16.9	61.92	23.58	986
Qwen3.5-9B				
Base agent	8.83	17.80	44.97	608
SFT	6.02	16.34	46.23	600
DPO	9.22	33.97	47.20	744
RSIE (w/o experience)	13.1	41.20	46.32	713
RSIE	**14.9**	**48.80**	**48.56**	**825**
Qwen3.6-35B				
Base agent	12.1	23.37	48.43	838
SFT	13.4	28.87	50.63	866
RSIE (w/o experience)	19.2	52.21	55.70	1153
RSIE	**20.1**	**56.90**	**59.50**	**1339**

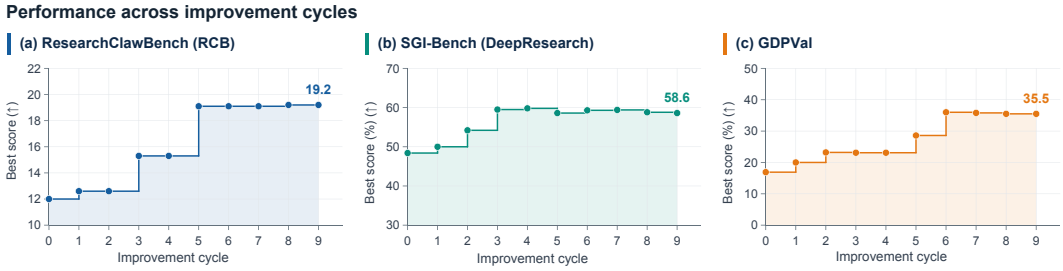


Figure 4: **Performance across improvement cycles.** Panels show the best score up to each cycle. Curves use diagnostic skill self-improvement without parameter training. GDPVal reports LLM-judge scores in there.whereas all other GDPVal results reports Elo ratings.

4.3 The Dynamics of Recursive Improvement in Expert Environments

Section 4.2 shows that RSIE improves over repeated cycles on expert tasks. We now characterize the dynamics underlying this improvement through three questions: why do fixed tasks eventually saturate, how does experience organization affect transfer, and does progressive exposure to new expert environments outperform one-shot injection?

Unless otherwise stated, the analyses in this section are conducted on ResearchClawBench, whose detailed execution traces make it possible to inspect how behavior, task outcomes, and experience change across improvement cycles. We also evaluate conventional supervised fine-tuning on ResearchClawBench as an offline control; the complete training setup, data-selection procedure, trajectory reconstruction, and quality-control process are described in Appendix A.6. Even after sub-

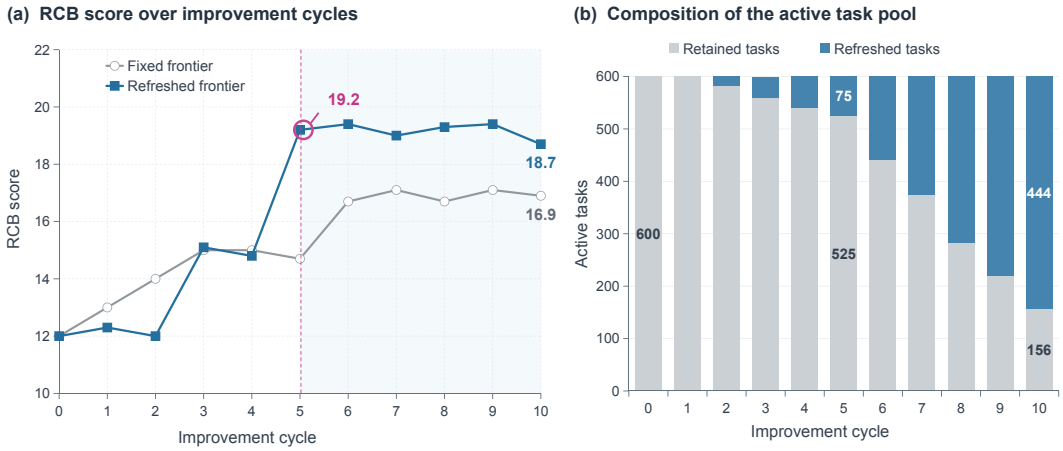


Figure 5: **Task-level overfitting and frontier refresh.** (a) ResearchClawBench scores on fixed and refreshed task frontiers over ten improvement cycles. (b) Composition of the active task pool during frontier refresh. The active pool contains 600 tasks at each cycle; blue denotes refreshed tasks.

Table 3: **Ablation results.** RCB score and skill overlap for RSIE and its variants.

Method	Skill overlap (%) ↓	RCB score ↑
RSIE	56.7	**19.2**
w/o skill categorization	87.0	18.4
w/ checklist	56.7	18.7
w/ one-shot injection	56.7	16.9

stantial filtering and quality control, SFT produces only a limited improvement under the controlled data and compute budget used here. This result does not rule out gains from larger datasets or alternative curation strategies, but it illustrates the difficulty of acquiring expert-level task competence from offline replay alone. We therefore analyze how recursive interaction with an evolving task frontier supplies additional learning signals over successive cycles.

4.3.1 Task-Level Overfitting and Frontier Refresh

Fixed tasks saturate and induce task-level overfitting. Figure 5(a) reports the ResearchClawBench score (without experience retrieval) over ten improvement cycles under a fixed task frontier and a refreshed frontier. Performance on the fixed frontier improves early and then remains within a narrow range. With frontier refresh, the score reaches 19.2 at cycle 5, the best checkpoint of the run, and remains above the fixed-frontier trajectory in the later cycles; the cycle-10 score is 18.7. Figure 5(b) shows the corresponding active task pool. The pool remains fixed at 600 tasks, while saturated tasks are progressively replaced; by cycle 10, 444 active slots have been refreshed. Results are consistent with task-level overfitting: repeated optimization of the same task identities eventually provides limited new signal, whereas changing the frontier exposes agent to new requirements.

4.3.2 Experience Organization for Transfer

Experience organization shapes transfer. Table 3 compares the full system with variants on ResearchClawBench. Based on recurring patterns in the extracted trajectories, we organize experience into four functional categories: task-specific knowledge, domain skill, general workflow, and failure avoidance. Domain skill and general workflow are internalized, while task-specific knowledge and failure avoidance remain externally retrievable. The full system has 56.7% skill overlap and scores 19.2. Removing skill categorization increases overlap to 87.0% and lowers the score to 18.4. The checklist variant keeps overlap at 56.7% but scores 18.7, consistent with checklist-driven task-level fitting rather than improved transfer. The comparison indicates that organizing experience by transfer scope is important for the observed gain.

4.3.3 PROGRESSIVE EXPOSURE TO NEW EXPERT ENVIRONMENTS

Progressive exposure outperforms one-shot injection. We compare two schedules that use the same verified experience collection and the same total data volume; they differ only in when the collection is introduced. Because the collection, evaluation protocol, and total data budget are matched, the comparison isolates the effect of exposure schedule. Progressively introducing the collection across improvement cycles reaches an ResearchClawBench score of 19.2, whereas injecting the complete collection in a single step reaches 16.9. The progressive result is the best checkpoint at cycle 5, while the corresponding refreshed run scores 18.7 at cycle 10. This comparison indicates that improvement depends on matching experience to the agent's evolving capabilities: each cycle provides an opportunity to apply newly available experience, evaluate its effect, and update the agent before the next increment is introduced. One-shot injection removes this staged interaction and the intermediate opportunities for feedback, even when the total amount of experience is held fixed.

Taken together, these analyses identify two complementary conditions for recursive improvement in expert environments. Frontier refresh introduces new requirements once familiar tasks cease to provide a useful learning signal, while structured and incremental experience accumulation determines how effectively the agent can reuse what it has learned across subsequent expert interactions. This separation also clarifies why continued improvement depends on both the novelty of the task frontier and the timing of experience exposure.

4.4 SCALING THE EXPERT TASK FRONTIER

We examine how expert-task data volume affects recursive improvement on ResearchClawBench. Table 4 reports the measured scores for the two base-model sizes. Both models improve as the data budget increases, but their marginal gains differ at the upper end of the measured range: increasing the budget from 400 to 600 examples improves Qwen3.6-35B from 17.8 to 19.2(without experience retrieval), while Qwen3.5-9B improves from 12.71 to 13.10.

The final column isolates this last measured increment: Qwen3.6-35B gains 1.40 points, whereas Qwen3.5-9B gains 0.39 points. The result is consistent with a model-scale-dependent data frontier: the larger model continues to benefit from additional expert-task data over the measured range, while the smaller model exhibits weaker late-stage gains. With only four measured budgets, we report the exact scores without fitting a scaling law or treating an extrapolated saturation point as evidence.

Table 4: **Data scaling on ResearchClawBench.** Measured RCB scores as the expert-task data budget increases for Qwen3.6-35B and Qwen3.5-9B. The final column reports the score change from $N = 400$ to $N = 600$.

Base model	$N = 64$	$N = 140$	$N = 400$	$N = 600$	$\Delta_{400\to600}$
Qwen3.6-35B	14.20	16.10	17.80	**19.20**	**+1.40**
Qwen3.5-9B	8.97	10.56	12.71	**13.10**	+0.39

5 CONCLUSION

We study recursive self-improvement in expert environments, where tasks require long-horizon execution and provide sparse, delayed, and domain-specific feedback. We introduce RSIE, a closed-loop system that combines executable task frontiers from FrontierWorld, verified experience reuse through ResearchHarness, and parametric internalization through on-policy self-distillation. Across ResearchClawBench, SGI-Bench, and GDPVal, RSIE consistently improves over the corresponding base agents and offline SFT controls.

Our analyses show that continued improvement depends on refreshing saturated tasks and organizing experience for progressive transfer across new expert environments. Together, these results identify evolving task frontiers and verified experience as complementary substrates for recursive improvement on increasingly diverse real-world work.

AI USE STATEMENT

We used generative AI tools to support several parts of the research and writing process, including polishing and drafting manuscript text, searching for related work and background material, constructing synthetic datasets and task resources, and checking mathematical derivations and proofs. These tools were used as assistants within these workflows and were not used to determine the research claims, independently design the experiments, or produce the reported results. All AI-assisted materials, data, and mathematical statements were reviewed and revised by the authors, who take responsibility for the final manuscript, claims, results, and released artifacts.

REPRODUCIBILITY STATEMENT

We will release the main components of RSIE, including the recursive improvement loop, FrontierWorld task construction, experience representation and reuse, execution and evaluation adapters, model checkpoints, and reproduction configurations. Detailed benchmark, frontier, execution, experience, and SFT settings are provided in Appendices A.2–A.6. All benchmarks retain their native task formats and evaluators, which we neither re-implement nor substitute.

REFERENCES

Siyuan Chen, Runlin Hou, Shenxiu Wu, Yansong Sun, Junming Cao, Yiyu Zhang, Shudi Shao, Junhao Qiu, Zhichao Lu, and Qingfu Zhang. Beyond scaling: Self-evolving llm agents for hardware kernel optimization via an experience-driven workflow and experience graph memory. *arXiv preprint arXiv:2608.25570*, 2026a.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Xuefei Cao, Yuanhao Xiong, Haibo Tong, et al. Scaling agent learning via experience synthesis. In *International Conference on Learning Representations*, volume 2026, pp. 121394–121420, 2026b.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agentworld: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

Tianyu Fan and Chao Huang. Helix: Model-harness co-evolution for recursive self-improvement. *arXiv preprint arXiv:2608.13951*, 2026.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. Genenv: Difficulty-aligned co-evolution between llm agents and environment simulators. *arXiv preprint arXiv:2512.19682*, 2025.

Yuchen Guo, Junli Gong, Weicheng Wang, Hongmin Cai, Yiu-ming Cheung, and Weifeng Su. Peam: Parametric embodied agent memory through contrastive internalization of experience in minecraft. *arXiv preprint arXiv:2605.27762*, 2026.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026.

Zae Myung Kim, Young-Jun Lee, Seungyeon Jwa, and Dongyeop Kang. Meta^n: Recursive self-improvement through emergent depth. *arXiv preprint arXiv:2608.24735*, 2026.

Tianyu Liu, Allen Xin Wang, Antonia Panescu, Lisa Xinyi Chen, Wenxin Long, Xinyu Wei, Yueqian Jing, Ziyao Zeng, Jihang Chen, Sihan Jiang, et al. Benchmarking ai agents for addressing scientific challenges across scales. *arXiv preprint arXiv:2606.12736*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simon Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. Gdpval: Evaluating ai model performance on real-world economically valuable tasks. In *International Conference on Learning Representations*, volume 2026, pp. 24005–24040, 2026.

Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, et al. Metarsi/rsi2: A meta-recursive self-improving system for recursive self-improving systems themselves. *arXiv preprint arXiv:2609.06396*, 2026.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. Wikiskill: Compiling agent experience into persistent knowledge for skill evolution. *arXiv preprint arXiv:2608.27454*, 2026.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents. *arXiv preprint arXiv:2605.25624*, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Wanghan Xu, Yuhao Zhou, Yifan Zhou, Qinglong Cao, Shuo Li, Jia Bu, Bo Liu, Yixin Chen, Xuming He, Xiangyu Zhao, et al. Probing scientific general intelligence of llms with scientist-aligned workflows. *arXiv preprint arXiv:2512.16969*, 2025.

Wanghan Xu, Shuo Li, Tianlin Ye, Qinglong Cao, Yixin Chen, Hengjian Gao, Yiheng Wang, Qi Li, Kun Li, Sheng Xu, et al. Researchclawbench: A benchmark for end-to-end autonomous scientific research. *arXiv preprint arXiv:2606.07591*, 2026.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026a.

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

Jiarong Zhao, Zhikai Lei, Zhiheng Xi, Rui Zheng, Hang Yan, Jie Zhou, Qin Chen, and Liang He. Nexforge: Scaling executable agent tasks via requirement-first synthesis. *arXiv preprint arXiv:2607.14186*, 2026.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, et al. Dream-rsi: Recursive self-improvement through evolving worlds. *arXiv preprint arXiv:2609.14858*, 2026.

Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. *Advances in Neural Information Processing Systems*, 38:113959–113991, 2026.

Sibo Zhu, Shicheng Fan, Xinyue Wang, Wenyi Wu, Kun Zhou, and Biwei Huang. Rsiagent: Autonomous exploration for recursive self-improvement in new environments. *arXiv preprint arXiv:2609.15364*, 2026.

A APPENDIX

A.1 OVERVIEW

This Appendix is structured as follows:

- Sec. A.2: Details of the three expert-task benchmarks.
- Sec. A.3: Task construction and frontier refresh in FrontierWorld.
- Sec. A.4: Details of task execution, covering the sandbox and the evaluation protocol.
- Sec. A.5: Experience representation and the two reuse paths, externalization into ExpertHarness and internalization by on-policy self-distillation.
- Sec. A.6: Supervised fine-tuning on ResearchClawBench.

A.2 EXPERT-TASK BENCHMARKS

We evaluate on three benchmarks that differ in domain, deliverable, and evaluation protocol. Each retains its native task format and its native scorer; we do not re-implement or substitute an evaluator, and scores are never averaged across benchmarks. Table 5 summarizes the three settings.

Table 5: **Expert-task benchmarks used for evaluation.** Each benchmark keeps its native deliverable, evaluator, and score scale. SGIBench is reported as two subsets because their evaluators differ.

Benchmark	Eval tasks	Deliverable	Evaluator
ResearchClawBench	40	Analysis code and a scientific report reproducing a study	Hidden per-task checklist, scored item by item
SGIBench (DeepResearch)	318	Answer to a research question with supporting steps	Exact match and an LLM judge over the released reference
SGIBench (Wet experiments)	68	Ordered experimental action sequence	Deterministic rubric over the extracted action sequence
GDPVal	220	Professional artifact for an occupational task	GPT-5.4 pairwise partial-order judge under the task rubric

Common execution settings. Each task runs in an isolated container with 4 CPU cores and 8 GB of memory.The runner records the complete request/response trajectory, tool calls, and produced artifacts before invoking the evaluator outside the container. A run is valid only when the task adapter completes and returns a score under its evaluator contract. Timeouts, worker failures, and contract failures are recorded as invalid executions and cannot promote experience or enter a parametric update.

ResearchHarness. All three benchmarks are executed through *ResearchHarness*, our common task runner. For each task, ResearchHarness materializes the released workspace and declared tools in an isolated container, launches the current agent checkpoint, records model requests and responses, tool calls, and produced artifacts, and exports the final artifacts to the benchmark-specific evaluator. Evaluator-only assets remain outside the container. Benchmark-specific adapters preserve each benchmark's native task interface, resource limits, evaluator, and score scale; invalid executions are excluded from comparisons and learning updates as described in Appendix A.4.

ResearchClawBench The agent reads the released task material, performs the analysis, and exports the requested scientific artifacts. The target study, checklist, and other evaluator-only assets remain hidden. A GPT-5.4 checklist evaluator scores the submitted report and code item by item using the benchmark's native weights. We report the resulting native checklist score rather than converting it to a percentage.

SGIBench covers scientist-aligned workflows and is reported as two subsets. The DeepResearch subset asks the agent to answer a research question by gathering and synthesizing evidence, and is

scored by exact match together with an LLM judge. The wet-experiment subset asks for an experimental action sequence and is scored by a deterministic rubric applied to the actions extracted from the trajectory, with no judge model in the loop. Because the two subsets use different evaluators, we report them as separate columns throughout and never average them.

GDPVal evaluates the production of real professional deliverables across occupations. The evaluation set contains 220 questions, and each task is executed through ResearchHarness under the same resource and timeout configuration. Submitted artifacts are compared pairwise under the task rubric. GPT-5.4 acts as a pairwise partial-order judge and records win/loss/tie outcomes. The headline GDPVal metric is an Elo rating fitted from all pairwise outcomes. For Figure 4 only, we additionally use the continuous rubric scores produced by the judge to plot performance across improvement cycles; this auxiliary score is not the metric reported in Table 2.

DPO baseline. We train DPO only for Qwen3.5-9B as an additional learning baseline. For each training task, the current policy samples candidate trajectories in ResearchHarness. The benchmark evaluator or GPT-5.4 judge ranks the trajectories, and we retain only strict preference pairs (y^+, y^-); ties, invalid executions, and missing artifacts are discarded. Evaluation tasks are excluded from rollout and preference construction. The DPO reference policy is the corresponding frozen base agent, and all rollouts use the same task interface and resource limits as the other baselines.

The evaluation task identities of all three benchmarks are held out from every part of the loop that can change the agent. They are never executed as active tasks, never used to extract experience, and never exported into a training dataset. For ResearchClawBench, the deliverables behind evaluation tasks are additionally removed from the FrontierWorld sourcing queue, so that a held-out study cannot reenter the task pool through construction. For SGIBench, the training pool is drawn from a disjoint corpus and its overlap with the evaluation questions was verified to be empty under normalized-text matching. For GDPVal, the training tasks and the evaluation manifest are disjoint by task identifier.

A.3 Task Construction and Frontier Refresh in FrontierWorld

FrontierWorld produces the executable expert environments consumed by the loop. The pipeline is offline: it does not query the agent while constructing a task, so no property of a constructed task depends on how the agent currently performs.

Construction starts from completed deliverables rather than from problems the system poses to itself. Existing seed tasks recover the domains and task semantics of interest, from which retrieval queries are generated at several granularities. Candidate deliverables are retrieved from public repositories through multiple channels and deduplicated by document identity. Titles, abstracts, and rule-based signals then screen for research value, for whether a well-posed task could be derived, and for preliminary execution feasibility; retained candidates are downloaded, parsed, and mined for links to released data and code. The parsed body and the released artifacts are scored on three properties—whether a task can be formed, how complete the released assets are, and whether the reported outcome can be checked—and because each property is necessary, a candidate must clear a threshold on all three. Domain quotas and these scores produce the target pool that task construction consumes. The interface between sourcing and construction is the selected pool alone, so retrieval channels can change without affecting downstream stages, and any deliverable behind a benchmark evaluation task is removed before selection.

A construction agent then turns each selected deliverable into an executable package. It writes the task instruction, assembles the workspace from the released data, code, and supporting literature, derives the evaluation criterion from the outcome the deliverable itself reports, and records a domain and difficulty label. The package separates what the agent may see—the task instruction, the data directory, and the supporting literature—from what the evaluator uses: the source deliverable, the reference solution, the evaluation checklist, and any evaluator-only assets. This split is what lets the criterion be demanding without being visible: the agent is scored against the outcome the original deliverable reports, but never reads it.

Before registration, a static validation pass checks the directory and file structure, the presence of the referenced data and documents, the structure and weighting of the evaluation criterion, the public/hidden boundary, the construction metadata and input fingerprints, and the stability of the task identity across re-runs. We state its scope precisely: validation certifies that the package satisfies the

task contract. It does not execute a solver rollout, and therefore does not certify that a future agent can complete the task, nor that the evaluator has been exercised end to end. Packages are registered through a content-addressed release with a manifest, content hashes, and recorded provenance linking each task back to the seed and query that surfaced its source deliverable.

The registered pool is maintained at a fixed capacity, and each task carries one of four states: *reserve*, *active*, *mastered*, and *revisit*. The refresh policy is deterministic. A task whose score reaches the mastery threshold becomes *mastered* and leaves the active set; a task that fails to strictly improve its best score on $n = 3$ consecutive valid evaluations becomes *revisit* and also leaves. A released slot is filled from the reserve pool, preferring a task with the same benchmark adapter, domain, and difficulty label, with ties broken by a fixed run seed. Refreshing a slot does not invoke a language model and does not construct a task package online; replacements are drawn from packages registered before the run. Because the active set is not a fixed population, scores measured on it are not a measure of progress, and every number we report comes from held-out evaluation tasks.

A.4 Task Execution

Each task runs in an isolated container built for that task. The runner materializes the released workspace, mounts the tools the task declares, and starts the agent against a served checkpoint; evaluator-side material never enters the container. The container's job is to produce artifacts, not to score them: when the agent finishes, the workspace contents and the final submission are exported out of the sandbox, and judging happens outside it against the benchmark's own evaluator. An execution is archived together with its trajectory and the raw request and response records of the underlying model calls.

Budgets are fixed per benchmark and identical across the systems compared within that benchmark. Every container is provisioned with 4 CPU cores and 8 GB of memory, and an agent is allowed roughly ten interaction rounds per task. The remaining settings differ by how long a benchmark's tasks take, and are listed in Table 6: ResearchClawBench tasks run for hours and are executed at a concurrency of 8, whereas GDPVal and SGIBench tasks complete in 30 and 15 minutes and run at concurrencies of 4 and 2.

Table 6: **Execution settings.** All benchmarks share the same container size and interaction budget; wall-clock scale and concurrency differ.

Setting	**Research ClawBench**	**GDPVal**	**SGIBench**
Container	4C / 8G	4C / 8G	4C / 8G
Interaction rounds	10	10	10
Time per task	Hours	30 min	15 min
Task concurrency	8	4	2

An execution is recorded as valid only when the benchmark evaluator ran to completion and returned a score through its own contract. Executions that end in a timeout, a worker crash, or a contract failure are recorded as invalid. Invalid records are excluded from the comparison in Eq. 3, cannot promote a candidate experience, cannot enter the parametric update, and cannot advance the task state used by the frontier policy, so that infrastructure failures are not read as evidence about the agent or about a candidate. One limitation of this filter is worth stating: a record can satisfy the evaluator contract and still be uninformative, for example when an entire run returns zero for a reason external to the agent. We therefore inspect the distribution of zero scores within a run rather than relying on the validity flag alone.

Scoring is delegated to each benchmark's native evaluator, invoked through a per-benchmark adapter. The loop requires only that the adapter return one bounded number, which it uses for the admission comparison and the frontier policy; it does not convert the three benchmarks to a common rubric. ResearchClawBench scores the submitted report and code against the task's hidden checklist, item by item, and returns the weighted checklist score. SGIBench (DeepResearch) scores the submitted answer by exact match and by an LLM judge against the released reference; we report these as the primary signals and do not use the step-level judge as a headline metric, because its parse

behaviour is unreliable on part of the set. SGIBench (wet experiments) extracts the action sequence from the trajectory and applies a deterministic rubric, with no judge model involved. GDPVal parses each submission into the required artifacts and compares systems pairwise under the task rubric using GPT-5.4. Each comparison produces a partial-order outcome, recorded as a win, loss, or tie. The final benchmark score is an Elo rating fitted from all pairwise outcomes. For Figure 4 only, we additionally aggregate the continuous rubric scores from the pairwise judge across evaluation tasks; this continuous score is used only for the cycle-level visualization and is not used as the headline GDPVal metric.

The models that sit outside the improvement loop are listed in Table 7. All of them are GPT-5.4, and none is updated by RSIE: the only parameters the loop changes are those of the agent. Because the diagnoser and the judges are instances of the same underlying model, an evaluation signal and the analysis of that signal share a model, and we report this rather than treating the two as independent.

Table 7: **Fixed models outside the improvement loop.** None of these models is updated by RSIE.

Role	Model
Diagnoser (experience extraction)	GPT-5.4
ResearchClawBench checklist evaluator	GPT-5.4
SGIBench (DeepResearch) judge	GPT-5.4
SGIBench (wet experiments)	Deterministic rubric

A.5 Experience Representation and Reuse

A.5.1 Representation and Extraction

An experience entry is a single record that can be injected directly into the agent's context. It is written under a fixed schema with four fields: a **trigger** stating when the entry applies, a **procedure** stating what to do, a **validation** stating how to check that the procedure worked, and a **repair** stating what to do when the check fails. The same record is additionally indexed into four functional views, which Sec. 3.3 treats differently when the entry is reused: k^{ts}, **task-specific knowledge**, holding facts about this deliverable, dataset, or task; k^{ds}, **domain skill**, a reusable method of the domain with instance values removed; k^{gw}, **general workflow**, a cross-task procedure for execution, verification, and artifact organization; and k^{fa}, **failure avoidance**, a guard derived from an observed failure.

The four views are structured projections of one entry rather than four independently validated entries: admission (Eq. 3) tests the entry as a whole, and we do not attribute the resulting score change to an individual view. Section 4.3.2 compares organized against unorganized experience; it does not isolate the contribution of a single view, which would require a separate per-view rollout.

Entries are written by the diagnoser, which reads the trajectory, the produced artifacts, and the evaluator feedback of a completed execution. Two properties of its instructions matter for how the resulting entries should be read. First, the evaluation evidence is the authority: the diagnoser is told to diagnose causes from the recorded evidence rather than to substitute its own scoring preferences. Second, entries are explicitly constrained against leakage, since the same text can later become the teacher condition of a parametric update. The instructions are given in Fig. 6.

A.5.2 Externalized Reuse

Within its own task, an entry is reused directly: the candidate written after one cycle is what the next execution of that task loads. This is the reuse that advances the task's best score, and we do not report it as transfer, since it takes place where the agent has already worked.

Confirmed entries are additionally written into the ExpertHarness experience memory, where they become available to *other* tasks. Retrieval is deliberately conservative. Only promoted entries are indexed, and the retrieval key is built from the transferable views, k^{ds} and k^{gw}, together with the domain label. The query is formed from the task instruction and its domain label; candidates from the same domain are preferred and then ranked by similarity. Retrieval returns at most one entry, and returns nothing when the best candidate falls below a similarity threshold—returning several long

```
You are the ReEvolver Diagnoser, an evidence-grounded skill author for
self-improving Actor agents.

Your responsibility is not to give casual feedback. Convert benchmark evidence
into reusable guidance that can improve the next run of the same task family.
The guidance may also be used as the teacher condition for OPSD training, so it
must be complete, concrete, transferable, and safe.

Evidence and reasoning principles:
- Use the benchmark evaluation evidence as the authority for scoring and failure
  diagnosis. Do not invent a better score, ignore failed rubric items, or replace
  official evaluation semantics with your own preferences.
- Read and reason from the task instruction, Actor output, available workspace
  artifacts, evaluation results, rubric details, logs, and evidence references
  supplied at runtime.
- Diagnose causes, not just symptoms. Connect each important score loss or
  fragile success to the Actor behavior that produced it.
- Compare the skill used this round with the Actor trajectory. Identify which
  guidance was followed, missed, ambiguous, harmful, or absent.
- If the Actor did well, preserve the winning behavior while guarding against
  likely regressions.

Skill quality principles:
- Produce a full execution plan, not a short reminder or patch note.
- State what the Actor should produce, how to produce it, how to verify it, and
  how to recover when checks fail.
- Keep guidance transferable to related tasks without becoming vague. Use precise
  operational steps, validation checks, artifact expectations, and repair
  strategies while abstracting private run details.

Safety and leakage constraints:
- Do not quote or expose internal evaluation wording, target answers, target
  images, review-only checklists, private file paths, private access material,
  endpoints, or contents of environment files.
- Do not copy runtime-only paths, temporary file names, local endpoint names,
  private run details, exact non-public answers, or review-only text into
  reusable guidance.
- Generalize private evaluation criteria into public-facing process guidance
  without exposing private benchmark assets.
- Use only the available evidence. Do not ask the user for additional information.
```

Figure 6: **Diagnoser instructions.** The diagnoser reads a completed execution together with its evaluation feedback and writes the candidate experience entry. The runtime output contract, which fixes the output path and JSON schema, is omitted here.

entries both crowds the context and increases the chance of anchoring on an unrelated procedure, so the memory is read as a suggestion rather than as a mandatory prefix. Two exclusions hold wherever the memory is read. A task is never handed the entry written from its own executions, which would measure memory of a task already seen rather than transfer. And k^{ts} is never retrievable, since it is bound to the deliverable it came from, while k^{fa} is returned only within the domain it was written in, since a guard against a specific failure can misfire elsewhere.

A.5.3 Internalization by On-Policy Self-Distillation

Internalization asks a different question from retrieval: whether the agent still behaves differently once the entry is no longer in context. RSIE therefore distills under an asymmetry between two views of the same task, held by the same model:

$$\text{student prompt} = x_e, \tag{7}$$

$$\text{teacher prompt} = x_e \oplus c(k), \tag{8}$$

$$c(k) = k^{\mathrm{ds}} \oplus k^{\mathrm{gw}}, \tag{9}$$

where x_e is the task instruction and $c(k)$ is the experience condition, which concatenates only the two transferable views. The remaining two views stay in the experience memory: k^{ts} is bound to one deliverable, and k^{fa} is worth only as much as recognizing that the failure is imminent. Neither prompt contains a hidden reference solution, an evaluator checklist, or a past trajectory; historical trajectories are used for evidence, filtering, and diagnosis only.

Table 8: **Settings for the parametric update.** Identical across the compared systems within a benchmark.

Setting	Value
Objective	Generalized JSD
On-policy fraction λ	1.0
Supervised mixing weight α_{SFT}	0
Teacher logits top-k	20
Tuner	LoRA
LoRA rank / α / dropout	16 / 32 / 0.05
Target modules	All linear
Learning rate	1×10^{-6}
Epochs per cycle	1–2
Per-device batch size	1
Gradient accumulation	1
Precision	bfloat16
Attention implementation	SDPA
Sharding (student / teacher)	ZeRO-2 / ZeRO-3
Validation split	None

Writing p_i^{T} and $p_i^{\mathrm{S}}(\theta)$ for the next-token distributions of teacher and student at position i of a response u, the student samples u from its own view and the teacher scores those same tokens under the privileged view:

$$\mathcal{L}(\theta) = \mathbb{E}_{(e,k)\sim\mathcal{B}_t}\, \mathbb{E}_{u\sim\pi_{\bar{\theta}}(\cdot\,|\,x_e)} \left[\frac{1}{|u|} \sum_{i=1}^{|u|} \mathrm{D}_\beta\big(p_i^{\mathrm{T}} \,\big\|\, p_i^{\mathrm{S}}(\theta)\big) \right]. \tag{10}$$

Here $\mathcal{B}_t$ holds the pairs (e, k_e^t) confirmed at cycle t, D_β is the generalized Jensen–Shannon divergence with mixture weight β, and $\bar{\theta} = \mathrm{sg}(\theta)$ applies a stop-gradient to the sampling policy. Every response is drawn from the current student rather than from an earlier policy, and the teacher does not generate a separate answer of its own.

A task contributes an example to cycle t only when all of the following hold: the loaded entry was non-empty and was not the empty-experience baseline; the adapter returned a valid score; that score strictly improved the task's previous best; the entry has a non-empty k^{ds} or k^{gw}; and the example belongs to the current cycle. An entry written during cycle t and not yet tested by a later execution is therefore never trained on.

After all active tasks in a cycle complete, the admitted examples are written to that cycle's dataset. If the dataset is non-empty, the current agent deployment is released, the update is run, and the resulting checkpoint is redeployed as the agent for the next cycle. If no example is admitted, the cycle records that training was skipped, the current deployment is left in place, and no new checkpoint is produced. In the runs reported here, one update draws examples from a single benchmark; we do not mix benchmarks within a single update.

The update is implemented as generalized knowledge distillation with the student sampling fully on-policy ($\lambda = 1$) and no auxiliary supervised term ($\alpha_{\mathrm{SFT}} = 0$), so the objective is exactly Eq. 10 and not a mixture of distillation and imitation. Teacher and student are launched as separate ZeRO-sharded processes, with the teacher serving token-level distributions truncated to its top-k logits. Settings are given in Table 8. The per-cycle dataset is small enough that no validation split is held out, and sequence budgets follow each benchmark's deliverable: SGIBench runs at $16{,}384/4{,}096$ tokens of context and completion, whereas GDPVal uses a long context with a short completion because its submission is the produced artifact rather than the response text.

Table 9: Offline SFT results on ResearchClawBench. Scores use the same native evaluator as the main comparison.

Base agent	Base	SFT	Δ
Qwen3.5-9B	8.83	6.02	-2.81
Qwen3.6-35B	12.1	13.4	$+1.3$

A.6 Supervised Fine-Tuning on ResearchClawBench

A.6.1 Why Offline SFT Is Difficult

We include conventional supervised fine-tuning (SFT) as an offline control to test whether the gains from recursive interaction can be reproduced by replaying previously collected ResearchClawBench trajectories. Long-horizon executions may contain tool failures, incomplete artifacts, evaluator-invalid records, and task-specific decisions that do not transfer across tasks. Moreover, repeated executions of the same task need not provide new coverage: a trajectory audit found 87.3% skill overlap across repeated appearances. Replaying all attempts would therefore overrepresent repeated task-specific behavior while retaining noisy execution artifacts. The SFT experiment measures how much trajectory reconstruction, deduplication, and quality control are required before offline imitation becomes useful.

A.6.2 Trajectory Construction and Quality Control

Training examples are drawn exclusively from the ResearchClawBench training pool; evaluation tasks contribute no prompts, trajectories, artifacts, or evaluator signals. Each execution is reconstructed from archived model messages, tool interactions, and workspace artifacts, and is retained only when the record is complete, internally consistent, and exportable to the supervised format. At most one example is kept per task.

We begin with 1,000 completed trajectories. Schema validation removes broken records, unmatched tool calls, missing artifacts, and invalid evaluator outputs; canonicalization, per-task deduplication, and behavioral quality control leave 348 training examples. A retained trajectory must follow the task contract, use the available evidence, perform the required analysis, and produce the expected artifact without unsupported claims. This reduction shows that successful long-horizon executions are not automatically suitable SFT targets.

The student receives only the task instruction and is trained to reproduce the corresponding assistant/tool trajectory; experience entries, evaluator checklists, hidden references, and evaluator-only assets are excluded. We use LoRA with rank 16, $\alpha = 32$, dropout 0.05, learning rate 10^{-6}, one epoch, all linear target modules, and bfloat16 precision. No validation split is held out. The fine-tuned agent is evaluated with the same sandbox, budget, native evaluator, and held-out protocol as the base agent, with experience memory disabled.

A.6.3 Results and Interpretation

Table 9 summarizes the resulting RCB scores. SFT decreases the smaller model from 8.83 to 6.02, while the larger model gains only 1.3 points, from 12.1 to 13.4. The larger-model gain is obtained only after trajectory reconstruction, per-task deduplication, and behavioral quality control; without these steps, the SFT run does not improve over its base agent.

These results do not show that SFT cannot benefit from more data, stronger curation, or different hyperparameters. They show that, under the controlled data and compute budget used here, offline replay provides a small and capacity-sensitive gain on expert tasks. This is the motivation for recursive interaction: the system must obtain and verify new evidence while acting, rather than relying only on a small, highly filtered set of historical demonstrations.