RSI-CUA: TOWARDS RECURSIVE SELF-IMPROVEMENT OF COMPUTER-USE AGENTS IN AN ENDLESS TRAINING GROUND

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents (CUAs) can improve through environment interaction, but simply accumulating more experience does not determine which experiences best drive progress. As an agent's mastered capabilities and their compositions continually evolve, the central question for sustained improvement is: *what should the agent learn next?* We present **RSI-CUA**, a framework that drives recursive self-improvement through an explicit, evolving *capability state*. In each stage, trajectory-grounded capability profiling analyzes task semantics, interaction traces, and execution outcomes to distinguish capabilities that were successfully exercised, attempted but failed, or unreached. Guided by these identified gaps, RSI-CUA synthesizes new executable tasks calibrated to the learner's empirical capability boundary by balancing complexity, horizon, and composition. Validated tasks accumulate into an expanding *Endless Training Ground* for policy optimization via rejection-sampling SFT and online RL with GRPO, while a fixed 400-task profiling benchmark, GeneralCUABench, tracks stage-consistent capability progression. Evaluated on Qwen3-VL-8B-Instruct across three successive stages, pass@1 success on GeneralCUABench rises from 3.25% to 8.50%, 11.25%, and 17.75%, with the capability-guided curriculum outperforming an unguided counterpart under a matched training budget. On OSWorld, held out from profiling, synthesis, and policy optimization, the score improves from 18.06% to 25.28%. Capability analysis confirms that these gains stem from better action composition, robust state tracking, artifact persistence, and reliable task termination.

1 INTRODUCTION

Computer-use agents (CUAs) operate graphical interfaces through mouse and keyboard actions to accomplish long-horizon tasks across heterogeneous software applications (Xie et al., 2024; Qin et al., 2025; Wang et al., 2026b). Beyond fixed demonstrations, moving toward *experience scaling* through interactive environments allows agents to learn from state transitions, execution outcomes, and recovery attempts (Liu et al., 2026; Zhang et al., 2026a; Xu et al., 2025; Wang et al., 2026a). Yet, simply accumulating more interactions does not determine which experiences best drive progress. As training proceeds, both an agent's mastered capabilities and their compositions continually evolve. The central question for sustained improvement is therefore: *what should the agent learn next?*

Recent work has begun to address this question through learner-aware experience construction, combining autonomous exploration, verifiable task generation, failure-driven learning, policy-aware filtering, and adaptive curricula (Xue et al., 2026a; Kang et al., 2026; He et al., 2026; Sun et al., 2025; Lin et al., 2026; Lv et al., 2026). These approaches establish the value of using the current learner's feedback to shape subsequent training. However, signals such as task success, overall difficulty, or local failure provide only partial descriptions of the capabilities involved in long-horizon GUI behavior. An early navigation error may prevent a later editing operation from being reached, whereas a correctly executed edit may still be followed by a failure to save the artifact or terminate appropriately. The same task-level failure can therefore reflect fundamentally different learning needs.

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.

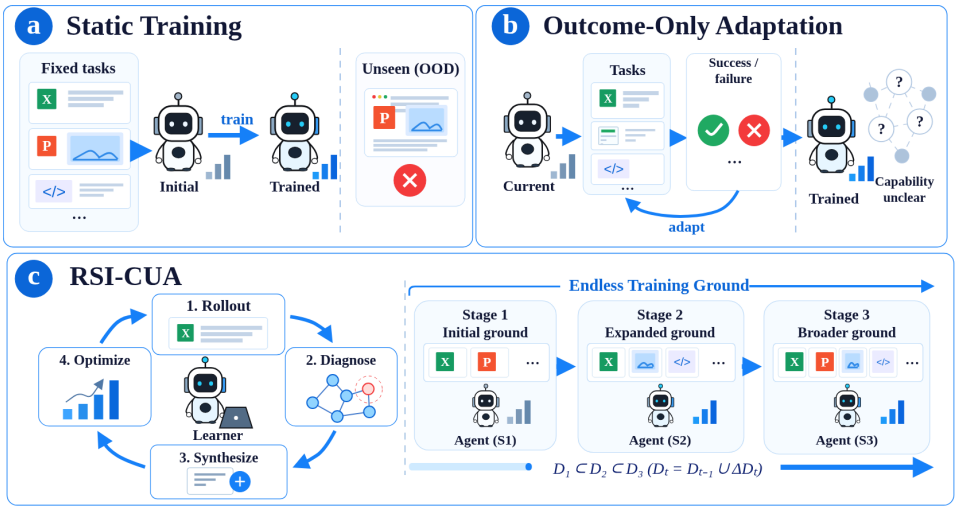


Figure 1: **Conceptual comparison of training paradigms for computer-use agents.** (a) Static training improves within a fixed task distribution but struggles on unseen tasks. (b) Outcome-only adaptation expands training from coarse success/failure signals, leaving the learner's capability structure unclear. (c) RSI-CUA instead diagnoses capability states and gaps, then recursively expands an endless training ground with targeted tasks.

Deciding what to learn next requires identifying not only whether a task failed, but which capabilities were exercised, where execution broke down, and under what conditions those gaps emerged. We present **RSI-CUA**, a framework that drives computer-use agents toward recursive self-improvement through an explicit, evolving *capability state*. In each stage, the learner interacts with executable environments, after which trajectory-grounded capability profiling analyzes task semantics, interaction trajectories, and execution outcomes to distinguish capabilities that were successfully exercised, attempted but failed, or unreached. Rather than relying on static curricula or coarse binary rewards, RSI-CUA uses this structured capability state to guide the next cycle of learner-conditioned experience construction.

Guided by the identified capability gaps, new executable tasks are synthesized and calibrated against the learner's *empirical capability boundary* through execution feedback, balancing task complexity, interaction horizon, and capability composition. Validated tasks accumulate into a cumulative curriculum, forming an expanding *Endless Training Ground*, while a fixed 400-task profiling benchmark, GeneralCUABench, serves as a stage-consistent reference for tracking capability progression. At each stage, the learner is initialized from the same base model and optimized on the cumulative curriculum via rejection-sampling supervised fine-tuning and online reinforcement learning with GRPO. As updated policies reveal new capability gaps, the learner and its training distribution co-evolve through an automated stage-wise loop.

Extensive experiments on Qwen3-VL-8B-Instruct (Bai et al., 2025) across three successive stages demonstrate consistent gains: pass@1 success on the GeneralCUABench rises from 3.25% to 17.75%, with a capability-guided curriculum outperforming an unguided counterpart under a matched training budget. Furthermore, on the held-out OSWorld benchmark (Xie et al., 2024), the overall task completion rate improves from 18.06% to 25.28%. Capability-level analysis confirms that these gains stem from better action composition, robust state tracking, and reliable task termination. Our main contributions are threefold:

- We introduce a trajectory-grounded capability profiling mechanism that combines task semantics, interaction trajectories, and execution outcomes to characterize the learner's capability state and track capability gaps against a profiling benchmark, GeneralCUABench.
- We develop a learner-conditioned experience construction pipeline that synthesizes executable tasks targeting identified capability gaps and calibrates their difficulty through execution feedback, building an expanding *Endless Training Ground*.

- We establish RSI-CUA as a closed-loop framework in which the learner and its curriculum co-evolve, demonstrating sustained multi-stage improvements and strong transfer to held-out environments.

2 Related Work

2.1 Data-Centric Training for Computer-Use Agents

Recent work increasingly improves computer-use agents (CUAs) by scaling and refining interaction data. OpenCUA Wang et al. (2026b), ScaleCUA Liu et al. (2026), TongUI Zhang et al. (2026a), and AgentTrek Xu et al. (2025) expand supervision across software, platforms, and long-horizon procedures, while HATS Shao et al. (2026), ANCHOR Wei et al. (2026), and SCALECUA Lv et al. (2026) emphasize task difficulty, verifiability, and learning value. RSI-CUA complements this line by studying how training experience should evolve with the capability state of the current learner.

2.2 Automated Training Grounds for Computer-Use Agents

Scalable CUA training also depends on executable environments and reliable feedback. Gym-Anything Aggarwal et al. (2026), CUA-Gym Wang et al. (2026a), WebGym Bai et al. (2026), InfiniteWeb Zhang et al. (2026b), GUI-GENESIS Cao et al. (2026), WebFactory Fan et al. (2026), and AutoPlay Ramrakhya et al. (2026) automate environment, task, trajectory, or evaluator construction. While these works primarily scale the training environment and task space, RSI-CUA focuses on how this training ground should adapt as the learner evolves.

2.3 Recursive Self-Improvement for Computer-Use Agents

Recent methods increasingly use learner feedback to adapt subsequent training, including iterative grounding or knowledge refinement Yuan et al. (2025); Zhang et al. (2025), capability-aware difficulty adaptation Kang et al. (2026), trajectory reuse Lin et al. (2026), and closed-loop experience generation and policy optimization Xue et al. (2026b); He et al. (2026); Xue et al. (2026a); Huang et al. (2026). RSI-CUA advances this direction by introducing an explicit, stage-comparable *capability state*: a fixed diagnostic space identifies capability gaps, which directly guide task generation and learner-conditioned calibration in an evolving training space.

3 RSI-CUA

3.1 Overview

RSI-CUA recursively improves computer-use agents through capability profiling, learner-conditioned experience construction, and stage-wise policy optimization. As shown in Fig. 2, GeneralCUABench provides a fixed profiling benchmark, while the *Endless Training Ground* evolves with the learner's diagnosed capability gaps.

Let $\mathcal{B}$ denote the profiling benchmark, $\mathcal{D}_t$ the cumulative curriculum, and π_t the learner at stage t. Each stage follows

$$(\mathcal{S}_t, \mathcal{W}_t) = \mathrm{Profile}(\pi_t, \mathcal{B}), \qquad \mathcal{D}_{t+1} = \mathcal{D}_t \cup \mathrm{Construct}(\pi_t, \mathcal{S}_t, \mathcal{W}_t),$$

where $\mathcal{S}_t$ and $\mathcal{W}_t$ denote the capability state and diagnosed gaps. The next learner is re-initialized from the shared base policy and optimized on $\mathcal{D}_{t+1}$, initiating the next RSI stage.

3.2 GeneralCUABench: A Fixed Capability Profiling Benchmark

Benchmark Construction. We construct GeneralCUABench from real-world software knowledge. For each application, we collect operational knowledge from public resources, including official tutorials, user manuals, technical documentation, and other web resources, and organize it into an application-specific knowledge base. We then instantiate this knowledge as executable GUI tasks, each consisting of a natural-language instruction, a reproducible environment setup, and an automated evaluator. Following the validation procedure described below, the resulting benchmark

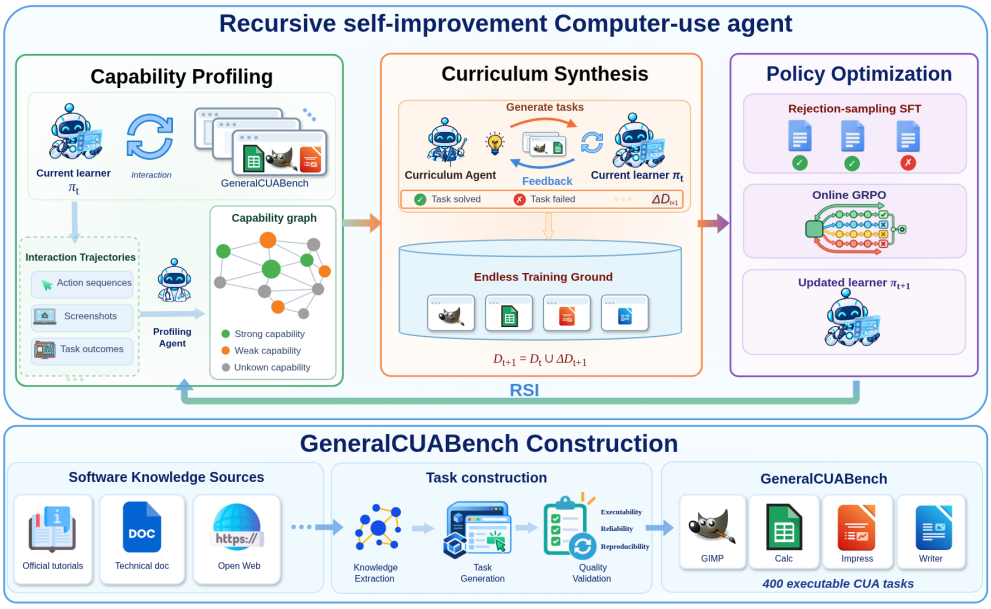


Figure 2: **Overview of RSI-CUA.** A fixed GeneralCUABench profiles the learner's capability state, which guides targeted task synthesis and expansion of the *Endless Training Ground*. The accumulated curriculum then optimizes the next learner, closing the recursive loop.

contains 400 tasks evenly distributed across GIMP, LibreOffice Calc, LibreOffice Impress, and LibreOffice Writer.

Closed-Loop Task Validation. Each constructed task undergoes a closed-loop validation procedure before inclusion in GeneralCUABench. For each task, we instantiate the specified environment in a virtual machine and construct the corresponding target artifact. The resulting system state and artifact are then evaluated by the automated evaluator to check consistency between the task specification and evaluation logic. Human experts further inspect the task specification, constructed artifact, and evaluator logic. Ambiguous or unreliable tasks are revised or discarded.

Capability Coverage and Stage-Consistent Profiling. As shown in Fig. 3, GeneralCUABench contains 400 tasks across four applications, covering diverse task categories and operational patterns. It remains fixed throughout RSI, providing a stage-consistent reference for capability profiling. Task semantics, trajectories, and outcomes are mapped into a shared hierarchical capability space to identify capability states and gaps for subsequent experience construction.

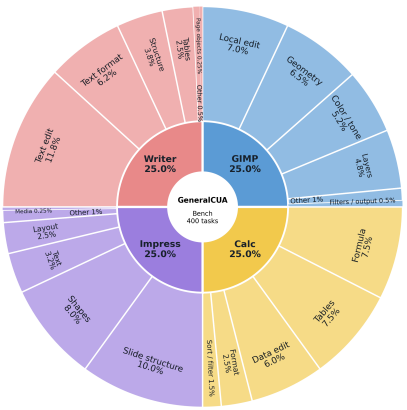


Figure 3: **Task coverage of GeneralCUABench.** The 400 tasks are evenly distributed across four applications and span diverse task categories.

3.3 Endless Training Ground: Learner-Conditioned Experience Construction

While GeneralCUABench provides a fixed reference for capability profiling, the *Endless Training Ground* evolves with the learner. As illustrated in Fig. 4, RSI-CUA identifies capability gaps from trajectory-grounded evidence, synthesizes executable tasks targeting these gaps, and calibrates them using execution feedback from the current learner, thereby tracking its *empirical capability bound-*

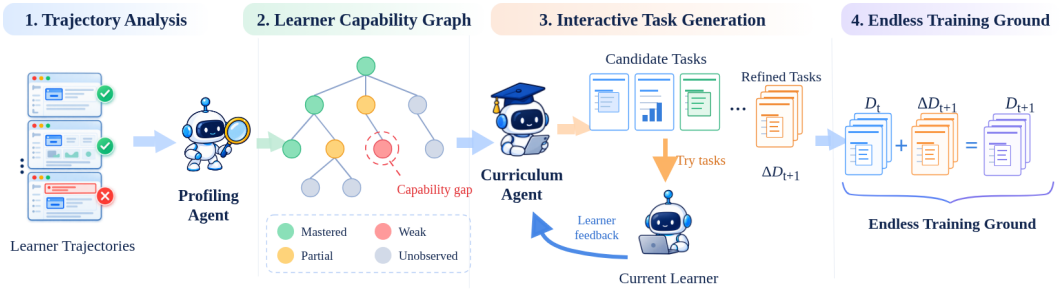


Figure 4: From learner feedback to the Endless Training Ground. Interaction trajectories are transformed into a structured capability state and capability gaps, which guide targeted task generation. Candidate tasks are further calibrated through feedback from the current learner and accumulated into the training curriculum across RSI stages.

ary—the task conditions under which its current capabilities and their compositions succeed or fail in practice. Task synthesis and calibration jointly consider task complexity, interaction horizon, and capability composition, rather than relying only on coarse capability labels or task-level success.

Trajectory-Grounded Capability Profiling. For each benchmark task x_i, the learner π_t produces a trajectory $\tau_{i,j}^t$ and an execution outcome $y_{i,j}^t$. While the outcome provides an episode-level signal, the trajectory exposes finer-grained evidence about which capabilities are exercised and where execution breaks down. A *Capability Profiling Agent* jointly analyzes task semantics, trajectories, and outcomes:

$$e_{i,j}^t = A_{\text{prof}}\left(x_i, \tau_{i,j}^t, y_{i,j}^t\right), \tag{1}$$

where $e_{i,j}^t$ records capability-level evidence as *successfully exercised*, *attempted but failed*, or *unreached*. This distinction prevents downstream capabilities blocked by earlier failures from being incorrectly labeled as failed. We then aggregate evidence across tasks and rollouts:

$$(\mathcal{S}_t, \mathcal{W}_t) = \text{Aggregate}\left(\{e_{i,j}^t\}_{i,j}\right), \tag{2}$$

where $\mathcal{S}_t$ denotes the learner's capability state and $\mathcal{W}_t$ captures capability gaps and their execution contexts, providing the basis for subsequent training-target selection.

Capability-Targeted Task Synthesis and Calibration. Conditioned on $(\mathcal{S}_t, \mathcal{W}_t)$, a *Curriculum Synthesis Agent* generates candidate tasks targeting the learner's current capability gaps:

$$\widetilde{\mathcal{D}}_{t+1} = A_{\text{syn}}\left(\mathcal{S}_t, \mathcal{W}_t\right). \tag{3}$$

Each candidate includes an instruction, a reproducible environment setup, and an automated evaluator, with task complexity, interaction horizon, and capability composition varied to exercise the target capabilities. The current learner π_t then executes these candidates, and its outcomes, progress, and failure modes are fed back to iteratively calibrate task difficulty and structure against its empirical capability boundary. Thus, capability profiling determines *what to learn next*, while execution feedback determines *how to configure the corresponding tasks* for the current learner.

Task Validation and Curriculum Expansion. Before entering the curriculum, candidate tasks are validated for reproducible initialization, executability, and evaluation reliability. Calibration and validation serve complementary roles: calibration adapts newly synthesized tasks to the learner's empirical capability boundary, while validation ensures that the tasks are well-posed and reliably evaluated. Validated tasks form the curriculum increment $\Delta\mathcal{D}_{t+1}$:

$$\mathcal{D}_{t+1} = \mathcal{D}_t \cup \Delta\mathcal{D}_{t+1}. \tag{4}$$

By retaining $\mathcal{D}_t$ and appending newly validated tasks, the *Endless Training Ground* preserves earlier experience while continually expanding toward the learner's evolving capability gaps.

3.4 Stage-Wise Policy Optimization

At each RSI stage, we re-initialize the learner from a shared base policy π_{base} and optimize it over the accumulated curriculum $\mathcal{D}_t$. Optimization follows two phases: rejection-sampling supervised fine-tuning (SFT), followed by online reinforcement learning with Group Relative Policy Optimization (GRPO):

$$\pi_{\text{base}} \xrightarrow{\text{SFT on } \mathcal{D}_t} \pi_t^{\text{SFT}} \xrightarrow{\text{GRPO on } \mathcal{D}_t} \pi_t. \tag{5}$$

Rejection-Sampling Supervised Fine-Tuning. We first sample execution rollouts on $\mathcal{D}_t$ and retain trajectories that successfully complete the corresponding tasks according to the automated evaluators. These successful trajectories form the rejection-sampling SFT dataset $\mathcal{R}_t = \{(x, \tau)\}$. Starting from π_{base}, the policy is fine-tuned on $\mathcal{R}_t$ to acquire successful execution patterns grounded in the current cumulative curriculum.

Online Reinforcement Learning. Starting from π_t^{SFT}, we further optimize the policy with online GRPO on $\mathcal{D}_t$. The learner generates fresh rollouts and receives a binary task-completion reward from the automated evaluator:

$$r(x, \tau) = \begin{cases} 1, & \text{if the task is successfully completed,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Initializing each stage from π_{base} provides a consistent parameter starting point while allowing each learner to train on the full cumulative curriculum, thereby helping alleviate potential capability forgetting across stages, as observed in our experiments and shown in Table 3.

4 Experiments

4.1 Experimental Setup

4.1.1 Benchmarks and Metrics.

Benchmarks. We use GeneralCUABench, comprising 400 deterministically executable GUI tasks, as a fixed capability profiling benchmark, as described in Sec. 3. We additionally evaluate on OSWorld Xie et al. (2024), an external held-out benchmark covering diverse desktop environments and complex multi-application workflows. To ensure contamination-free evaluation, no OSWorld tasks, trajectories, or annotations are used for capability profiling, curriculum generation, or policy optimization. OSWorld therefore measures zero-shot transfer to unseen real-world tasks.

Evaluation Metrics. For GeneralCUABench, we report the pass@1 task success rate for primary comparisons, alongside the repeated-rollout success rate SR_8 for fine-grained trajectory-level diagnostics. For OSWorld, we adopt the overall task completion rate as the primary metric, complemented by application-specific success rates for in-depth analysis. Further details regarding the evaluation protocols and metric definitions are provided in the Appendix.

4.1.2 Baselines.

We benchmark RSI-CUA against several representative open-source computer-use agents, including UI-TARS-1.5-7B Qin et al. (2025), ARPO-UI-TARS-1.5-7B Lu et al. (2025), BEPA-7B-S2 Wang et al. (2026c), EvoCUA-8B Xue et al. (2026a), GUI-Owl-1.5-8B-Instruct Xu et al. (2026), and ANCHOR Qwen3-VL-8B Wei et al. (2026).

4.1.3 Implementation Details

We employ Qwen3-VL-8B-Instruct Bai et al. (2025) as the base model and the initial policy. Both the *Capability Profiling Agent* and the *Curriculum Synthesis Agent* are instantiated with the Codex harness and the GPT-5.6 Luna model. At each stage, training restarts from the base policy and proceeds on the cumulative curriculum using rejection-sampling SFT followed by online GRPO. Detailed agent configurations, training setups, and stage-wise dataset statistics are provided in the Appendix.

4.2 Main Results

4.2.1 Performance on Held-Out Benchmark

We evaluate the final-stage policy, RSI-CUA^{S3}, on the held-out OSWorld benchmark Xie et al. (2024), which is excluded from capability profiling, task synthesis, and policy optimization. As shown in Table 1, RSI-CUA^{S3} achieves 25.28% overall task success, outperforming all evaluated open-source 7B/8B baselines, with notable gains on multi-application workflows, Thunderbird, and Chrome. These results show effective transfer to held-out environments, while application-specific gaps remain.

Table 1: **Results on OSWorld.** Overall task completion and application-level task success rates (%). RSI-CUA^{S3} denotes the policy optimized after three recursive stages.

Model	Overall	GIMP	Calc	Impress	Writer	Chrome	Multi-apps	OS	Thunderbird	VLC	VS Code
BEPA-7B-S2 Wang et al. (2026c)	16.11	30.77	6.38	19.15	26.09	15.22	5.38	33.33	33.33	6.25	26.09
ARPO-UI-TARS-1.5-7B Lu et al. (2025)	15.56	34.62	8.51	17.02	21.74	17.39	5.38	29.17	26.67	12.50	17.39
UI-TARS-1.5-7B Qin et al. (2025)	23.30	46.15	10.64	**29.70**	**39.13**	21.57	2.15	**34.78**	26.67	**35.29**	**60.87**
ANCHOR Qwen3-VL-8B Wei et al. (2026)	18.61	42.31	**14.89**	14.89	**39.13**	17.39	6.45	20.83	33.33	12.50	30.43
RSI-CUA^{S3}	**25.28**	**57.69**	12.77	25.53	30.43	**26.09**	**11.83**	33.33	**53.33**	12.50	43.48

4.2.2 Systematic Evaluation on GeneralCUABench

We evaluate RSI-CUA^{S3} on GeneralCUABench, the fixed in-loop profiling benchmark for tracking capability progression. As shown in Table 2, RSI-CUA^{S3} achieves the highest overall pass@1 success rate of 17.75%, outperforming EvoCUA-8B (15.75%) and ANCHOR Qwen3-VL-8B (10.75%). It performs particularly well on Calc (35.00%) and Impress (18.00%), while application-specific gaps remain. Since GeneralCUABench participates in the profiling loop, these results reflect in-loop capability progression rather than independent generalization; held-out transfer is evaluated separately on OSWorld.

Table 2: **Results on GeneralCUABench.** Pass@1 task success rates (%) across 400 fixed profiling tasks.

Model	Overall	GIMP	Calc	Impress	Writer
EvoCUA-8B Xue et al. (2026a)	15.75	**7.00**	24.00	14.00	**18.00**
ANCHOR Qwen3-VL-8B Wei et al. (2026)	10.75	2.00	24.00	9.00	8.00
GUI-Owl-1.5-8B-Instruct Xu et al. (2026)	5.75	2.00	10.00	5.00	6.00
ARPO-UI-TARS-1.5-7B Lu et al. (2025)	3.75	3.00	2.00	8.00	2.00
BEPA-7B-S2 Wang et al. (2026c)	2.75	0.00	1.00	7.00	3.00
UI-TARS-1.5-7B Qin et al. (2025)	0.50	1.00	1.00	0.00	0.00
RSI-CUA^{S3}	**17.75**	5.00	**35.00**	**18.00**	13.00

4.2.3 Recursive Improvement across Stages

We further examine whether performance continues to improve across recursive stages. As shown in Fig. 5, performance improves consistently across all three stages. On GeneralCUABench, the pass@1 success rate increases from 3.25% for the Qwen3-VL-8B-Instruct base model to 8.50%, 11.25%, and 17.75% at Stages 1–3, respectively, yielding an overall gain of 14.50 percentage points. On the held-out OSWorld benchmark, the overall task completion rate increases from 18.06% to 22.50%, 23.06%, and 25.28%, corresponding to an overall gain of 7.22 percentage points. The rate of improvement varies across stages. In particular, OSWorld exhibits a smaller gain at Stage 2, followed by a larger improvement at Stage 3. Nevertheless, the learner improves at every stage on both benchmarks, indicating that the recursive loop continues to yield measurable gains over multiple cycles of capability profiling, experience construction, and policy optimization.

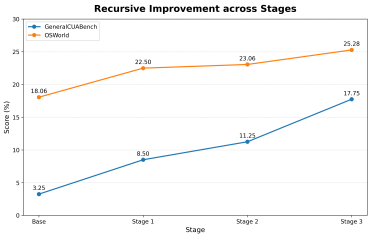


Figure 5: **Recursive improvement across stages.** Performance steadily improves on both GeneralCUABench and OSWorld across three recursive stages.

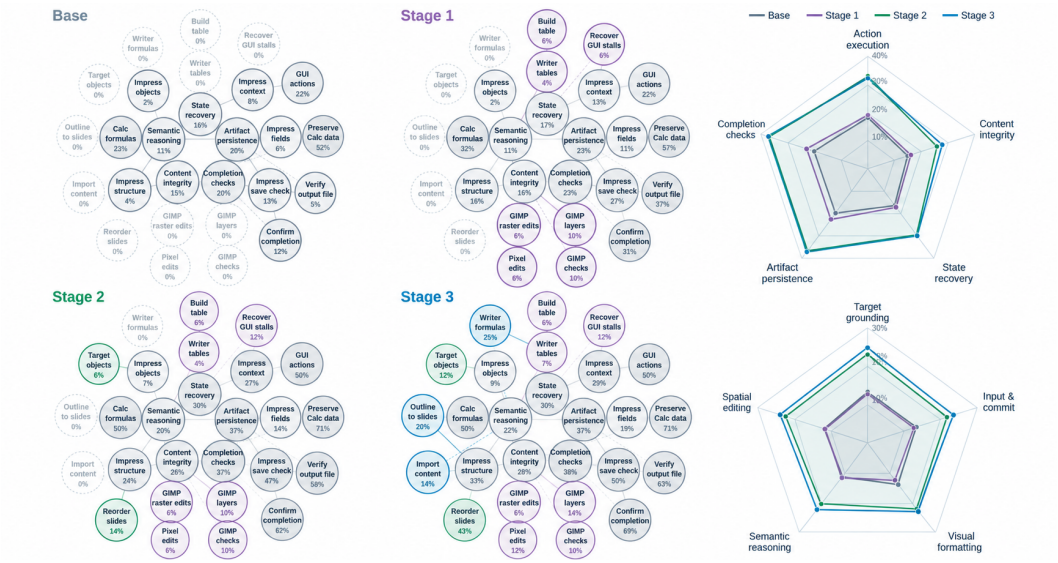


Figure 6: **Capability evolution across RSI stages.** The left panels visualize fine-grained capability states under a shared taxonomy, while the right radar charts summarize high-level capability profiles. Gray, purple, green, and blue denote Base, Stage 1, Stage 2, and Stage 3, respectively, with the same color coding used across the capability graphs and radar charts.

4.3 Capability Evolution in the Endless Training Ground

End-to-end task success captures overall progression but does not reveal how the learner's capability state evolves across recursive stages. We therefore map the Base, Stage 1, Stage 2, and Stage 3 learners into the shared capability space introduced in Sec. 3.3, providing a stage-consistent reference for tracing multidimensional capability evolution.

Broad Capability Expansion across Recursive Stages. We first examine changes in the learner's high-level capability profile. As shown in Fig. 6, the radar charts reveal broad expansion from Stage 1 to Stage 2, with all high-level capability families improving within the fixed profiling space. Representative gains include *Artifact Persistence & Output Control*, which increases from 22.7% to 36.6%, and *Outcome Verification & Task Completion*, which increases from 23.0% to 37.1%. Stage 3 largely retains this expanded profile while further improving selected capability families.

Heterogeneous Fine-Grained Capability Dynamics. Broad improvement at the family level does not imply uniform progress across fine-grained capabilities. The capability graphs in Fig. 6 reveal distinct trajectories across individual capability nodes. For example, although *Artifact Persistence & Output Control* improves overall, its constituent branches evolve differently. *Presentation Persistence* improves consistently from 13.3% for the Base model to 26.7%, 46.7%, and 50.0% across Stages 1–3. *Spreadsheet Persistence* reaches 50.0% at Stage 2 and retains this level at Stage 3. By contrast, *Document Persistence* decreases slightly from 55.2% at Stage 2 to 51.7% at Stage 3, while atomic GIMP path control remains at 0% throughout all stages. These results show that recursive improvement is heterogeneous at the capability level: sustained gains and capability retention coexist with local regressions and persistent gaps. Such heterogeneity further motivates maintaining an explicit capability state, rather than characterizing the learner solely through aggregate task success.

From Partial Competence to Coherent Execution. We next examine how changes in the capability state translate into end-to-end task execution. From Stage 1 to Stage 2, artifact success increases modestly from 9.09% to 10.41%, while strict success rises from 3.31% to 8.41%. Consequently, the strict-to-artifact conversion rate increases from 36.4% to 80.8%. Success on tasks involving multiple capabilities also increases from 18.52% to 31.48%. Together, these results indicate that Stage 2 is more effective at converting partial task progress into complete execution. Overall, the capability-

level analysis in Fig. 6 suggests a transition from *partial competence* to more *coherent execution*, with improved reliability in composing multiple capabilities over long-horizon interactions.

4.4 Ablation Studies

We examine three design questions underlying RSI-CUA: (1) whether capability-targeted curriculum expansion is more effective than indiscriminate exploration, (2) whether rejection-sampling SFT and online GRPO provide complementary optimization signals, and (3) whether cumulative cross-stage optimization is more effective than sequential continual training. All variants are summarized in Table 3.

Does Capability-Targeted Curriculum Outperform Indiscriminate Exploration? We compare RSI-CUA^{S1} with a budget-matched *Random Curriculum*. As shown in Table 3, capability-targeted training improves GeneralCUABench from 7.50% to 8.50% and OSWorld from 21.11% to 22.50%. This indicates that conditioning task construction on the learner's capability state yields a more effective training distribution under the same budget.

Table 3: **Stage-wise ablation studies.**

Variant	GeneralCUABench	OSWorld
Qwen3-VL-8B-Instruct	3.25	18.06
Stage 1		
SFT Only	5.25	21.11
GRPO Only	5.00	21.67
Random Curriculum	7.50	21.11
RSI-CUA^{S1}	**8.50**	**22.50**
Stage 2		
Sequential SFT	4.25	19.17
Cumulative SFT	5.75	21.67
Sequential SFT+GRPO	9.00	22.78
RSI-CUA^{S2}	**11.25**	**23.06**
Stage 3		
SFT Only	7.00	22.78
RSI-CUA^{S3}	**17.75**	**25.28**

Are Rejection-Sampling SFT and Online GRPO Mutually Complementary? At Stage 1, SFT Only and GRPO Only achieve 5.25% and 5.00% on GeneralCUABench, while RSI-CUA^{S1} reaches 8.50%; on OSWorld, the corresponding scores are 21.11%, 21.67%, and 22.50%. The same pattern persists at later stages: adding GRPO to cumulative SFT improves GeneralCUABench from 5.75% to 11.25% at Stage 2 and from 7.00% to 17.75% at Stage 3, with corresponding OSWorld gains from 21.67% to 23.06% and 22.78% to 25.28%. These results support the complementary roles of rejection-sampling SFT and online GRPO.

Why Prioritize Cumulative Optimization over Sequential Continual Training? We compare cumulative optimization, which re-initializes from the shared base policy and trains on the full accumulated *Endless Training Ground*, with sequential continual training from the previous-stage checkpoint. At Stage 2, cumulative training outperforms sequential training on both GeneralCUABench (5.75% vs. 4.25% for SFT; 11.25% vs. 9.00% after SFT+GRPO) and OSWorld (21.67% vs. 19.17%; 23.06% vs. 22.78%). These results favor cumulative optimization and suggest reduced path dependence and potential capability forgetting across stages.

5 Conclusion

In this paper, we introduced **RSI-CUA**, a recursive self-improvement framework that operationalizes trajectory-grounded capability profiling to steer learner-conditioned experience construction within an expanding *Endless Training Ground*. Over three recursive stages, RSI-CUA yields substantial performance gains, boosting the pass@1 accuracy on GeneralCUABench from 3.25% to 17.75% (over a five-fold increase) and demonstrating robust out-of-distribution generalization by advancing held-out OSWorld task completion from 18.06% to 25.28%. These empirical results substantiate that dynamically tailoring the curriculum to the agent's evolving failure modes and competence boundaries unlocks a viable, scalable paradigm for the sustained self-evolution of computer-use agents without human supervision.

AI USE STATEMENT

Generative AI was used both as an integral component of RSI-CUA and as a research-assistance tool. Specifically, the *Capability Profiling Agent* and the *Curriculum Synthesis Agent* are instantiated with the Codex harness and GPT-5.6 Luna, and are used for trajectory-grounded capability profiling and learner-conditioned task construction, respectively. Generative AI tools were also used to assist with manuscript editing and code implementation. All research outputs, experimental results, and final claims were reviewed by the authors, who take full responsibility for the final content of this work.

ETHICS STATEMENT

GeneralCUABench is constructed from publicly accessible software tutorials, user manuals, technical documentation, and other web resources. Automatically constructed tasks are validated for executability and evaluation reliability before inclusion in the benchmark. Human experts further review task specifications, target artifacts, and evaluator logic, while ambiguous or unreliable tasks are revised or discarded. These procedures are intended to reduce errors and unreliable artifacts introduced during automated task construction.

REPRODUCIBILITY STATEMENT

We provide detailed specifications of the RSI-CUA pipeline, including capability profiling, task synthesis and calibration, curriculum accumulation, rejection-sampling SFT, and online GRPO. The Appendix further documents the construction and statistics of GeneralCUABench, the capability ontology, agent prompts and interaction protocol, stage-wise curricula and training data, training configurations, and evaluation settings. These materials are intended to support independent reproduction of the benchmark, training pipeline, and reported results.

REFERENCES

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-anything: Turn any software into an agent environment. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026.

Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. Webgym: Scaling training environments for visual web agents with realistic tasks. *arXiv preprint arXiv:2601.02439*, 2026.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, et al. Gui-genesis: Automated synthesis of efficient environments with verifiable rewards for gui agent post-training. *arXiv preprint arXiv:2602.14093*, 2026.

Sicheng Fan, Qingyun Shi, Shengze Xu, Shengbo Cai, Tieyong Zeng, Li Ling, Yanyi Shang, and Dehan Kong. Webfactory: Automated compression of foundational language intelligence into grounded web agents. *arXiv preprint arXiv:2603.05044*, 2026.

Tiantian He, Yihang Chen, Keyue Jiang, Ka Yiu Lee, Kaiwen Zhou, Kun Shao, and Shuai Wang. Ee-mcp: Self-evolving mcp-gui agents via automated environment generation and experience learning. *arXiv preprint arXiv:2604.09815*, 2026.

Mianqiu Huang, Taofeng Xue, Chong Peng, Jinrui Ding, Sicheng Fan, Jiale Hong, Yufei Gao, Xiaocheng Zhang, Linsen Guo, Xin Yang, et al. Evocua-1.5: Online reinforcement learning for multi-turn computer-use agents. *arXiv preprint arXiv:2607.09773*, 2026.

Linjia Kang, Zhimin Wang, Yongkang Zhang, Duo Wu, Jinghe Wang, Ming Ma, Haopeng Yan, and Zhi Wang. Learning with challenges: Adaptive difficulty-aware data generation for mobile gui agent training. *arXiv preprint arXiv:2601.22781*, 2026.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, et al. Ui-voyager: A self-evolving gui agent learning via failed experience. *arXiv preprint arXiv:2603.24533*, 2026.

Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, et al. Scalecua: Scaling open-source computer use agents with cross-platform data. In *International Conference on Learning Representations*, volume 2026, pp. 62707–62766, 2026.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025.

Bowen Lv, Xiao Liu, Yanyu Ren, Hanyu Lai, Bohao Jing, Hanchen Zhang, Yanxiao Zhao, Shuntian Yao, Jie Tang, and Yuxiao Dong. Scalecua: Scaling computer use agents with verifiable task synthesis and efficient online rl. *arXiv preprint arXiv:2607.11185*, 2026.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Ram Ramrakhya, Andrew Szot, Omar Attia, Bogdan Mazoure, Anh Nguyen, Yuhao Yang, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. In *International Conference on Learning Representations*, volume 2026, pp. 123630–123660, 2026.

Rui Shao, Ruize Gao, Bin Xie, Yixing Li, Kaiwen Zhou, Shuai Wang, Weili Guan, and Gongwei Chen. Hats: Hardness-aware trajectory synthesis for gui agents. *arXiv preprint arXiv:2603.12138*, 2026.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. Seagent: Self-evolving computer use agent with autonomous learning from experience. *arXiv preprint arXiv:2508.04700*, 2025.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, et al. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents. *arXiv preprint arXiv:2605.25624*, 2026a.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, et al. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems*, 38:139756–139806, 2026b.

Zezhou Wang, Ziyun Zhang, Xiaoyi Zhang, Zhuzhong Qian, and Yan Lu. From off-policy to on-policy: Enhancing gui agents via bi-level expert-to-policy assimilation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23581–23623, 2026c.

Jinbiao Wei, Yilun Zhao, Kangqi Ni, and Arman Cohan. Anchor: Branch-point data generation for gui agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17031–17047, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026a.

Tianci Xue, Zeyi Liao, Tianneng Shi, Zilu Wang, Kai Zhang, Dawn Song, Yu Su, and Huan Sun. Autonomous continual learning of computer-use agents for environment adaptation. *arXiv e-prints*, pp. arXiv–2602, 2026b.

Xinbin Yuan, Jian Zhang, Kaixin Li, Zhuoxuan Cai, Lujian Yao, Jie Chen, Enguang Wang, Qibin Hou, Jinwei Chen, Peng-Tao Jiang, et al. Enhancing visual grounding for gui agents via self-evolutionary reinforcement learning. *arXiv preprint arXiv:2505.12370*, 2025.

Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. Tongui: Internet-scale trajectories from multimodal web tutorials for generalized gui agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 12367–12375, 2026a.

Ziyun Zhang, Xinyi Liu, Xiaoyi Zhang, Jun Wang, Gang Chen, and Yan Lu. Ui-evol: Automatic knowledge evolving for computer use agents. *arXiv preprint arXiv:2505.21964*, 2025.

Ziyun Zhang, Zezhou Wang, Xiaoyi Zhang, Zongyu Guo, Jiahao Li, Bin Li, and Yan Lu. Infiniteweb: Scalable web environment synthesis for gui agent training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28465–28492, 2026b.

APPENDIX

A Operational Definitions of RSI-CUA **14**
A.1 Stage State and Recursive Data Flow 14
A.2 Task, Trajectory, and Outcome Objects 14
A.3 Capability Profiling and Evidence Aggregation 15
A.4 Candidate Synthesis, Calibration, and Validation 16
A.5 Stage-Wise Policy Optimization 16

B Capability Ontology **17**
B.1 Cross-Domain Capability Families 17
B.2 Application-Specific Workflow Capabilities 17
B.3 Atomic GUI Skills 18

C Agent Prompts and GUI Protocol **20**
C.1 Observation, Action, and Termination 20
C.2 Agent Roles and Prompt Templates 21

D Stage Curricula and Training Data **32**
D.1 SFT Corpus Size 32
D.2 GRPO Curriculum Schedule 33

E Model Training and Runtime Configuration **33**
E.1 Initialization and Optimization 33
E.2 SFT Configuration 33
E.3 GRPO Configuration 33
E.4 Runtime and Checkpointing 34

F Additional Experimental Results **34**
F.1 GeneralCUABench Pass@1 34
F.2 OSWorld Transfer Results 34

G Trajectory Case Study **35**
G.1 Stage-Wise GIMP Trajectory Case Study 35

A Operational Definitions of RSI-CUA

This appendix makes the core objects and stage-wise transitions in RSI-CUA explicit. We describe the task and trajectory representations, capability profiling and aggregation, learner-conditioned task construction, curriculum accumulation, and stage-wise policy optimization used throughout the recursive improvement process.

A.1 Stage State and Recursive Data Flow

Let $\mathcal{B}$ denote the fixed profiling benchmark, $\mathcal{D}_t$ the cumulative training curriculum available at stage t, and π_t the learner obtained by optimizing the shared base policy on $\mathcal{D}_t$. The learner π_t is evaluated on $\mathcal{B}$ to obtain its capability state and diagnosed gaps, which guide construction of the curriculum increment $\Delta\mathcal{D}_{t+1}$. After validation, the cumulative curriculum is updated as $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \Delta\mathcal{D}_{t+1}$, and the next learner π_{t+1} is obtained by re-initializing from the shared base policy and optimizing on $\mathcal{D}_{t+1}$. The profiling benchmark remains fixed across stages, whereas the training curriculum evolves.

A.2 Task, Trajectory, and Outcome Objects

For task i, rollout j, and learner stage t, we use the following notation.

Table 4: Core notation used in RSI-CUA.

Symbol	Type	Operational meaning
x_i	Task specification	A GUI task defined by a natural-language instruction, reproducible initial environment, target state or artifact, task-specific evaluator, and capability annotation.
π_t	Learner policy	The learner evaluated at stage t and used for capability profiling and learner-conditioned task calibration.
$\tau_{i,j}^t$	Rollout	An ordered interaction trajectory consisting of observations, GUI actions, environment transitions, and termination.
$y_{i,j}^t$	Execution outcome	The task-level outcome produced by the automated evaluator for rollout $\tau_{i,j}^t$.
A_{prof}	Capability Profiling agent	Maps the task specification, interaction trajectory, and execution outcome to structured capability-level evidence.
$e_{i,j}^t$	Capability evidence	Capability-level evidence indicating whether a capability was successfully exercised, attempted but failed, or unreached during a rollout.
Aggregate	Aggregation operator	Aggregates rollout-level capability evidence in the shared capability space to obtain the capability state $\mathcal{S}_t$ and capability gaps $\mathcal{W}_t$.
A_{syn}	Curriculum synthesis agent	Generates candidate GUI tasks conditioned on the capability state $\mathcal{S}_t$ and diagnosed gaps $\mathcal{W}_t$.
$\widetilde{\mathcal{D}}_{t+1}$	Candidate task set	Newly synthesized tasks that undergo learner-conditioned calibration and task validation.
$\Delta\mathcal{D}_{t+1}$	Curriculum increment	Validated tasks added to the cumulative training curriculum for the next RSI stage.

Artifact Success, Strict Success, and Repeated Evaluation. For a valid rollout τ on task x, we distinguish artifact success from strict task success. Artifact success indicates that all artifact-level assertions defined by the task-specific evaluator are satisfied:

$$S_{\text{art}}(x, \tau) = \mathbb{I}\left[\text{all artifact-level assertions pass}\right].$$

Strict success additionally requires all remaining evaluator-defined task conditions to be satisfied and the learner to explicitly report successful completion:

$$\begin{aligned} S_{\text{strict}}(x, \tau) = S_{\text{art}}(x, \tau) \\ \cdot \mathbb{I}\left[\text{all remaining evaluator-defined task conditions pass}\right] \\ \cdot \mathbb{I}\left[\text{learner reports successful completion}\right]. \end{aligned}$$

For brevity, we use

$$S(x, \tau) \equiv S_{\text{strict}}(x, \tau)$$

throughout the remaining evaluation definitions.

For a task evaluated with a single rollout, pass@1 is given by $S_{\text{strict}}(x, \tau)$. Benchmark-level pass@1 is the average strict-success rate across the corresponding evaluation tasks.

For repeated evaluation, we define the empirical repeated-rollout success rate over eight valid rollouts as

$$\text{SR}_8(x_i; \pi_t) = \frac{1}{8} \sum_{j=1}^{8} S_{\text{strict}}(x_i, \tau_{i,j}^t).$$

Thus, SR_8 measures the empirical fraction of strict successes across eight repeated trials and may be reported as a count such as $6/8$.

We separately define pass@8 as whether at least one of the eight valid rollouts achieves strict success:

$$\text{pass@8}(x_i; \pi_t) = \mathbb{I}\left[\sum_{j=1}^{8} S_{\text{strict}}(x_i, \tau_{i,j}^t) > 0\right].$$

Thus, $\text{pass@8}(x_i; \pi_t) = 1$ if at least one of the eight rollouts achieves strict success, and 0 otherwise.

Artifact-to-Strict Conversion. Let $\mathcal{V}_t$ denote the set of valid evaluation rollouts used for the artifact-versus-strict analysis at stage t. We define the artifact-success rate and strict-success rate as

$$\text{AS}_t = \frac{1}{|\mathcal{V}_t|} \sum_{(x,\tau)\in\mathcal{V}_t} S_{\text{art}}(x, \tau),$$

and

$$\text{SS}_t = \frac{1}{|\mathcal{V}_t|} \sum_{(x,\tau)\in\mathcal{V}_t} S_{\text{strict}}(x, \tau).$$

The strict-to-artifact conversion rate is defined as

$$\text{Conv}_t = \frac{\text{SS}_t}{\text{AS}_t} = \frac{\sum_{(x,\tau)\in\mathcal{V}_t} S_{\text{strict}}(x, \tau)}{\sum_{(x,\tau)\in\mathcal{V}_t} S_{\text{art}}(x, \tau)},$$

for $\text{AS}_t > 0$. This quantity measures the fraction of artifact-successful rollouts that also satisfy the strict task-completion criterion.

A.3 Capability Profiling and Evidence Aggregation

For each profiling task x_i, rollout j, and learner stage t, the Capability Profiling Agent jointly analyzes the task specification, interaction trajectory, and execution outcome:

$$e_{i,j}^t = A_{\text{prof}}\left(x_i, \tau_{i,j}^t, y_{i,j}^t\right),$$

where $e_{i,j}^t$ denotes structured capability-level evidence extracted from the rollout.

For each capability involved in a task, the profiling process distinguishes three execution states: *successfully exercised*, *attempted but failed*, and *unreached*. This distinction is important for long-horizon GUI interaction, where an early execution failure may prevent downstream capabilities from being exercised. In such cases, capabilities that are not reached are treated as unobserved rather than failed, avoiding the propagation of a single upstream error into multiple downstream capability failures.

We aggregate rollout-level evidence from the fixed profiling benchmark to obtain the learner's stage-wise capability state and diagnosed capability gaps:

$$(\mathcal{S}_t, \mathcal{W}_t) = \text{Aggregate}\left(\{e_{i,j}^t\}_{i,j}\right).$$

Here, $\mathcal{S}_t$ represents the learner's capability state in the shared capability space, while $\mathcal{W}_t$ captures diagnosed capability gaps together with the task and trajectory contexts in which they are observed. The resulting capability state and diagnosed gaps provide the conditioning signals for learner-conditioned task synthesis in the next stage.

Capability-Node Scoring. For the quantitative capability analysis, let $\mathcal{B}_c \subseteq \mathcal{B}$ denote the fixed set of profiling tasks associated with capability node c by the benchmark capability mapping.

The stage-wise score of capability node c is defined as

$$C_t(c) = \frac{1}{|\mathcal{B}_c|} \sum_{x_i \in \mathcal{B}_c} \text{pass@8}(x_i; \pi_t).$$

Therefore, $C_t(c)$ measures the fraction of profiling tasks associated with capability c for which the learner succeeds at least once among eight valid rollouts.

The same scoring rule is applied directly to K-, D-, and M-level capability nodes using their corresponding fixed supporting-task sets. Higher-level D- and M-level scores are therefore computed over the tasks associated with those nodes, rather than by macro-averaging the scores of their child nodes. A task associated with multiple capability nodes contributes once to the supporting-task set of each corresponding node.

The trajectory-level states—*successfully exercised*, *attempted but failed*, and *unreached*—serve as qualitative evidence for capability profiling and gap identification. They are distinct from the quantitative node score $C_t(c)$ used to visualize stage-wise capability evolution.

For Base, Stage 1, and Stage 2, the eight rollouts are obtained from the corresponding repeated-rollout evaluation batch. For Stage 3, pass@8 is computed over eight independently executed pass@1 trials.

In this way, capability profiling combines trajectory-grounded evidence about *where execution succeeds or breaks down* with a stage-comparable quantitative capability score, providing both the contextual gaps $\mathcal{W}_t$ used for subsequent training-target selection and the scored capability state used for analysis.

A.4 Candidate Synthesis, Calibration, and Validation

Conditioned on the diagnosed capability state and gaps, the Curriculum Synthesis Agent generates a set of candidate tasks:

$$\widetilde{\mathcal{D}}_{t+1} = A_{\text{syn}}(\mathcal{S}_t, \mathcal{W}_t).$$

Each candidate task contains a natural-language instruction, a reproducible environment setup, and an automated evaluator, and is constructed to target the diagnosed capability gaps. During synthesis, task complexity, interaction horizon, and capability composition are varied to produce tasks that exercise the target capabilities under different conditions.

The current learner π_t then interacts with the synthesized tasks. Its execution outcomes, progress, and failure modes provide feedback for calibrating task difficulty and structure to the learner's empirical capability boundary. In this way, capability profiling determines *what* should be learned next, while learner feedback helps determine *how* the corresponding training tasks should be instantiated.

Calibration and task validation serve different purposes. Calibration adapts candidate tasks to the current learner, whereas validation ensures that the resulting tasks can be reproducibly initialized, executed, and reliably evaluated. Only validated tasks are added to the curriculum. Let $\Delta\mathcal{D}_{t+1}$ denote the validated task increment; the cumulative curriculum is updated as

$$\mathcal{D}_{t+1} = \mathcal{D}_t \cup \Delta\mathcal{D}_{t+1}.$$

By retaining previously validated tasks and appending newly constructed ones, the Endless Training Ground expands across RSI stages while preserving earlier training experience.

A.5 Stage-Wise Policy Optimization

At each RSI stage, the learner is re-initialized from the shared base policy π_{base} and optimized on the accumulated curriculum $\mathcal{D}_t$. Training consists of rejection-sampling supervised fine-tuning followed by online reinforcement learning with GRPO:

$$\pi_{\text{base}} \xrightarrow{\text{RS-SFT}(\mathcal{D}_t)} \pi_t^{\text{SFT}} \xrightarrow{\text{GRPO}(\mathcal{D}_t)} \pi_t.$$

For rejection-sampling SFT, we sample execution trajectories on $\mathcal{D}_t$ and retain those that successfully complete the corresponding tasks according to the automated evaluators. The resulting training set is

$$\mathcal{R}_t = \{(x, \tau) : r(x, \tau) = 1\},$$

where $r(x, \tau)$ denotes the task-completion reward. The SFT objective is

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(x,\tau)\sim\mathcal{R}_t} \sum_k \log \pi_\theta(a_k \mid x, h_k),$$

where h_k denotes the interaction history before action a_k.

Starting from π_t^{SFT}, we further optimize the learner with online GRPO on the same cumulative curriculum. The learner generates fresh rollouts and receives a binary task-completion reward from the corresponding automated evaluator:

$$r(x, \tau) = \begin{cases} 1, & \text{if the task is successfully completed,} \\ 0, & \text{otherwise.} \end{cases}$$

The resulting policy π_t is used for capability profiling and learner-conditioned task calibration in the subsequent RSI stage.

B Capability Ontology

For stage-consistent capability profiling, RSI-CUA uses a fixed three-level capability ontology shared across all recursive stages. The ontology contains 181 nodes, including 10 cross-domain capability families (M-level), 26 application-specific workflow capabilities (D-level), and 145 atomic GUI skills (K-level). Since a fine-grained skill may support multiple higher-level capabilities, the ontology is represented as a graph rather than a strict tree. The ontology structure remains fixed throughout RSI, while stage-dependent capability scores are overlaid on the same capability space to characterize the learner's evolving capability state. Tables 5–7 list the complete capability space used in our analysis.

B.1 Cross-Domain Capability Families

Table 5: Cross-domain capability families (M-level).

Node ID	Capability
M_ARTIFACT_PERSISTENCE	Artifact persistence
M_CLOSED_LOOP_RECOVERY	State recovery
M_CONTENT_PRESERVATION	Content integrity
M_DOMAIN_SEMANTIC_REASONING	Semantic reasoning
M_EXACT_INPUT_AND_COMMIT	Input & commit
M_EXECUTABLE_ACTION_CONTROL	Action execution
M_POSTCONDITION_COMPLETION	Completion checks
M_SPATIAL_STRUCTURE_MANIPULATION	Spatial editing
M_STYLING_AND_VISUAL_FORMAT	Visual formatting
M_TARGET_GROUNDING	Target grounding

B.2 Application-Specific Workflow Capabilities

Table 6: Application-specific workflow capabilities (D-level) and their linked cross-domain capability families.

Node ID	App.	Workflow Capability	Linked M-level Capabilities
D_CALC_ANALYSIS_TOOLS	Calc	Calc analysis tools	Content integrity; Semantic reasoning

Continued on next page

Node ID	App.	Workflow Capability	Linked M-level Capabilities
D_CALC_BASIC_EDITING_FILL	Calc	Calc cell & fill edits	State recovery; Semantic reasoning; Input & commit; Spatial editing
D_CALC_DATA_INTEGRITY	Calc	Calc data integrity	Content integrity; Completion checks
D_CALC_FORMATTING	Calc	Calc formatting	Input & commit; Visual formatting
D_CALC_FORMULA_REASONING	Calc	Calc formulas	Semantic reasoning; Input & commit
D_CALC_INTERACTION_GROUNDING	Calc	Calc navigation & focus	State recovery; Input & commit; Action execution; Spatial editing; Visual formatting; Target grounding
D_CALC_PERSISTENCE_COMPLETION	Calc	Calc save checks	Artifact persistence; Content integrity; Completion checks
D_CALC_SHEET_STRUCTURE	Calc	Calc sheet structure	Spatial editing
D_GIMP_COLOR_TONE	GIMP	GIMP color & tone	Semantic reasoning; Input & commit; Visual formatting
D_GIMP_INTERACTION_CONTROL	GIMP	GIMP action recovery	State recovery; Semantic reasoning; Action execution; Target grounding
D_GIMP_LAYER_COMPOSITION	GIMP	GIMP layers	Content integrity; Spatial editing; Visual formatting
D_GIMP_LOCAL_RASTER_EDITING	GIMP	GIMP raster edits	Content integrity; Semantic reasoning; Spatial editing; Visual formatting
D_GIMP_PERSISTENCE_COMPLETION	GIMP	GIMP export checks	Artifact persistence; Content integrity; Input & commit; Completion checks
D_GIMP_SELECTION_GROUNDING	GIMP	GIMP target selection	Spatial editing; Target grounding
D_GIMP_TRANSFORM_CANVAS	GIMP	GIMP canvas transforms	Spatial editing
D_IMPRESS_CONTEXT_CONTROL	Impress	Impress context	State recovery; Action execution; Target grounding
D_IMPRESS_FIELDS_TEXT	Impress	Impress fields	Semantic reasoning; Input & commit; Visual formatting
D_IMPRESS_OBJECT_EDITING	Impress	Impress objects	Semantic reasoning; Input & commit; Spatial editing; Visual formatting; Target grounding
D_IMPRESS_PERSISTENCE_COMPLETION	Impress	Impress save check	Artifact persistence; Completion checks
D_IMPRESS_PRESENTATION_DESIGN	Impress	Impress slide design	Semantic reasoning; Visual formatting
D_IMPRESS_SLIDE_STRUCTURE	Impress	Impress structure	Content integrity; Semantic reasoning; Spatial editing; Target grounding
D_WRITER_CONTEXT_CONTROL	Writer	Writer context	State recovery; Action execution; Target grounding
D_WRITER_FIELDS_METADATA	Writer	Writer fields & metadata	Semantic reasoning
D_WRITER_PERSISTENCE_COMPLETION	Writer	Writer save checks	Artifact persistence; Completion checks
D_WRITER_TABLE_EDITING	Writer	Writer tables	State recovery; Semantic reasoning; Input & commit; Spatial editing; Target grounding
D_WRITER_TEXT_EDITING	Writer	Writer text editing	Content integrity; Semantic reasoning; Input & commit; Spatial editing; Visual formatting; Target grounding

B.3 Atomic GUI Skills

Table 7: Atomic GUI skills (K-level).

Node ID	App.	Atomic Skill
K_GIMP_001	GIMP	GIMP GUI actions
K_GIMP_002	GIMP	Brush geometry
K_GIMP_003	GIMP	GIMP export dialogs
K_GIMP_004	GIMP	Save GIMP artifact
K_GIMP_005	GIMP	Finish GIMP task
K_GIMP_006	GIMP	Pixel edits
K_GIMP_007	GIMP	Rotation settings
K_GIMP_008	GIMP	Numeric field input
K_GIMP_009	GIMP	Precise masks
K_GIMP_010	GIMP	GIMP selection tools
K_GIMP_011	GIMP	GIMP color controls
K_GIMP_012	GIMP	Targeted pixel fill
K_GIMP_013	GIMP	RGB color reasoning
K_GIMP_014	GIMP	Raster text editing
K_GIMP_015	GIMP	Copy & clone pixels
K_GIMP_016	GIMP	Visual target location
K_GIMP_017	GIMP	Recover stalled GIMP
K_GIMP_018	GIMP	Plan GIMP workflow
K_GIMP_019	GIMP	GIMP checks
K_GIMP_020	GIMP	Crop geometry
K_GIMP_021	GIMP	Repair malformed actions

Continued on next page

Node ID	App.	Atomic Skill
K_GIMP_022	GIMP	Curves adjustments
K_GIMP_023	GIMP	Export paths & names
K_GIMP_024	GIMP	Image & layer scaling
K_GIMP_025	GIMP	Indexed color conversion
K_GIMP_026	GIMP	Canvas coordinate mapping
K_GIMP_027	GIMP	Gradient settings
K_GIMP_028	GIMP	Image & layer flips
K_GIMP_029	GIMP	Levels adjustments
K_GIMP_030	GIMP	GIMP menus & files
K_GIMP_031	GIMP	Paths & Stroke Path
K_GIMP_032	GIMP	Raster displacement
K_GIMP_033	GIMP	GIMP layer management
K_GIMP_034	GIMP	Transparency & erase
K_GIMP_035	GIMP	Canvas resizing
K_GIMP_036	GIMP	Diagnose missing inputs
K_GIMP_037	GIMP	Other GIMP adjustments
K_GIMP_038	GIMP	Color temperature
K_GIMP_039	GIMP	Cut vs crop
K_GIMP_040	GIMP	Active tool & dialog state
K_GIMP_041	GIMP	Brightness & contrast
K_LIBREOFFICE_CALC_001	Calc	Calc GUI actions
K_LIBREOFFICE_CALC_002	Calc	Worksheet navigation
K_LIBREOFFICE_CALC_003	Calc	Exact Calc range selection
K_LIBREOFFICE_CALC_004	Calc	Preserve Calc data
K_LIBREOFFICE_CALC_005	Calc	Fill down
K_LIBREOFFICE_CALC_006	Calc	Fill-handle drag
K_LIBREOFFICE_CALC_007	Calc	Recover stalled Calc
K_LIBREOFFICE_CALC_008	Calc	Calc output destination
K_LIBREOFFICE_CALC_009	Calc	Cellwise calculation
K_LIBREOFFICE_CALC_010	Calc	Save Calc artifact
K_LIBREOFFICE_CALC_011	Calc	Other Calc tasks
K_LIBREOFFICE_CALC_012	Calc	Calc save dialogs
K_LIBREOFFICE_CALC_013	Calc	Close save transaction
K_LIBREOFFICE_CALC_014	Calc	Row-wise Fill Left
K_LIBREOFFICE_CALC_015	Calc	Target cells & enter data
K_LIBREOFFICE_CALC_016	Calc	Calc completion checks
K_LIBREOFFICE_CALC_017	Calc	Spreadsheet shape checks
K_LIBREOFFICE_CALC_018	Calc	Numeric sequences
K_LIBREOFFICE_CALC_019	Calc	Geometric sequences
K_LIBREOFFICE_CALC_020	Calc	Repair numeric fills
K_LIBREOFFICE_CALC_021	Calc	Delete cell contents
K_LIBREOFFICE_CALC_022	Calc	Row & cell structure
K_LIBREOFFICE_CALC_023	Calc	Reorder or copy sheets
K_LIBREOFFICE_CALC_024	Calc	Safe Calc text entry
K_LIBREOFFICE_CALC_025	Calc	Insert columns
K_LIBREOFFICE_CALC_026	Calc	Worksheet topology
K_LIBREOFFICE_CALC_027	Calc	Merge or unmerge ranges
K_LIBREOFFICE_CALC_028	Calc	Exact cell colors
K_LIBREOFFICE_CALC_029	Calc	Solver setup
K_LIBREOFFICE_CALC_030	Calc	Correlation analysis
K_LIBREOFFICE_CALC_031	Calc	Formula entry & commit
K_LIBREOFFICE_CALC_032	Calc	Descriptive statistics
K_LIBREOFFICE_CALC_033	Calc	Calc formatting controls
K_LIBREOFFICE_CALC_034	Calc	Number formats
K_LIBREOFFICE_CALC_035	Calc	Fourier analysis
K_LIBREOFFICE_CALC_036	Calc	F-test analysis
K_LIBREOFFICE_CALC_037	Calc	Text to Columns
K_LIBREOFFICE_CALC_038	Calc	Formula references
K_LIBREOFFICE_CALC_039	Calc	Conditional formatting
K_LIBREOFFICE_CALC_040	Calc	Cross-sheet formulas
K_LIBREOFFICE_CALC_041	Calc	Data sorting
K_LIBREOFFICE_CALC_042	Calc	Transpose ranges
K_LIBREOFFICE_CALC_043	Calc	Pivot Tables
K_LIBREOFFICE_CALC_044	Calc	Safe GUI serialization
K_LIBREOFFICE_CALC_045	Calc	Selective bold
K_LIBREOFFICE_CALC_046	Calc	Goal Seek
K_LIBREOFFICE_CALC_047	Calc	Multiple Operations
K_LIBREOFFICE_CALC_048	Calc	Advanced Filter
K_LIBREOFFICE_CALC_049	Calc	Named ranges & formulas
K_LIBREOFFICE_CALC_050	Calc	Repeated formula fill
K_LIBREOFFICE_CALC_051	Calc	Other statistics
K_LIBREOFFICE_IMPRESS_001	Impress	GUI actions
K_LIBREOFFICE_IMPRESS_002	Impress	Editable slide state
K_LIBREOFFICE_IMPRESS_003	Impress	Impress document context

Continued on next page

Node ID	App.	Atomic Skill
K_LIBREOFFICE_IMPRESS_004	Impress	Recover GUI stalls
K_LIBREOFFICE_IMPRESS_005	Impress	Dismiss blocking dialogs
K_LIBREOFFICE_IMPRESS_006	Impress	Select slide
K_LIBREOFFICE_IMPRESS_007	Impress	Delete slide
K_LIBREOFFICE_IMPRESS_008	Impress	Duplicate slide
K_LIBREOFFICE_IMPRESS_009	Impress	Insert slides
K_LIBREOFFICE_IMPRESS_010	Impress	Reorder slides
K_LIBREOFFICE_IMPRESS_011	Impress	Impress Save As
K_LIBREOFFICE_IMPRESS_012	Impress	Verify output file
K_LIBREOFFICE_IMPRESS_013	Impress	Confirm completion
K_LIBREOFFICE_IMPRESS_014	Impress	Preserve other slides
K_LIBREOFFICE_IMPRESS_015	Impress	Header & Footer controls
K_LIBREOFFICE_IMPRESS_016	Impress	Footer & page numbers
K_LIBREOFFICE_IMPRESS_017	Impress	Date field options
K_LIBREOFFICE_IMPRESS_018	Impress	Insert date fields
K_LIBREOFFICE_IMPRESS_019	Impress	Edit date fields
K_LIBREOFFICE_IMPRESS_020	Impress	Slide text editing
K_LIBREOFFICE_IMPRESS_021	Impress	Slide text formatting
K_LIBREOFFICE_IMPRESS_022	Impress	Target objects
K_LIBREOFFICE_IMPRESS_023	Impress	Object geometry
K_LIBREOFFICE_IMPRESS_024	Impress	Draw shapes
K_LIBREOFFICE_IMPRESS_025	Impress	Shape styling
K_LIBREOFFICE_IMPRESS_026	Impress	Object z-order
K_LIBREOFFICE_IMPRESS_027	Impress	Connectors
K_LIBREOFFICE_IMPRESS_028	Impress	Impress tables
K_LIBREOFFICE_IMPRESS_029	Impress	Slide transitions
K_LIBREOFFICE_IMPRESS_030	Impress	Slide backgrounds
K_LIBREOFFICE_IMPRESS_031	Impress	Slide masters
K_LIBREOFFICE_IMPRESS_032	Impress	Outline to slides
K_LIBREOFFICE_IMPRESS_033	Impress	Import content
K_LIBREOFFICE_IMPRESS_034	Impress	Open presentation
K_LIBREOFFICE_IMPRESS_035	Impress	Impress controls
K_LIBREOFFICE_WRITER_001	Writer	Writer GUI actions
K_LIBREOFFICE_WRITER_002	Writer	Writer document context
K_LIBREOFFICE_WRITER_003	Writer	Writer save target
K_LIBREOFFICE_WRITER_004	Writer	Save & finish Writer edit
K_LIBREOFFICE_WRITER_005	Writer	Recover stalled Writer
K_LIBREOFFICE_WRITER_006	Writer	Enter Writer text
K_LIBREOFFICE_WRITER_007	Writer	Localized text replacement
K_LIBREOFFICE_WRITER_008	Writer	Precise text selection
K_LIBREOFFICE_WRITER_009	Writer	Preserve Writer content
K_LIBREOFFICE_WRITER_010	Writer	Text & paragraph formatting
K_LIBREOFFICE_WRITER_011	Writer	Find & Replace
K_LIBREOFFICE_WRITER_012	Writer	Paragraph styles
K_LIBREOFFICE_WRITER_013	Writer	Paragraph structure
K_LIBREOFFICE_WRITER_014	Writer	Build table
K_LIBREOFFICE_WRITER_015	Writer	Target table cells
K_LIBREOFFICE_WRITER_016	Writer	Table context & protection
K_LIBREOFFICE_WRITER_017	Writer	Fields & metadata
K_LIBREOFFICE_WRITER_018	Writer	Writer formulas

C Agent Prompts and GUI Protocol

C.1 Observation, Action, and Termination

The GUI policy interacts with the desktop environment through screenshots and the `computer_use` interface, which supports mouse, keyboard, and wait actions. In the Stage 1 SFT interaction protocol, spatial actions use normalized 0–999 coordinates, and the policy is instructed to act only on visible interface elements. The policy follows the prescribed interaction format using `<thinking>` and `<tool_call>` blocks for intermediate reasoning and executable actions.

The policy is instructed to issue a success termination only when it judges the task to be complete. Final task completion is determined separately by the task-specific evaluator. This separation prevents the learner's self-reported termination decision from being treated as sufficient evidence of task success.

In the saved RL configuration, each task is limited to at most 20 interaction turns, with a history-window configuration of 3, up to 512 generated tokens per step, and a sampling temperature of 1.0.

C.2 Agent Roles and Prompt Templates

RSI-CUA uses two orchestration agents in the recursive improvement loop. The *Capability Profiling Agent*, denoted by A_{prof}, analyzes the task specification, learner trajectory, and execution outcome to construct structured capability evidence. The *Curriculum Synthesis Agent*, denoted by A_{syn}, conditions on the resulting capability state and diagnosed gaps to generate and refine executable training tasks.

These two agents are distinct from the GUI policy being optimized. The GUI policy acts as the learner: it executes tasks in the desktop environment and produces the interaction trajectories used for capability profiling and policy optimization. Components used for benchmark construction and task-contract repair serve supporting roles in the data-generation and validation pipeline rather than constituting additional RSI agents.

Capability Profiling Agent. The Capability Profiling Agent corresponds to A_{prof} in the Method. It analyzes rule-labeled trajectories associated with the same task, contrasts successful and failed executions, and produces structured, evidence-grounded capability hypotheses. Each identified failure mode is grounded in concrete trajectory evidence and is accompanied by plausible alternative explanations, confidence, a falsifier, and targeted training-task recommendations. For presentation, the task-specific absolute dossier path is replaced with <DOSSIER_PATH>, and the prompt wording is normalized for consistency with the paper terminology while preserving its operational requirements and output schema.

```
# RSI-CUA Capability Profiling Agent

Analyze one group of rule-labeled GUI trajectories associated with the
    same task.

Read the compact trajectory dossier:

`<DOSSIER_PATH>`

The rule-evaluator outcomes are authoritative. Do not relabel task
    success or failure.

Compare successful and failed trajectories for the same task and
    identify evidence-grounded capability patterns. For each observed
    failure mode, formulate a falsifiable capability hypothesis in the
    following form:

> Under condition X, the learner tends to exhibit observable error Y,
    suggesting a potential gap in capability K.

Requirements:

1. Distinguish artifact completion from explicit task termination.
2. Ground every identified failure mode in concrete episode IDs and
    action-step evidence.
3. Contrast successful and failed trajectories; do not infer a
    capability gap from failures alone.
4. Distinguish learner behavior from possible infrastructure or
    evaluator failures. All included episodes are reward-valid, but note
     any remaining ambiguity.
5. Provide plausible alternative explanations for each capability
    hypothesis.
6. Provide a falsifier describing what future evidence would invalidate
     each hypothesis.
7. Recommend targeted training tasks that address the identified
    capability gaps; do not propose new profiling-benchmark items.
8. Avoid vague capability descriptions such as "bad reasoning" or "GUI
    weakness."
9. Return exactly one JSON object following the schema below.
```

```
{
  "schema_version": "openguigym.capability_hypothesis_analysis.v1",
  "group_id": "...",
  "task_summary": "...",
  "outcome_summary": {
    "valid_n": 0,
    "success_n": 0,
    "failure_n": 0,
    "success_rate": 0.0
  },
  "success_strategy": "...",
  "failure_modes": [
    {
      "id": "FM1",
      "condition_x": "...",
      "observed_error_y": "...",
      "capability_k": "...",
      "expected_action": "...",
      "evidence": [
        {
          "episode_id": "...",
          "steps": [0],
          "observation": "..."
        }
      ],
      "alternative_explanations": ["..."],
      "confidence": "low|medium|high",
      "falsifier": "..."
    }
  ],
  "training_task_recommendations": [
    {
      "targets_failure_mode": "FM1",
      "task_family": "...",
      "difficulty_control": "...",
      "rule_observable_outcome": "..."
    }
  ],
  "limitations": ["..."]
}

Return JSON only. Do not use Markdown and do not write files.
```

Listing 1: Normalized Capability Profiling Agent prompt for trajectory-grounded capability profiling; the task-specific dossier path is represented by a placeholder.

Curriculum Synthesis Agent. The Curriculum Synthesis Agent corresponds to A_{syn} in the Method. It generates executable training tasks conditioned on the learner's capability evidence and the stage-specific curriculum specification. The shared prompt template receives capability and difficulty controls together with evaluator and artifact-rendering constraints, and produces a rule-grounded task contract targeting the designated operational capability. Dynamic fields are populated from the active curriculum specification at run time.

```
# RSI-CUA Curriculum Synthesis Agent

Generate exactly one high-quality English GUI training task conditioned
     on the current learner and curriculum specification.

Inputs:

- Current curriculum specification: {SPEC_JSON}
- Implemented assertion catalog: {ASSERTION_CATALOG}
- Task-contract schema: {RULE_GROUNDED_TASK_SCHEMA}
```

```
- Deterministic artifact renderer: {ARTIFACT_RENDERER}

The curriculum specification summarizes measured capability evidence,
    task-role information, difficulty controls, and observed-boundary
    information for the current learner and curriculum stage.

Generate a novel executable training task that materially exercises the
     designated operational capability. Preserve the target capability
    while varying content, values, layout, wording, target identity,
    filename, and visual context. Do not copy a profiling-benchmark task
    .

## Capability and difficulty contract

1. Follow `task_variant.role`, `task_variant.role_contract`, `
    assigned_difficulty`, and the observed-boundary information provided
     in the curriculum specification.
2. The generated task must materially exercise `operational_definition
    `; a generic edit that bypasses the designated capability is invalid
    .
3. `simplified_boundary` must represent a genuinely small but
    consequential task, rather than a trivial file-existence task.
4. `perturbation_recovery` must encode the perturbation in the initial
    artifact. Evaluation remains based on the final artifact and must
    not claim to verify the learner's interaction path.
5. `bounded_composition` may contain at most two dependent semantic
    changes. Do not expand it into a long workflow.
6. Do not copy any representative task's instruction, coordinates,
    visible strings, values, filenames, or complete layout. Never expose
     raw pixel coordinates in the user instruction.

## Rule-grounded contract

1. Return exactly one `openguigym.rule_grounded_task.v2` JSON object;
    do not return a batch wrapper.
2. Set `curriculum_stage` to the active stage value provided in the
    specification, and set `task_id` exactly to the specification's `
    training_item_id`.
3. Copy `knowledge_evidence` exactly. Preserve `source_evidence` as the
     supplied one-item list.
4. Use a binary deterministic final-artifact reward: initial=0, golden
    =1, and every near miss=0.
5. Every success criterion must be covered by implemented assertions
    and targeted by at least one semantic near miss.
6. Provide at least three semantic near misses when possible, including
     wrong-target or collateral-damage cases when relevant. Do not use
    weak file-existence checks or LLM/VLM evaluators.
7. All task-facing and artifact-visible strings must be in English.
8. Use exactly the supported extension for each domain: GIMP `.png`,
    Calc `.xlsx`, Writer `.odt`, and Impress `.odp`.
9. The natural user instruction must explicitly name the exact `
    artifact.remote_output_path` and instruct the learner to Save As or
    Export to that path. The evaluator reads the delivered artifact
    rather than the open UI state.
10. Do not use `__GOLDEN_ARTIFACT__`, pixel equality against a hidden
    golden image, or any hidden reference artifact. Express the desired
    outcome through implemented semantic assertions.
11. For GIMP, the instruction must use the exact output verb `Export`;
    `Save As` alone is invalid.
12. Every criterion ID, including preservation criteria, must be
    targeted by at least one near miss.
13. The instruction must identify targets semantically using visible
    content, labels, values, or relative relationships. It must not
    expose evaluator internals, bounding-box values, raw coordinates, or
     rule IDs.
```

```
14. Return one complete, compact JSON object and no additional text.

## Renderer invariants

- GIMP: each `content_spec` must contain sibling top-level `canvas` and
     `objects` fields; never place `objects` inside `canvas`. Shapes are
     represented as flattened pixels, and rectangle/ellipse outlines are
     3 px.
- Calc: each `content_spec` must contain a top-level `sheets` list;
    each state represents a complete workbook. Formula results are not
    calculated, so formulas must be checked using formula assertions.
- Writer: each `content_spec` must contain top-level `paragraphs` and
    may optionally contain `title` and `tables`; embedded images are
    unsupported.
- Impress: each `content_spec` must contain a top-level `slides` list;
    each state represents a complete presentation.

The generated task must read as a natural user request about the
    desired final artifact, rather than as an evaluator specification,
    menu tutorial, or GUI-action script.

Return JSON only. Do not use Markdown and do not write files.
```

Listing 2: Curriculum Synthesis Agent prompt template for capability-targeted training-task construction.

GUI Learner Policy. The GUI learner is the policy optimized by RSI-CUA and is distinct from the Capability Profiling Agent and Curriculum Synthesis Agent. The following system message specifies the learner's computer-use interface and step-wise interaction protocol. It is a normalized snapshot extracted from Stage 1 SFT data, with tool-schema formatting standardized for readability while preserving the original interaction semantics.

```
You are a GUI agent that completes user tasks by interacting with a
    graphical desktop interface.

# Tools

You may use the following tool to inspect and interact with the visible
     GUI.

You are provided with function signatures within <tools></tools> XML
    tags:

<tools>
{
  "type": "function",
  "function": {
    "name_for_human": "computer_use",
    "name": "computer_use",
    "description": "Use a mouse and keyboard to interact with a desktop
     GUI and take screenshots.\n* Coordinates use a normalized 1000x1000
     virtual screen space, regardless of the screenshot's pixel size.\n*
     Use only visible GUI affordances from the current screenshot.\n*
    The environment may already be prepared before the first step, so
    inspect the screenshot before launching or switching apps.\n*
    Coordinates must be given in the 0-999 virtual screen range.\n* For
    clicks, place the cursor tip near the center of the target element.\
    n* For `left_click_drag`, `start_coordinate` is the start and `
    coordinate` is the end.\n* Only emit a tool call when the required
    arguments are present.",
    "parameters": {
      "type": "object",
      "properties": {
```

```
        "action": {
          "type": "string",
          "enum": [
            "key",
            "type",
            "mouse_move",
            "left_click",
            "left_click_drag",
            "right_click",
            "double_click",
            "triple_click",
            "scroll",
            "wait",
            "terminate",
            "answer"
          ],
          "description": "The single GUI action to execute. Action-
    specific fields are required as described below."
        },
        "keys": {
          "type": "array",
          "description": "Required by key, for example [\"ctrl\", \"s
    \"] or [\"enter\"]."
        },
        "text": {
          "type": "string",
          "description": "Required by type and answer."
        },
        "coordinate": {
          "type": "array",
          "description": "Required by click/move actions; [x, y] in the
     0-999 coordinate range."
        },
        "start_coordinate": {
          "type": "array",
          "description": "Required by left_click_drag."
        },
        "pixels": {
          "type": "number",
          "description": "Required by scroll."
        },
        "time": {
          "type": "number",
          "description": "Optional for wait."
        },
        "status": {
          "type": "string",
          "enum": ["success", "failure"],
          "description": "Required by terminate."
        }
      },
      "required": ["action"],
      "allOf": [
        {
          "if": {
            "properties": {
              "action": {"const": "key"}
            }
          },
          "then": {
            "required": ["keys"]
          }
        },
        {
          "if": {
```

```
            "properties": {
              "action": {"enum": ["type", "answer"]}
            }
          },
          "then": {
            "required": ["text"]
          }
        },
        {
          "if": {
            "properties": {
              "action": {
                "enum": [
                  "mouse_move",
                  "left_click",
                  "right_click",
                  "double_click",
                  "triple_click"
                ]
              }
            }
          },
          "then": {
            "required": ["coordinate"]
          }
        },
        {
          "if": {
            "properties": {
              "action": {"const": "left_click_drag"}
            }
          },
          "then": {
            "required": ["start_coordinate", "coordinate"]
          }
        },
        {
          "if": {
            "properties": {
              "action": {"const": "scroll"}
            }
          },
          "then": {
            "required": ["pixels"]
          }
        },
        {
          "if": {
            "properties": {
              "action": {"const": "terminate"}
            }
          },
          "then": {
            "required": ["status"]
          }
        }
      ]
    },
    "args_format": "Format arguments as a JSON object and satisfy the
    action-specific required fields. Use left_click rather than click."
  }
}
</tools>

# Response format
```

```
For every interaction step, output exactly two blocks in the following
    order:

1. A single <thinking>...</thinking> block describing the visible UI
    state, current task progress, relevant uncertainty, and the next
    action.
2. A single <tool_call>...</tool_call> block containing only the tool-
    call JSON:
   {"name": <function-name>, "arguments": <args-json-object>}.

Rules:

- Output exactly one <thinking> block followed by one <tool_call> block
    , and end with </tool_call>.
- Choose the action most likely to make meaningful progress toward
    completing the task.
- When uncertain, prefer an action that reveals useful information
    about the current UI state.
- Ground decisions in visible interface evidence.
- Use only visible GUI affordances from the current screenshot.
- Respect the required application and do not relaunch or switch to it
    when it is already open.
- Avoid repeating actions that produced no visible progress.
- Do not output any text outside the two required blocks.
- Use `action=answer` only when the task requires a textual answer.
- Use `action=terminate` with `status=success` only when the task is
    visibly complete.
- Use `action=terminate` with `status=failure` only when reasonable
    recovery options have been exhausted.
- Any response that does not follow this format is invalid.

The only valid response pattern is:

<thinking>I can see the target UI element, so I will take the next task
    -progressing action.</thinking>
<tool_call>{"name":"computer_use","arguments":{"action":"left_click","
    coordinate":[500,500]}}</tool_call>
```

Listing 3: Normalized GUI learner system-message snapshot extracted from Stage 1 SFT data.

Supporting Construction and Validation Prompts. In addition to the two RSI orchestration agents, RSI-CUA uses supporting prompts for benchmark construction and executable task validation. These prompts support different stages of the data-generation pipeline and are not treated as additional RSI orchestration agents.

GeneralCUABench Task Proposer. The GeneralCUABench Task Proposer constructs fixed profiling tasks from software knowledge. It is separate from the Curriculum Synthesis Agent, which generates stage-dependent training tasks.

```
# GeneralCUABench Task Proposer

Generate exactly one high-quality English GUI profiling task from a
    single software-knowledge item.

Read only the following local resources:

- Knowledge item: `{KNOWLEDGE_ITEM_PATH}`
- Research plan: `{RESEARCH_PLAN_PATH}`
- Rule-grounded design: `{RULE_GROUNDED_DESIGN_PATH}`
- Implemented artifact/assertion catalog: `{ASSERTION_CATALOG_PATH}`
- Task-contract schema: `{CONTRACT_SCHEMA_PATH}`
```

```
## Purpose

The generated task will become part of the fixed GeneralCUABench
    profiling benchmark.

The task must exercise the capability represented by the source
    knowledge while remaining a natural and executable user request.
    Convert process-oriented knowledge into a consequential, observable
    final-state or final-artifact requirement whenever possible.

Do not copy software documentation, menu instructions, or GUI
    procedures directly into the user instruction. The instruction
    should describe the desired final result rather than the interaction
     path used to obtain it.

## Design requirements

1. Use exactly one primary knowledge source. Do not combine unrelated
    knowledge items.

2. Preserve the capability represented by the source knowledge. When
    process-oriented knowledge must be converted into an artifact-level
    task, follow `taskability_review.capability_hypothesis`, `
    artifact_task_direction`, and `supported_assertion_families` as the
    reviewed design boundary. Do not replace the intended capability
    with a different one merely because it is easier to evaluate.

3. Do not claim that the evaluator verifies a particular click path,
    keyboard shortcut, selected tool, cursor state, or dialog state when
     only the final artifact is observable.

4. Construct a realistic, compact initial artifact containing enough
    surrounding content to expose wrong-target and collateral-damage
    errors.

5. The user instruction must specify every required semantic outcome
    and must not include requirements that cannot be verified by the
    implemented evaluator.

6. Preserve unrelated content whenever the instruction explicitly
    requires it to remain unchanged.

7. Use only renderer and assertion types implemented in the provided
    assertion catalog.

8. Set `rule_spec.aggregation` to `all`. The formal task reward must be
     binary.

9. Every success criterion and negative constraint must map to one or
    more implemented assertions. Every assertion's `criterion_ids` must
    contain all criteria enforced by that assertion.

   Do not duplicate a positive criterion as a mirror-image negative
    constraint. Prefer `negative_constraints: []` unless there is a
    distinct, independently asserted preservation or prohibition
    requirement.

10. Provide at least three semantic near misses when possible. Every
    criterion must be targeted by at least one near miss, and every near
     miss must identify all criteria that it violates.

11. The initial state must score 0, the golden state must score 1, and
    every near miss must score 0 under the declared assertions.
```

```
12. Do not use LLM/VLM judging, file-existence checks, non-empty-file
    checks, or other weak proxy rewards.

13. Include `title`, `curriculum_stage`, `difficulty`, `
    estimated_gui_steps`, `rationale`, `generalization_axes`, `
    novelty_notes`, `required_operations`, and `rewrite_notes`.

14. Set `curriculum_stage` to `fixed_diagnostic_400_v1`.

15. Add a top-level `knowledge_evidence` object copied from the source
    knowledge item containing:
    `knowledge_id`, `knowledge_title`, `documentation_source`, `
    selection_reason`, and `knowledge_item_path`.

16. Preserve `source_evidence` as a schema-compatible one-item list.
    Use the knowledge ID as `task_id`, use `knowledge://<domain>/<
    knowledge_id>` as `episode_dir`, and state that the task is derived
    from software knowledge rather than from a trajectory-level failure
    claim.

17. Use a task ID beginning with `diagnostic_400_v1_`.

18. All task-facing and artifact-visible text must be in English,
    including titles, instructions, rationales, criteria, constraints,
    required operations, rewrite notes, near-miss descriptions, and
    visible artifact content.

19. Source-language text may appear only in provenance fields inside `
    knowledge_evidence`.

20. Use exactly the supported artifact extension for each domain:
    - GIMP: `.png`
    - Calc: `.xlsx`
    - Writer: `.odt`
    - Impress: `.odp`

    The filename, remote input path, and remote output path must use
    the corresponding extension. `.xcf`, `.ods`, and `.pptx` are not
    supported by this rendering pipeline.

## Domain-specific constraints

### GIMP

- Evaluate the intended final image region or geometry while preserving
     unrelated pixels when required.
- Do not make a particular tool, menu, layer structure, sampling mode,
    or interaction path the target of evaluation.
- Express process-oriented requirements as observable final pixel
    outcomes.
- Each PNG `content_spec` must contain sibling top-level `canvas` and `
    objects` fields.
- `canvas` contains only dimensions, background, and mode; drawable
    objects belong in `content_spec.objects`.
- Rectangle and ellipse outlines are rendered with a width of 3 px.
- For uniform-color regions, use matching fill and outline colors or
    place assertions away from object borders.
- Transparency requires `canvas.mode: "RGBA"` and explicit alpha values
    .
- The renderer does not represent layers, opacity state, or compositing
     history; each state must directly specify the flattened final
    pixels.

### Calc
```

```
- Preserve formulas, values, sheet names, and used ranges whenever
    collateral edits are relevant.
- Each XLSX `content_spec` must contain a top-level `sheets` list.
- Each sheet must be specified as a complete worksheet state rather
    than as an edit script.
- Sorting operations must preserve complete rows.
- Store numeric JSON values as numbers rather than numeric strings.
- Assertion ranges must match the rendered worksheet and used range
    exactly.
- The renderer does not calculate formulas or populate cached formula
    results.
- Verify formulas using `calc_formula_equals` or `
    calc_formula_normalized_equals`; do not verify computed formula
    results with `calc_cell_equals`.
- Define number formats through `number_formats` and assert exact Excel
    /openpyxl format codes.
- Do not place `number_format` inside `formats`, use UI labels such as
    `Currency` or `Percent` as format codes, or represent an absent fill
     as `#NONE`.

### Writer

- Each ODT `content_spec` must contain top-level `paragraphs` and may
    optionally contain `title` and `tables`.
- Do not nest paragraphs or tables inside another object.
- Use exact block, text, table, and implemented text-style assertions
    when formatting is part of the target capability.
- Table, row, and column assertion indices are 1-based.
- `content_spec.title` creates an additional Heading 1 before `
    paragraphs`; omit it when the heading is already represented by the
    paragraph sequence, or include it consistently in exact structure
    assertions.
- Embedded images are unsupported by the compact fixture renderer.

### Impress

- Each ODP `content_spec` must contain a top-level `slides` list.
- Do not nest slides inside a deck or presentation object.
- Preserve slide order and slide text when required.
- Use the narrow assertion family appropriate to the reviewed
    capability.
- Supported properties include exact shape geometry and appearance,
    native tables, text styles, slide visibility, backgrounds,
    transitions, and ODF field objects.

## Output

Return exactly one JSON object conforming to `openguigym.
    rule_grounded_task.v2`.

Do not wrap the result in Markdown or a batch object.

Use only read-only file access when inspecting the listed resources. Do
     not modify files or execute validation or generation scripts.

Before returning the result, verify conceptually that:
- the initial state scores 0;
- the golden state scores 1;
- every near miss scores 0;
- every success criterion is assertion-covered;
- every success criterion is exercised by at least one near miss.

Return the JSON object only.
```

Listing 4: Normalized prompt for constructing fixed GeneralCUABench profiling tasks from software knowledge.

Task-Contract Repair. The Task-Contract Repair prompt is used during executable validation to revise generated task specifications that fail rendering or semantic rule validation while preserving their intended capability targets and evaluation contracts.

```
# RSI-CUA Task-Contract Repair

Repair exactly one generated rule-grounded GUI task after executable
    validation failure.

Inputs:

- Current task proposal: {PROPOSAL_JSON}
- Detailed validation result: {VALIDATION_RESULT_JSON}
- Rule-contract validator: {RULE_CONTRACT_VALIDATOR}
- Artifact renderer: {ARTIFACT_RENDERER}
- Assertion catalog: {ASSERTION_CATALOG}
- Semantic assertion implementation: {SEMANTIC_ASSERTION_IMPLEMENTATION
    }

The current task contract has passed schema validation but failed
    executable rendering or semantic rule validation.

Repair the contract so that the unchanged validation pipeline verifies:

- initial score = 0;
- golden score = 1;
- every near-miss score = 0.

## Repair requirements

1. Diagnose the exact renderer or assertion mismatch using the detailed
     validation results or exception information.

2. Use only implemented renderer fields, supported file types,
    implemented assertion types, and the actual indexing conventions of
    the assertion implementation.

3. Preserve the original knowledge source, natural English task intent,
     task ID, and `knowledge_evidence`.

4. If a formatting requirement is unsupported by the implementation, it
     may be simplified into the nearest observable editing outcome that
    still exercises the intended knowledge capability.

5. Do not weaken the evaluator into a file-existence check, non-empty-
    content check, or unrelated proxy reward.

6. Every success criterion must remain covered by implemented
    assertions and by at least one semantic near miss.

7. Ensure that every content specification is internally complete and
    can be rendered independently.

8. Preserve `artifact.remote_output_path` and keep the exact path
    visible in `task_instruction`.

9. Keep all task-facing and artifact-visible text in English. Source-
    language text may appear only inside provenance fields of `
    knowledge_evidence`.
```

```
10. Preserve the required artifact-delivery behavior:
    - GIMP tasks must explicitly instruct the learner to Export the
    completed PNG.
    - Calc, Writer, and Impress tasks must explicitly instruct the
    learner to save the completed document.
    - The evaluator reads the delivered artifact and does not
    automatically save the open application state.

## Renderer invariants

### GIMP / PNG

- Every state must contain sibling top-level `content_spec.canvas` and
    `content_spec.objects`.
- Never place `objects` inside `canvas`.
- Shapes are represented as flattened pixels; layers and tool state are
     unsupported.

### Calc / XLSX

- Every state must contain top-level `content_spec.sheets`.
- Each state represents a complete workbook.
- Formula results are not calculated, so verify formulas rather than
    computed values.

### Writer / ODT

- Every state must contain top-level `content_spec.paragraphs` and may
    optionally contain `title` and `tables`.
- Embedded images are unsupported.

### Impress / ODP

- Every state must contain top-level `content_spec.slides`.
- Exact assertions must match the independently rendered presentation
    state.

### Cross-domain validation

- Initial and golden states must differ.
- Every near miss must differ from the golden state in at least one
    assertion-observed field.

## Output

Return exactly one corrected `openguigym.rule_grounded_task.v2` JSON
    object.

Return JSON only. Do not use Markdown, do not write files, and do not
    include explanations outside the JSON.
```

Listing 5: Normalized task-contract repair prompt used during executable task validation.

D Stage Curricula and Training Data

D.1 SFT Corpus Size

Table 8 summarizes the serialized supervision samples in the cumulative SFT corpora used across RSI stages. Each JSONL row corresponds to one serialized supervision sample and should not be interpreted as a unique task or unique trajectory.

Table 8: Serialized SFT sample counts across RSI stages.

SFT corpus snapshot	JSONL samples
Stage 1 rejection-sampled SFT	1,858
Stage 2 cumulative SFT	10,565
Stage 3 cumulative SFT	13,037

Table 9: Summary of GRPO curriculum schedules across RSI stages.

Stage	Source Tasks	Slots	Unique Tasks	Repeated Slots
Stage 1	815	4,096	815	3,281
Stage 2 cumulative	1,773	4,096	1,764	2,332
Stage 3 final RL	2,173	1,600	1,362	238

The Stage 2 and Stage 3 corpora are cumulative and contain training experience retained from earlier stages together with newly collected supervision. Consequently, the corpus sizes in Table 8 represent cumulative serialized sample counts rather than the number of newly added samples at each stage.

D.2 GRPO Curriculum Schedule

The online GRPO curriculum is constructed as a scheduled sequence of tasks sampled from the accumulated training pool. A task may appear multiple times in a schedule; therefore, the number of scheduled slots can exceed the number of unique tasks.

Stage 1 and Stage 2 allocate their planned curriculum slots uniformly across the four applications. Stage 3 allocates 200 reachable and 200 exploratory slots to each application, yielding 1,600 scheduled training slots in total. The resulting schedules combine tasks from reachable and exploratory task pools while retaining previously accumulated training experience.

E Model Training and Runtime Configuration

E.1 Initialization and Optimization

The shared base policy is Qwen3-VL-8B-Instruct. At each RSI stage, the learner is initialized from the shared base policy and optimized on the cumulative curriculum constructed for that stage. Training consists of rejection-sampling supervised fine-tuning followed by online reinforcement learning with GRPO:

$$\pi_{\text{base}} \xrightarrow{\text{RS-SFT}(\mathcal{D}_t)} \pi_t^{\text{SFT}} \xrightarrow{\text{GRPO}(\mathcal{D}_t)} \pi_t.$$

Thus, recursive information is transferred across stages through capability profiling and the accumulated training curriculum rather than through direct continuation from the previous stage's policy parameters.

E.2 SFT Configuration

Table 10 summarizes the common optimization settings recorded for rejection-sampling SFT.

Rejection-sampling SFT is performed on successful interaction trajectories collected from the cumulative curriculum. These trajectories provide supervised action sequences that initialize the policy before online GRPO optimization.

E.3 GRPO Configuration

Table 11 summarizes the core optimization and rollout settings used for online GRPO.

Table 10: Core rejection-sampling SFT configuration.

Setting	Value	Setting	Value
Optimizer	Adam	Learning rate	1×10^{-6}
LR schedule	Cosine	Minimum learning rate	1×10^{-6}
Warmup fraction	0.1	Weight decay	0.1
Training epochs	1	Precision	BF16
Rollout batch size	64	Global batch size	64
World size	8	Parallel topology	TP=4, DP=2, PP=1
Sequence parallelism	Enabled	Dynamic batch sizing	Enabled
Loss masking	Qwen3 loss mask		

Table 11: Core GRPO optimization and rollout configuration.

Setting	Value	Setting	Value
Physical GPUs	8	Actor / rollout GPUs	8 / 8, colocated
Parallel topology	TP=4, DP=2, PP=1	Rollout batch	16 prompts
Candidates / retained	10 / 8	Global batch	128
Concurrent GUI episodes	160	Optimizer / LR	Adam / 1×10^{-6}
LR schedule / weight decay	Constant / 0.1	Adam β_1/β_2	0.9 / 0.98
GRPO iterations	1	PPO clip / high clip	0.20 / 0.28
KL coefficient	0	Entropy coefficient	0
Gradient clipping	1.0	Rollout temperature	1.0
Maximum turns	20	History window	3
Maximum tokens per step	512	Maximum tokens per GPU	8,192
Precision	BF16	Checkpoint interval	Every 10 steps

For each rollout batch, 16 task prompts are processed with up to 10 candidate rollouts per prompt, corresponding to as many as 160 concurrent GUI episodes. Up to eight candidates are retained for optimization according to the GRPO rollout configuration.

E.4 Runtime and Checkpointing

The training runtime supports automatic retries for infrastructure-level execution failures and preserves episode artifacts generated during interaction. Checkpoints are periodically saved during optimization to support model recovery and subsequent evaluation.

F Additional Experimental Results

F.1 GeneralCUABench Pass@1

Table 12 reports pass@1 performance on GeneralCUABench for the stage-wise models and ablation variants. Since GeneralCUABench contains 100 tasks for each of the four applications, the application-level percentages directly correspond to the number of successfully completed tasks.

Across the three recursive stages, GeneralCUABench pass@1 improves from 3.25% for the base model to 8.50%, 11.25%, and 17.75% after Stages 1–3, respectively. The application-level results further show that the improvements are distributed across multiple desktop applications rather than being restricted to a single environment.

F.2 OSWorld Transfer Results

We evaluate the base model and the stage-wise RSI-CUA policies on the held-out OSWorld benchmark using a common evaluation protocol. OSWorld is excluded from capability profiling, curricu-

Table 12: Pass@1 results on GeneralCUABench (%).

Model / Stage	Overall	GIMP	Calc	Impress	Writer
Qwen3-VL-8B-Instruct	3.25	0.00	7.00	3.00	3.00
Stage 1 SFT Only	5.25	1.00	12.00	3.00	5.00
Stage 1 GRPO Only	5.00	0.00	13.00	2.00	5.00
Stage 1 Random Curriculum	7.50	2.00	16.00	9.00	3.00
RSI-CUA^{S1}	8.50	3.00	22.00	4.00	5.00
Stage 2 Sequential SFT	4.25	2.00	8.00	4.00	3.00
Stage 2 Cumulative SFT	5.75	1.00	14.00	6.00	2.00
Stage 2 Sequential SFT+GRPO	9.00	1.00	22.00	7.00	6.00
RSI-CUA^{S2}	11.25	1.00	24.00	11.00	9.00
Stage 3 SFT Only	7.00	1.00	14.00	8.00	5.00
RSI-CUA^{S3}	17.75	5.00	35.00	18.00	13.00

Table 13: Stage-wise task completion rate on the held-out OSWorld benchmark (%).

Checkpoint	Task Completion Rate
Qwen3-VL-8B-Instruct	18.06
RSI-CUA^{S1}	22.50
RSI-CUA^{S2}	23.06
RSI-CUA^{S3}	25.28

lum construction, and policy optimization, and is used only for evaluating transfer beyond the RSI training loop.

OSWorld task completion increases from 18.06% for the base model to 22.50%, 23.06%, and 25.28% after the three RSI stages, respectively. These results show that the improvements observed on the fixed profiling benchmark are accompanied by improved performance on a held-out computer-use benchmark.

G Trajectory Case Study

G.1 Stage-Wise GIMP Trajectory Case Study

We illustrate stage-wise behavior using a GeneralCUABench task that requires horizontally flipping an image. The internal task identifier is `diagnostic_400_v1_gimp_1172_flip_horizontal`. The agent must horizontally mirror the complete image, preserve its dimensions and top header, and export the resulting image as a PNG. The task therefore requires both successful image manipulation and correct artifact persistence.

The task evaluator verifies properties of the exported artifact, including the requested horizontal transformation, image dimensions, and preservation of the header.

Figure 7 compares representative trajectories from the base policy and the three RSI stages. The Base trajectory illustrates an important distinction between performing the requested visual operation and completing the full task: the canvas reaches the horizontally mirrored state, but the required output artifact is not successfully persisted before the episode terminates. In contrast, the representative trajectories from later stages complete both the transformation and artifact production.

Across eight valid rollouts per stage, SR_8 improves from 0/8 for the Base policy to 6/8, 8/8, and 8/8 for RSI-CUA^{S1}, RSI-CUA^{S2}, and RSI-CUA^{S3}, respectively..

The Base trajectory further illustrates that task-level evaluator failure does not necessarily imply failure of every constituent capability. In this example, the requested horizontal transformation is visibly completed, while the remaining failure occurs at artifact persistence and task completion. This

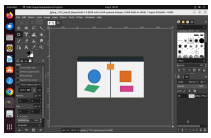
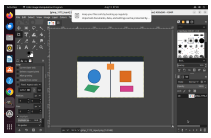
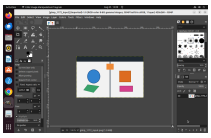
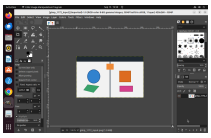
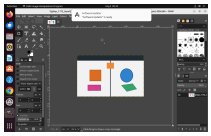
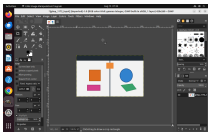
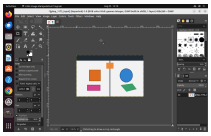
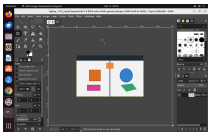
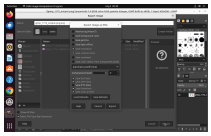
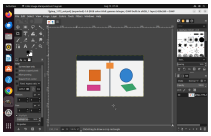
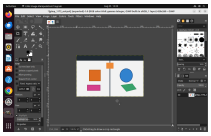
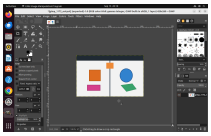
	Base	**Stage 1**	**Stage 2**	**Stage 3**
SR_8	0/8	6/8	8/8	8/8
Initial				
After flip				
Episode end				

Figure 7: Stage-wise trajectories on the same GIMP horizontal-flip task. Rows show the initial state, the state after the flip operation, and the episode endpoint. The Base policy completes the visual transformation in the representative rollout but does not successfully persist the required output artifact, whereas the later-stage examples complete both the edit and artifact production. The repeated-rollout success rate SR_8 improves from 0/8 for the Base policy to 6/8, 8/8, and 8/8 across RSI-CUA Stages 1–3.

distinction motivates the trajectory-grounded capability profiling used in RSI-CUA, which separates successfully exercised capabilities from capabilities that are attempted but fail or remain unreached.