CUAEVOLVE: SELF-IMPROVING COMPUTER-USE AGENTS FOR EVOLVING APPLICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents (CUAs) can solve increasingly complex, long-horizon tasks, but application updates and evolving workflows can erode their acquired capabilities. Maintaining up-to-date training data requires frequent collection and curation, a costly and time-consuming process that is difficult to sustain. In this work, we introduce CUAEvolve, a computer-use agent that adapts its policy and reinforcement learning (RL) curriculum to application and workflow evolution through a self-improving loop. To improve robustness to visual and interface changes, CUAEvolve interleaves GUI actions with executable code that can consolidate multiple GUI operations into fewer coding sequences and bypass visual grounding when programmatic interfaces remain compatible across updates. To reduce environment data curation as applications and workflows evolve, we introduce a self-improving RL loop that uses failure diagnoses to synthesize targeted task instructions, initial environment states, and evaluators at scale, enabling continual policy improvement. To measure performance on evolved CUA tasks, we introduce OSEvolveBench, which pairs original tasks with application and workflow changes. Concretely, 1) application variants modify one interface property with surface appearance, geometry, naming, or interaction; 2) workflow variants revise user instructions and introduce previously unseen workflows. Experimental results show that the self-improving RL loop raises task scores on evolved applications and workflows to 25.0% and 22.3%, outperforming prior open-source CUAs at comparable scales. CUAEvolve also shows strong retention and cross-platform generalization, achieving 52.7% and 46.7% on OSWorld-Verified and WindowsAgentArena.

1 INTRODUCTION

Recent advances in multimodal agents have enabled computer-use agents (CUAs) to complete increasingly complex, long-horizon tasks across desktop applications (Qin et al., 2025; Wang et al., 2025; Xue et al., 2026). By automating everyday digital work, these agents can reduce the manual effort required to operate desktop applications (Wu et al., 2024; Bonatti et al., 2025). However, maintaining these capabilities requires adaptation to changes in both application interfaces and user workflows. Such changes can invalidate learned execution procedures and erode previously acquired capabilities (Sun et al., 2026a; Fan et al., 2026).

CUA tasks evolve in two fundamental aspects (Figure 1): a) Application updates can alter surface appearance, geometry, naming, and interaction sequences. A relocated control, revised label, or additional confirmation step can disrupt a previously successful CUA model even when the user's objective remains unchanged. b) Workflows and user preferences evolve as new workflows provide alternative ways to complete tasks. In fixed environments and task distributions, the learned procedures become outdated and do not establish an ability to improve as tasks evolve. However, the scarcity of up-to-date training data limits continual CUA adaptation, as keeping pace with rapidly evolving applications and workflows requires frequent data collection that is costly and time-consuming.

To support continual adaptation to application updates and workflow evolution, we introduce CUAEvolve, a computer-use agent that adapts its policy and reinforcement learning (RL) curriculum (Dennis et al., 2020; Jiang et al., 2021; Parker-Holder et al., 2022) to application and workflow evolution through a self-improving loop. To improve robustness to visual and interface changes,

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.

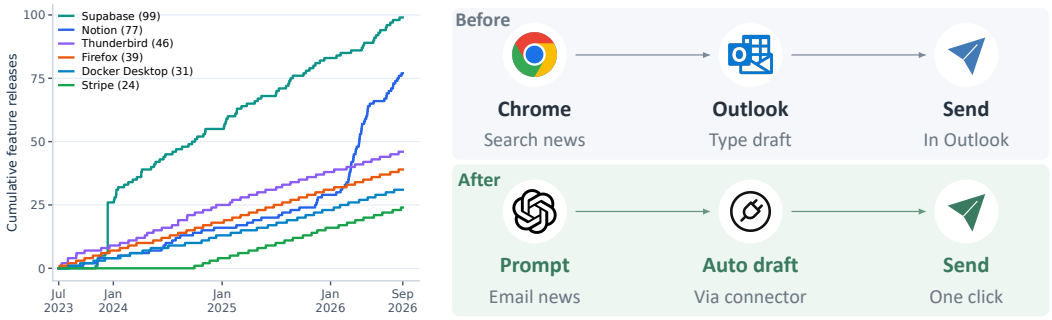


Figure 1: (a) Cumulative feature-release counts for six applications, July 2023–August 2026 (sources and counting rules in Appendix A.1). (b) An example of workflow evolution: sharing news via email shifts from searching the web and manually drafting messages in Outlook to retrieving news in ChatGPT and sending emails through app connectors.

we train the model on interleaved GUI and coding trajectories, enabling it to consolidate multiple GUI operations into fewer programmatic actions. To reduce environment data curation as applications and workflows evolve, we introduce a self-improving RL loop that uses failure diagnoses to synthesize executable tasks and environments at scale: 1) a diagnoser analyzes training trajectories, execution feedback, and task outcomes to identify failure modes; 2) a curriculum generator uses these diagnoses to construct application and workflow evolution RL tasks. Application variants modify interface surface appearance, geometry, naming, and interaction sequences from original task pools, and workflow variants revise instructions, deliverables, or introduce new workflows. Admitted variants update the task pool and sampling distribution for subsequent RL rounds, allowing the policy and its training curriculum to evolve together; 3) the admission gate evaluates the task in the environment to verify solvability and confirm that the application or workflow changes are clearly observable.

To evaluate CUA performance under task evolution, we introduce OSEvolveBench, a benchmark that organizes original and evolved task conditions into two evaluation families. The application-evolution spans 31 mock web applications and varies four interface properties: surface appearance, geometry, naming, and interaction. The workflow-evolution introduces novel user instructions and deliverables, as well as previously unseen workflows. Experimental results show that the self-improving RL loop raises CUAEvolve's scores on evolved applications and workflows to 25.0% and 22.3%, outperforming prior open-source CUAs at comparable scales. It also maintains strong retention performance on OSWorld-Verified (Xie et al., 2024) and WindowsAgentArena (Bonatti et al., 2025), achieving task scores of 52.7% and 46.7%, respectively. Our contributions are as follows:

- We introduce CUAEvolve, a computer-use agent that leverages interleaved GUI and coding trajectories with robustness towards visual and interface changes and reduces execution steps.
- We develop a self-improving RL loop that translates failure diagnoses into executable application and workflow variants, validates their task and evaluator consistency, and incorporates them into an evolving training curriculum.
- We introduce OSEvolveBench, a benchmark to measure performance on evolved tasks, covering application and workflow evolutions and changes.

2 Related Work

2.1 Task and Environment Synthesis

Task and environment synthesis adapts an agent's training distribution to its evolving capabilities. Regret-based environment design constructs curricula through environment generation and iterative editing, as demonstrated by PAIRED and ACCEL (Dennis et al., 2020; Parker-Holder et al., 2022). RLAnything (Wang et al., 2026b) jointly updates the policy and reward model and uses critic feedback to revise training environments. For computer use, CUA-Gym (Wang et al., 2026a) synthesizes

task instructions, environment states, and executable reward programs through independent reward construction and execution-based validation. Liu et al. (2026) expand cross-platform supervision through a data pipeline combining automated agents and human expertise. UI-CUBE evaluates systematic interface variations, multiple resolutions, and complex workflows through application-state verification. In this work, CUAEvolve constructs task and environment synthesis to explicitly address application and workflow evolution, which has not been studied in prior work.

2.2 Self-Improving Computer-Use Agents

Computer-use agents improve through interaction experience and hybrid trajectories. SEAgent (Sun et al., 2025) uses autonomous software exploration, step-level assessment, and curriculum generation to acquire and consolidate application-specific experience. Xue et al. (2026) further incorporate policy-aware task filtering and adaptive curricula into online RL. Sun et al. (2026b) adapts the agent at inference time, and diagnoses unsuccessful trajectories and generates code patches with lightweight human verification without additional policy training. Hybrid trajectory methods, such as using tools (Hu et al., 2026), learn to coordinate graphical actions and tools through synthesized hybrid trajectories and staged training, and Li et al. (2026) evaluates long-horizon GUI-CLI coordination with trajectory-aware judging to detect shortcuts missed by outcome-only evaluation. In this work, CUAEvolve adapts GUI-Coding policy and RL curriculum to application and workflow evolution through a self-improving loop, where failures observed in rollouts guide the task and environment construction. The curriculum generator and discriminator admit variants that update the RL task pool, allowing the policy and training curriculum to evolve together.

3 OSEvolveBench

Self-improving computer-use agents continually refine their capabilities through interaction with their environments (Sun et al., 2025; Xue et al., 2026). However, performance on the predefined task suites of existing CUA benchmarks (Xie et al., 2024; Bonatti et al., 2025) does not directly characterize adaptation to application updates and evolving workflow requirements. To evaluate these capabilities, we introduce OSEvolveBench, which pairs base tasks with evolved variants across two families: *application evolution* and *workflow evolution*. We construct the base task pool for OSEvolveBench by selecting tasks from CUA-Gym (Wang et al., 2026a), AgentNet (Wang et al., 2025), and ScaleCUA (Liu et al., 2026).

Application evolution. Application evolution modifies one interface property from the base task, preserving objective, target content and evaluator. We define four axes (Figure 2): **surface appearance (S)** changes visual styling, including colors, typography, and decorative elements; **geometry (G)** changes the positions or arrangement of interface elements; **naming (N)** changes control and menu labels without changing their functionality; and **interaction (I)** changes the required interaction sequence, instantiated through an additional confirmation step before committing an operation.

We implement these axes on 31 React-based mock applications by sampling edits within a designated axis. Evaluators assess the resulting application state independently of the rendered interface, allowing interface variants to share the same success criteria. Validation checks the intended change in rendered screens and in executed action sequences. Each admitted variant is paired with its unchanged source task. Implementation details and examples are provided in Appendix A.2.

Workflow evolution. Workflow evolution changes task requirements or application composition, where we construct two types of variants. *Revised cross-application workflows* introduce explicit user instructions or preferences that change the required deliverables, where we sample combinations of 2-5 applications and retain combinations absent from the reference evaluation suites. *Novel and unseen workflows* curate new workflows and the application functions required to execute them. For paired evaluation, a workflow variant is associated with its source task and evaluated against its own task specification. Construction procedures are described in Appendix A.4.

Statistics and Human Audit. The application-evolution track contains 99 admitted source tasks and 386 evolved variants: 98 surface, 99 geometry, 98 naming, and 91 interaction variants. Figure 3

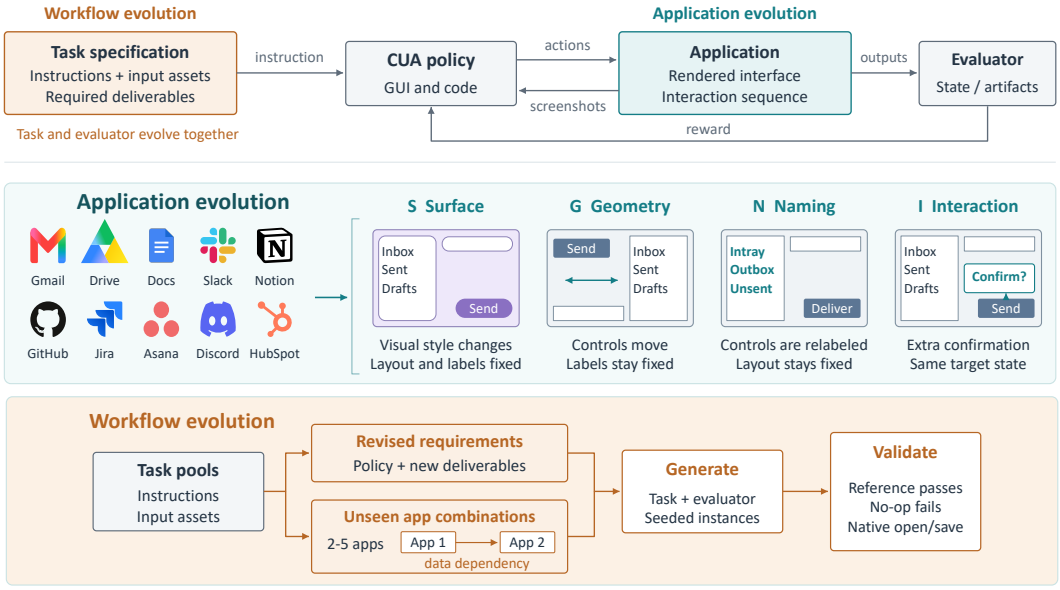


Figure 2: OSEvolveBench dataset construction. Top: application and workflow families of the evolution loop. Middle: application evolution samples surface, geometry, naming, and interaction changes. Bottom: workflow evolution supply source tasks and assets for revised requirements and unseen workflow combinations.

summarizes application domains and workflow compositions, and Appendix A.6 reports the per-application counts.

To complement automated validation, we manually review a random 50% of generated tasks. Annotators assess (i) *evaluator validity*: preservation of task and evaluators for application variants, and alignment with revised requirements for workflow variants; (ii) *task solvability*: a reference solution achieves full credit using available inputs in the deployed environment; (iii) *evolution validity*: the intended change is observable and non-target properties remain unchanged; and (iv) *application integration*: deliverables can be run and saved on a real machine. Approximately 92% of audited tasks pass all four criteria, where failed samples are discarded. Detailed admission criteria are provided in Appendix A.5.

4 CUAEvolve

CUAEvolve couples a unified GUI–CLI policy with an evolving training curriculum to support adaptation to application updates and changing workflow requirements. Supervised fine-tuning (SFT) initializes the policy to interleave GUI actions and executable code. During subsequent reinforcement learning (RL), we introduce a self-improving RL loop that uses failure diagnoses to synthesize executable tasks and environments at scale for identified weaknesses (Figure 4).

The self-improving RL loop leverages an orchestrator to coordinate four components: 1) The *CUA model* executes GUI actions and executable code through a single policy with SFT and RL; 2) The *diagnoser* analyzes training rollouts and task outcomes to identify failure modes; 3) The *task generator* uses these diagnoses to synthesize application and workflow-evolved tasks for RL training, allowing the policy and its training curriculum to evolve together. 4) the admission gate follows discriminator validation (Wang et al., 2026a) before entering training pools.

4.1 CUA Policy Model

CUAEvolve uses the policy $\pi_\theta(a_t \mid h_t, g)$ over GUI and coding actions, where g denotes the task instruction and h_t contains screenshots, previous actions, and execution feedback. GUI actions include pointer, keyboard, and interface-control operations, and coding actions execute policy-generated Python or Bash in the task environment. The resulting exit status, standard output, and standard error are returned as observations that condition subsequent decisions. Coding actions improve ro-

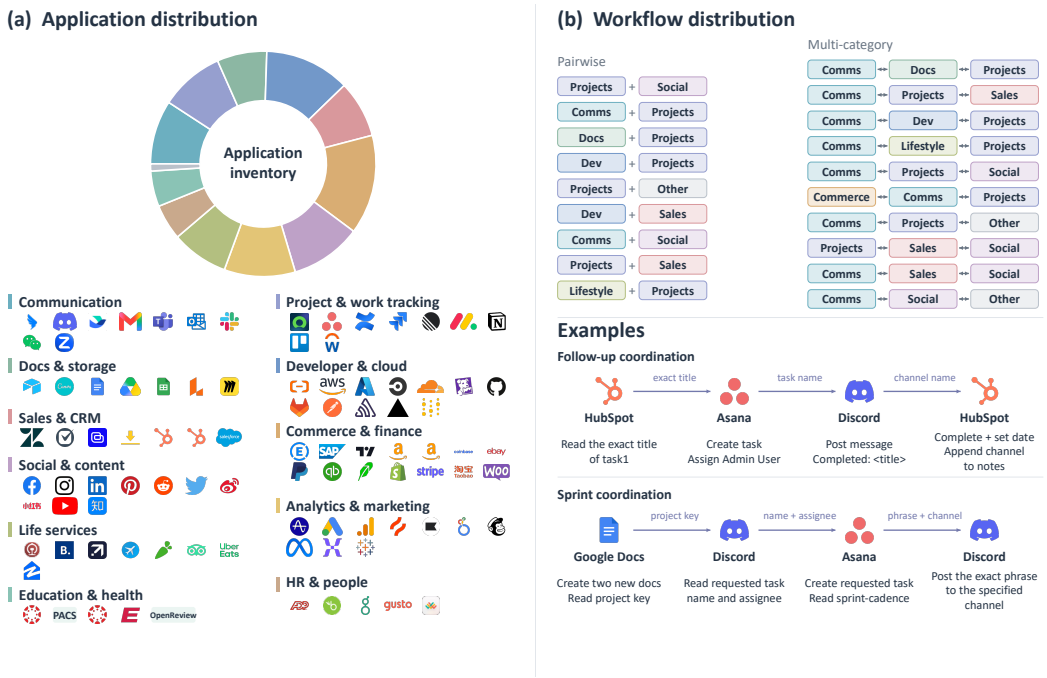


Figure 3: Application and workflow distributions. (a) The application inventory. (b) Examples of cross-app workflows. HubSpot → Asana → Discord → HubSpot; Google Docs → Discord → Asana → Discord.

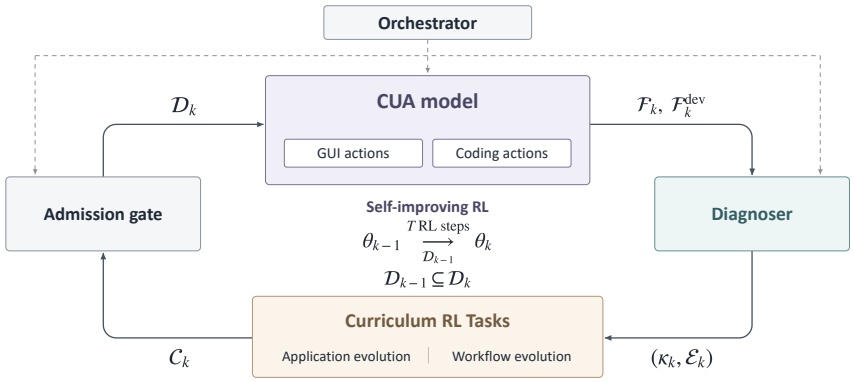


Figure 4: CUAEvolve self-improving RL loop. The orchestrator coordinates the CUA policy, diagnoser, curriculum generator, and admission gate discriminator to couple policy optimization with feedback-driven task adaptation.

bustness to visual and interface changes, and provide alternative execution paths through compatible file formats, APIs, and command-line interfaces. A single coding action can also execute several operations that would otherwise require separate GUI actions, reducing the total length of CUA trajectory. Complete actions are provided in Appendix B.1.

For SFT warmup, we construct interleaved trajectories using multimodal large language models (MLLMs): 1) *online synthesis* generates task-conditioned code from the current environment and executes it in a sandbox, collecting GUI actions, coding actions, and their observed outcomes; 2) *offline transformation* rewrites selected segments of existing GUI-only demonstrations as programmatic operations, without requiring concrete implementations or sandbox execution at this stage. These routes cover operations of varying granularity, from individual utilities to procedures combining multiple operations, and tasks with different requirements for graphical and programmatic interaction. For executed candidates corresponding to the same task, we retain the trajectory with the highest score and break ties by selecting trajectory with fewer steps.

Algorithm 1 CUAEvolve: self-improving policy and curriculum updates

Input: SFT policy π_{θ_0}; seed pool $\mathcal{D}_0$ with distribution μ_0; Φ, Ψ, Admit; total training steps T; generation steps $\mathcal{U} \subseteq \{1, \dots, T-1\}$; B tasks per step, n rollouts per task; $\mathcal{H}_0 \leftarrow \emptyset$.
for $s = 1, \dots, T$ **do**
  RL step: from θ_{s-1}, apply one update of Eq. (3) on B tasks from μ_{s-1}, n rollouts each; obtain θ_s and record the step's rollouts $\mathcal{F}_s$. $\mathcal{D}_s \leftarrow \mathcal{D}_{s-1}$; $\mu_s \leftarrow \mu_{s-1}$; $\mathcal{H}_s \leftarrow \mathcal{H}_{s-1}$.
  if $s \in \mathcal{U}$ **then**
    Rollout Diagnose: update $\hat{p}_q$ from the valid rollouts in $\mathcal{F}_s$; $\kappa_s \leftarrow \Phi(\mathcal{F}_s, \{\hat{p}_q\})$.
    Task Generate: $\mathcal{C}_s \leftarrow \Psi(\mathcal{D}_{s-1}, \{\hat{p}_q\}, \kappa_s, \mathcal{H}_{s-1})$.
    Admission Gate: $\mathcal{A}_s \subseteq \{c \in \mathcal{C}_s : \mathrm{Admit}(c) = 1\}$ with $|\mathcal{A}_s| \le M_{\mathrm{add}}$; $\mathcal{D}_s \leftarrow \mathcal{D}_{s-1} \cup \mathcal{A}_s$.
    Resample: $\mu_s \leftarrow \mathrm{Update}(\mathcal{D}_{s-1}, \mathcal{A}_s, \{\hat{p}_q\})$; $\mathcal{H}_s \leftarrow \mathcal{H}_{s-1} \cup \{(\kappa_s, \mathcal{C}_s, \mathcal{A}_s)\}$.
  end if
end for
return π_{θ_T} and the records $\mathcal{F}_{1:T}$ and $(\kappa_s, \mathcal{C}_s, \mathcal{A}_s, \mu_s)_{s \in \mathcal{U}}$.

4.2 DIAGNOSER

The diagnoser uses trajectory-based error analysis to connect policy failures with curriculum updates (Xue et al., 2026; Wang et al., 2026b). During online RL, it analyzes training trajectories, execution feedback, and terminal outcomes, aggregating failure statistics by task family and application domain. Application-evolution failures are organized by surface appearance, geometry, naming, and interaction, following which the curriculum generator conditions on the diagnoses to synthesize targeted tasks and guide task sampling for subsequent RL training.

4.3 CURRICULUM RL TASKS

Advantage and objective. At each RL step, we sample B tasks and collect n rollouts per task using the behavior policy $\pi_{\theta_{\mathrm{old}}}$. Let $\mathcal{I}_q$ denote the valid rollout indices for task q, with binary terminal rewards r_i assigned by V. Following the reward-centering approach (Liu et al., 2025), we compute advantages without group-standard-deviation normalization:

$$\hat{A}_i = r_i - \frac{1}{|\mathcal{I}_q|} \sum_{j \in \mathcal{I}_q} r_j, \qquad i \in \mathcal{I}_q. \tag{1}$$

Each interaction turn forms a training sample and inherits its trajectory-level advantage. We optimize a PPO clipped objective (Schulman et al., 2017), averaged over loss-bearing tokens across the batch. The objective, masking rules, and handling of invalid rollouts are detailed in Appendix B.2.

Curriculum task synthesis. The curriculum generator Ψ constructs candidate tasks from the current pool $\mathcal{D}_{k-1}$, empirical success estimates $\{\hat{p}_q\}$, training records $\mathcal{F}_k$, diagnoses κ_k. Feedback-driven environment editing enables curricula to respond to agent capabilities (Parker-Holder et al., 2022; Wang et al., 2026b); CUAEvolve applies this mechanism to application and workflow evolution with distinct construction and validity constraints.

Application-evolution candidates modify one axis in $\mathcal{M} = \{S, G, N, I\}$: surface appearance, geometry, naming, or interaction. They preserve the source instruction, target content, and terminal evaluator. Workflow-evolution candidates revise instructions and deliverables or introduce new workflow combinations with required information dependencies. Success estimates guide difficulty adjustments toward harder variants for consistently solved tasks and simpler conditions for persistently unsuccessful tasks.

Task admission gate. Following discriminator validation (Wang et al., 2026a), candidates should pass the agent's admission gate before entering training, consistent with the need for executable task specifications and reward checks in computer-use task synthesis.

The discriminator validation maintains disjoint training and OSEvolveBench's task instances and audit their underlying assets for overlap, including input files, document contents, and task-specific application states. We hold out five applications: Asana, ChatGPT, Claude, Twitter, and GitLab exclusively for evaluation. Failure diagnosis and curriculum generation operate exclusively on training tasks and their rollouts. Newly generated training tasks undergo the overlap check before admission

to the training pool. Applications and task categories may be shared across splits, while evaluation tasks use separate instances and assets.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Dataset. For SFT warm-up, we curate 4,300 GUI-only and interleaved GUI–Coding trajectories using the construction procedure in Section 4.1. We combine these trajectories with public training data from AgentNet (Wang et al., 2025) and ScaleCUA (Liu et al., 2026), which provide supervision for visual grounding, navigation, and task planning. This mixture combines broad coverage of GUI interaction, code and transitions between GUI and coding actions.

For evaluation, we use OSEvolveBench (Section 3) to assess performance on original and evolved task conditions. We evaluate on OSWorld-Verified (Xie et al., 2024) to assess retention of desktop task capabilities and on WindowsAgentArena (Bonatti et al., 2025) to examine cross-platform performance. We maintain disjoint training and evaluation task instances and assets following the separation protocol described in Section 4.3. Following the fine-grained evaluation principle (Yuan et al., 2026), we measure partial task completion on OSEvolveBench on the final application state and output artifacts. Each task receives a normalized score in $[0, 1]$ according to its evaluator's predefined scoring rules.

Implementation details. We initialize CUAEvolve from Qwen3-VL-32B (Bai et al., 2025), and perform SFT for two epochs with a peak learning rate of 1×10^{-5}, followed by cosine decay to 1×10^{-6}. The online RL runs 130 training steps with constant learning rate of 10^{-6}. Each step samples $B = 32$ tasks from μ_{s-1}, collects $n = 8$ rollouts per task, and applies one policy update. The system resolution is 1920×1080. For our evaluations, we use 366 of the 369 OSWorld-Verified tasks and 137 of the 141 tasks in our WindowsAgentArena evaluation set, respectively excluding three Google Drive tasks and four clock tasks due to API and system image limitations. Temperature is set to 0 for deterministic decoding during evaluation. Training-data composition, optimizer settings, task-sampling rules, and evaluation configurations are detailed in Appendix B.2.

Baselines. We compare CUAEvolve with general-purpose models and specialized computer-use agents. General-purpose baselines include Qwen3.8 (Qwen Team, 2026), Claude Opus 5 (Anthropic, 2026), GPT-5.6 Sol (OpenAI, 2026), and Gemini 3.5 Flash (Google DeepMind, 2026). Specialized baselines include UI-TARS-1.5-7B (ByteDance Seed Team, 2025), OpenCUA-32B (Wang et al., 2025), EvoCUA-32B (Xue et al., 2026), and GUI-Owl-1.5-32B-Instruct (Xu et al., 2026). To assess the effectiveness of self-improving RL loop, we apply it to EvoCUA-32B using the same initial task pool and matched RL settings, and compare its performance before and after adaptation.

5.2 MAIN RESULTS

Performance on application and workflow evolution Table 1 reports mean task scores and application and workflow families are reported separately. On application evolution, RL raises the our model from 19.0 after SFT to 25.0 with an absolute gain of 6.0 points, also raising the base score from 24.3 to 28.3. On workflow evolution, the score increases from 18.5 after SFT to 22.3 after RL with a gain of 3.8 points. The application gains differ in size across evolution axes. Scores improve on all four axes: S (20.3 to 28.9), G (19.7 to 24.8), N (18.6 to 24.0), and I (17.3 to 22.1), with the smallest gain on I (4.8 points). Both scores increase after RL, and the base–evolved gap narrows slightly (5.3 after SFT and 3.3 after RL).

Performance on cross-domain and cross-platform tasks Table 2 compares the SFT and RL models on OSWorld-Verified. Following online RL on application and workflow-evolution tasks, CUAEvolve achieves 52.7% success on OSWorld-Verified, compared with 46.7% for the SFT with a gain of 6.0 points. indicating that adaptation preserves performance on the original desktop benchmark.

To assess transfer beyond the RL environments, we further evaluate CUAEvolve on WindowsAgentArena for Windows platform computer-use. The policy achieves 46.7% task success

Table 1: Performance on OSEvolveBench. For application evolution, S, G, N, and I denote changes in surface appearance, geometry, naming, and interaction sequences, respectively. Base evaluates tasks before application updates, and Agg. is the mean score across the four evolution axes.

Model	Type	Application evolution						Workflow evolution
		Base	S	G	N	I	Agg.	
General-purpose models								
Qwen3.8	27B	28.6	27.3	24.2	26.5	25.2	25.8	25.9
Claude Opus 5	API	30.9	29.6	29.1	30.4	24.6	28.5	29.7
GPT-5.6 Sol	API	34.2	32.3	30.9	26.8	29.7	29.9	30.4
Gemini 3.5 Flash	API	27.6	26.3	23.1	25.2	17.6	23.1	21.7
Specialized computer-use agents								
UI-TARS-1.5-7B	7B	16.7	14.3	8.6	10.5	11.2	11.1	11.5
OpenCUA-32B	32B	20.9	17.6	11.3	10.2	15.4	13.6	15.0
EvoCUA-32B	32B	21.6	18.6	10.6	13.5	16.3	14.7	16.6
GUI-Owl-1.5-32B-Instruct	32B	22.9	21.5	20.3	18.2	18.3	19.6	16.6
Self-improving RL models								
EvoCUA-32B + RL	32B	25.1	24.0	15.2	20.9	20.2	20.1	21.2
CUAEvolve SFT	32B	24.3	20.3	19.7	18.6	17.3	19.0	18.5
CUAEvolve SFT + RL	32B	**28.3**	**28.9**	**24.8**	**24.0**	**22.1**	**25.0**	**22.3**

Table 2: OSWorld-Verified task success (%).

Model	Type	SR (%)
General models		
Qwen3.8	27B	86.1
Claude Opus 5	API	83.4
GPT-5.6 Sol	API	83.2
Gemini 3.5 Flash	API	78.4
Specialized CUA models		
UI-TARS-1.5-7B	7B	27.3
OpenCUA-32B	32B	34.1
EvoCUA-32B	32B	45.9
GUI-Owl-1.5-32B-Instruct	32B	51.3
Self-improving RL models		
EvoCUA-32B + RL	32B	50.0
CUAEvolve SFT	32B	46.7
CUAEvolve SFT + RL	32B	**52.7**

Table 3: WindowsAgentArena task success (%).

Model	SR (%)
Specialized CUA models	
UI-TARS-1.5-7B	18.1
OpenCUA-32B	29.3
EvoCUA-32B	**47.3**
GUI-Owl-1.5-32B-Instruct	46.9
Self-improving RL models	
CUAEvolve SFT	44.5
CUAEvolve SFT + RL	46.7

and up from 44.5% for the SFT (+2.2 percentage points), demonstrating that self-improving RL retains general computer-use capabilities and transfers to Windows environments beyond its Linux online training distribution. The training reward curve is provided in Appendix B.2.

5.3 Ablation Analysis

RL curriculum. Table 4(a) compares RL curricula trained from the same SFT model and rollout budgets, In the random-curriculum baseline, we omit failure diagnoses and uniformly sample applicable mutation axes or workflow types in RL, and we have the following findings: 1) Self-improving RL curriculum achieves the highest task score on OSEvolveBench, which is 6.0 points above SFT and 3.5 points above the combined static RL curriculum. 2) Moreover, we evaluate the self-improving loop by replacing it with a random RL curriculum, and experimental results show that the self-improving curriculum witnesses a distinct gain (3.1 points in application evolution and 2.4 points in workflow evolution) over a random curriculum. It demonstrates the effectiveness of translating diagnoses into executable application and workflow training curriculum.

GUI-Coding interleaved trajectory. The GUI+Coding model outperforms the GUI-only model on OSWorld-Verified at both stages, reaching 46.7% versus 42.9% after SFT and 52.7% versus 47.0% after RL. As illustrated in Appendix B.3, the GUI-Coding CUA takes at least one coding action in 57–81% of tasks in each LibreOffice and VS Code domain and in 83% of multi-application

Table 4: Ablation results. (a) GUI-only and GUI–coding policies and RL curricula. (b) Evolving to seen and held-out applications and workflows.

(a) Policy and curriculum ablations

Action	Training variant	OSEvolveBench		OSWorld-Verified
		Application	Workflow	
Interleaved GUI-coding trajectory				
GUI	SFT	16.8	16.2	42.9
	SFT + RL	18.9	19.7	47.0
GUI+Coding	SFT	19.0	18.5	46.7
	SFT + RL	**25.0**	**22.3**	**52.7**
RL curriculum				
SFT		19.0	18.5	46.7
RL + static application tasks		21.1	19.2	49.5
RL + static workflow tasks		18.7	20.6	47.5
RL + static combined tasks		21.5	20.6	50.8
RL + random curriculum		21.9	19.9	50.6
RL + self-improving curriculum		**25.0**	**22.3**	**52.7**

(b) Seen and held-out applications

Track	Evaluation subset	SFT	RL Random curriculum	RL Self-improving curriculum
Application	Seen set	20.1	23.3	26.4
	Held-out set	11.9	12.9	16.0
	All	19.0	21.9	25.0
Workflow	Seen set only	22.6	23.2	24.9
	Held-out set	16.2	18.1	20.9
	All	18.5	19.9	22.3

tasks, where coding actions make up 53.6% of steps per task. When programmatic interfaces remain compatible across updates, coding actions offer alternative execution paths that consolidate multiple GUI operations into shorter action sequences and improve robustness to visual and interface changes. Appendix C presents inference trajectories illustrating GUI-only and interleaved GUI–coding execution.

Evolve to held-out applications. To assess whether curriculum gains extend beyond applications seen during training, we exclude Asana, ChatGPT, Claude, Twitter, and GitLab from the training pool and evaluate those unseen applications in OSEvolveBench. Table 4(b) organizes the comparison of SFT, the random curriculum, and the self-improving curriculum on seen versus held-out applications. It shows that self-improving RL improves held-out application-evolution scores from 11.9 to 16.0 and held-out workflow scores from 16.2 to 20.9. It also outperforms the random curriculum on these subsets by 3.1 and 2.8 points, respectively, demonstrating the self-improving capabilities towards held-out applications and workflows.

6 Conclusion

Application updates and changing workflows can erode the capabilities acquired by computer-use agents. CUAEvolve addresses this challenge with a unified GUI–coding policy and a self-improving RL loop. The policy interleaves GUI actions with executable code, allowing code actions to reduce reliance on visual grounding and shorten execution trajectories when compatible programmatic interfaces are available. The self-improving RL loop uses failure diagnoses to synthesize application- and workflow-evolution tasks and environments at scale, allowing the policy and its training curriculum to evolve together. To evaluate CUA performance under task evolution, we introduce OSEvolveBench, a benchmark that pairs original and evolved task conditions across two families: application evolution and workflow evolution. Experiments show that our approach improves on both base and evolved application and workflow, outperforming prior open-source CUAs at comparable scales. It also maintains strong retention performance on OSWorld-Verified and WindowsAgentArena.

AI USE STATEMENT

As disclosed in Appendix D, large language models serves as a model for auxiliary trajectory synthesis and also acts as the diagnoser, curriculum generator and admission judge, which makes it part of trajectory generation, task generation and evaluation-related analysis. Several AI models are also evaluated as baselines, and the authors remain responsible for the final manuscript and its claims. We used large language models to polish the language of manuscript.

ETHICS STATEMENT

This work develops computer-use agents intended to improve efficiency by assisting with routine and repetitive computer tasks and reducing the manual effort they require. All experiments were conducted in benchmark and sandboxed environments, and the agents were not deployed with public users. Responsible deployment of such agents would require user authorization for their actions and oversight appropriate to each task.

REPRODUCIBILITY STATEMENT

We will release our code, benchmark construction and evaluation resources, and experimental configurations. The main text and appendices describe the data construction and filtering procedures, training objectives and configurations, and evaluation settings. Together, the configurations reported in the paper and the released materials are intended to support reproducing our methods and comparisons.

REFERENCES

Anthropic. Introducing Claude Opus 5. Official model announcement, July 2026. URL https://www.anthropic.com/news/claude-opus-5.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025. URL https://arxiv.org/abs/2511.21631.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. Windows agent arena: Evaluating multi-modal OS agents at scale. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4874–4910. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/bonatti25a.html.

ByteDance Seed Team. ByteDance Seed agent model UI-TARS-1.5 open source: Achieving SOTA performance in various benchmarks. Official model announcement, April 2025. URL https://seed.bytedance.com/en/blog/bytedance-seed-agent-model-ui-tars-1-5-open-source-achieving-sota-performance-in-various-benchmarks.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Siqi Fan, Minghao Li, Xiaoqian Ma, Wenhui Tan, Xiusheng Huang, Juntong Wu, Liujie Zhang, Shuo Shang, and Weihang Chen. Screenshots or tools? eliciting tool use and managing multimodal context in hybrid GUI–MCP computer-use agents. *arXiv preprint arXiv:2608.03327*, 2026. doi: 10.48550/arXiv.2608.03327. URL https://arxiv.org/html/2608.03327v2.

Google DeepMind. Gemini 3.5 Flash Model Card, May 2026. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Xuhao Hu, Xi Zhang, Haiyang Xu, Kyle Qiao, Jingyi Yang, Xuanjing Huang, Jing Shao, Ming Yan, and Jieping Ye. ToolCUA: Towards optimal GUI-Tool path orchestration for computer use agents. *arXiv preprint arXiv:2605.12481*, 2026.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jiang21b.html.

Wanli Li, Bowen Zhou, Yunyao Yu, Zhou Xu, Yifan Yang, Dongsheng Li, and Caihua Shan. WeaveBench: A long-horizon, real-world benchmark for computer-use agents with hybrid interfaces. *arXiv preprint arXiv:2606.09426*, 2026. doi: 10.48550/arXiv.2606.09426. URL https://arxiv.org/abs/2606.09426.

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.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

OpenAI. GPT-5.6 Sol model. OpenAI API model documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. Accessed September 23, 2026.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17473–17498. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. doi: 10.48550/arXiv.2501.12326. URL https://arxiv.org/abs/2501.12326.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork. Official model announcement, 2026. URL https://qwen.ai/blog?id=qwen3.8. Accessed September 23, 2026.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jingwei Sun, Jianing Zhu, Yuanyi Li, Tongliang Liu, Xia Hu, and Bo Han. AgentHijack: Benchmarking Computer Use Agent Robustness to Common Environment Corruptions. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://arxiv.org/abs/2605.25707.

Xueqiao Sun, Xiaohan Wang, Ludwig Schmidt, Serena Yeung-Levy, and Yuhui Zhang. Learning from failure: Inference-time self-improvement for computer-use agents. *arXiv preprint arXiv:2606.31270*, 2026b. doi: 10.48550/arXiv.2606.31270. URL https://arxiv.org/abs/2606.31270.

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. doi: 10.48550/arXiv.2508.04700. URL https://arxiv.org/abs/2508.04700.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. CUA-Gym: Scaling verifiable training environments and tasks for computer-use agents. arXiv preprint arXiv:2605.25624, 2026a. URL https://arxiv.org/abs/2605.25624.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y.Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025. doi: 10.48550/arXiv.2508.09123. URL https://arxiv.org/abs/2508.09123.

Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. RLAnything: Forge environment, policy, and reward model in completely dynamic RL system. *arXiv preprint arXiv:2602.02488*, 2026b. doi: 10.48550/arXiv.2602.02488. URL https://arxiv.org/abs/2602.02488.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-Copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024. doi: 10.48550/arXiv.2402.07456. URL https://arxiv.org/abs/2402.07456.

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, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. Mobile-Agent-v3.5: Multi-platform fundamental GUI agents. *arXiv preprint arXiv:2602.16855*, 2026. URL https://arxiv.org/abs/2602.16855.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, Jinrui Ding, Xiandi Ma, Yuchen Xie, Peng Pei, Xunliang Cai, and Xipeng Qiu. EvoCUA: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026. doi: 10.48550/arXiv.2601.15876. URL https://arxiv.org/abs/2601.15876.

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

A OSEvolveBench Construction

A.1 Release sources and counting rules.

Cumulative feature-release records, July 2023–August 2026, where entries are feature announcements or release-version records for Supabase, Notion, Thunderbird, Firefox, Docker Desktop, and Stripe.

A.2 Application-evolution axes and implementation

This section describes the construction and validation of application-evolution instances in OSEvolveBench. Table 5 defines four axes: surface appearance, geometry, naming, and interaction. Each variant modifies a single axis, preserving the task objective, target content, evaluator, and non-target application properties. Every variant is paired with its unchanged base task, enabling direct comparison under the same success criteria.

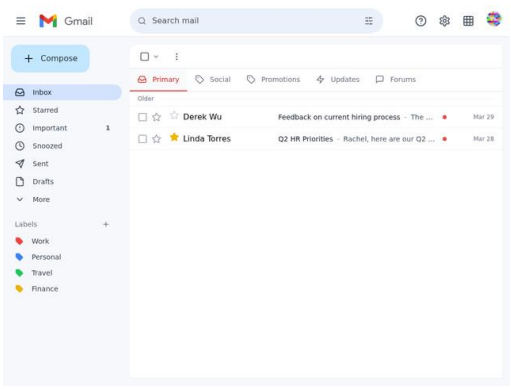
Table 5: Application-evolution axes. Each variant modifies one axis and preserves the task objective, target content, and evaluator.

Axis	Changed property	Intended separation
S	Surface appearance	Visual styling; keep target geometry and names.
G	Geometry	Control positions or layout; preserve names and behavior.
N	Naming	Interface labels; preserve geometry and target content.
I	Interaction sequence	One added confirmation step before a submit action; preserve the initial screen's labels and geometry, and the terminal outcome.

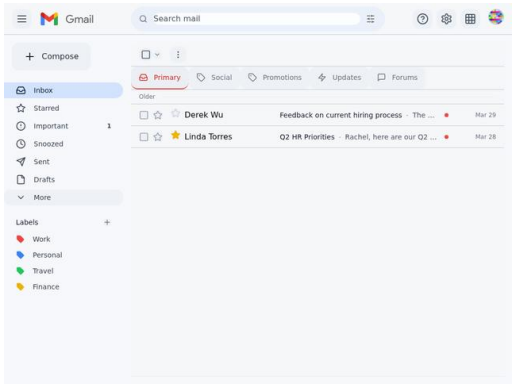
Mock web applications. We instantiate application evolution on 31 React-based mock applications that reproduce common productivity and communication interfaces. Task completion rate is assessed from the resulting application state, where the evaluator reads a dedicated JSON state endpoint independently of the document object model (DOM). This separation allows visual styling, layout, control labels, and interaction sequences to vary without modifying the reward program or success criteria. We apply mutations to built application artifacts and verify their effects in the deployed interface.

A.3 Worked example: application evolution

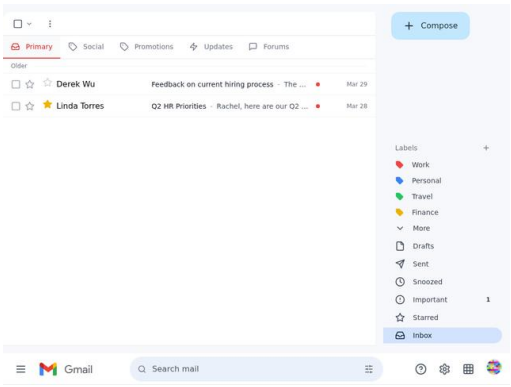
Figure 5 shows one mock mail application under surface appearance, geometry, naming, and interaction axes;



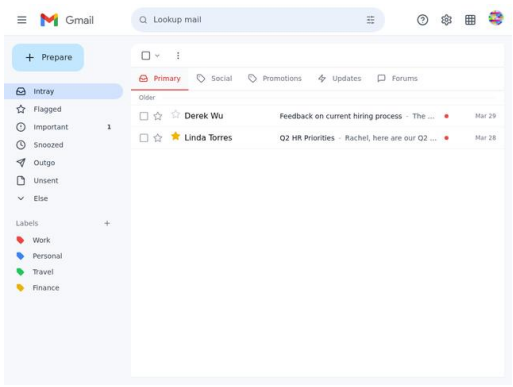
base



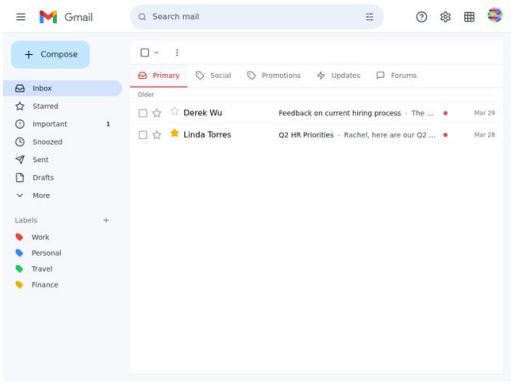
S: surface appearance



G: geometry



N: naming



I: interaction sequence, first frame

Figure 5: An example of application evolution (Gmail) with surface appearance, geometry, naming, and interaction axis. *base*: the unmutated built artifact. *S*: corner radius, stroke, and palette differ; *G*: the side rail moves to the opposite edge, the top bar to the bottom, and the navigation order is reversed; nothing is renamed. *N*: chrome labels are substituted (Search becomes Lookup, Compose becomes Prepare, Inbox becomes Intray, Starred becomes Flagged); positions and page data are untouched. *W*: the first frame is by construction indistinguishable from the base, because the arm differs only at the submit click.

A.4 WORKFLOW-EVOLUTION CONSTRUCTION

Workflow evolution encompasses cross-application workflows with evolving user preferences and working habits, as well as previously unseen workflows. We construct this family through revisions to existing task contracts and the introduction of workflows absent from the source task pool.

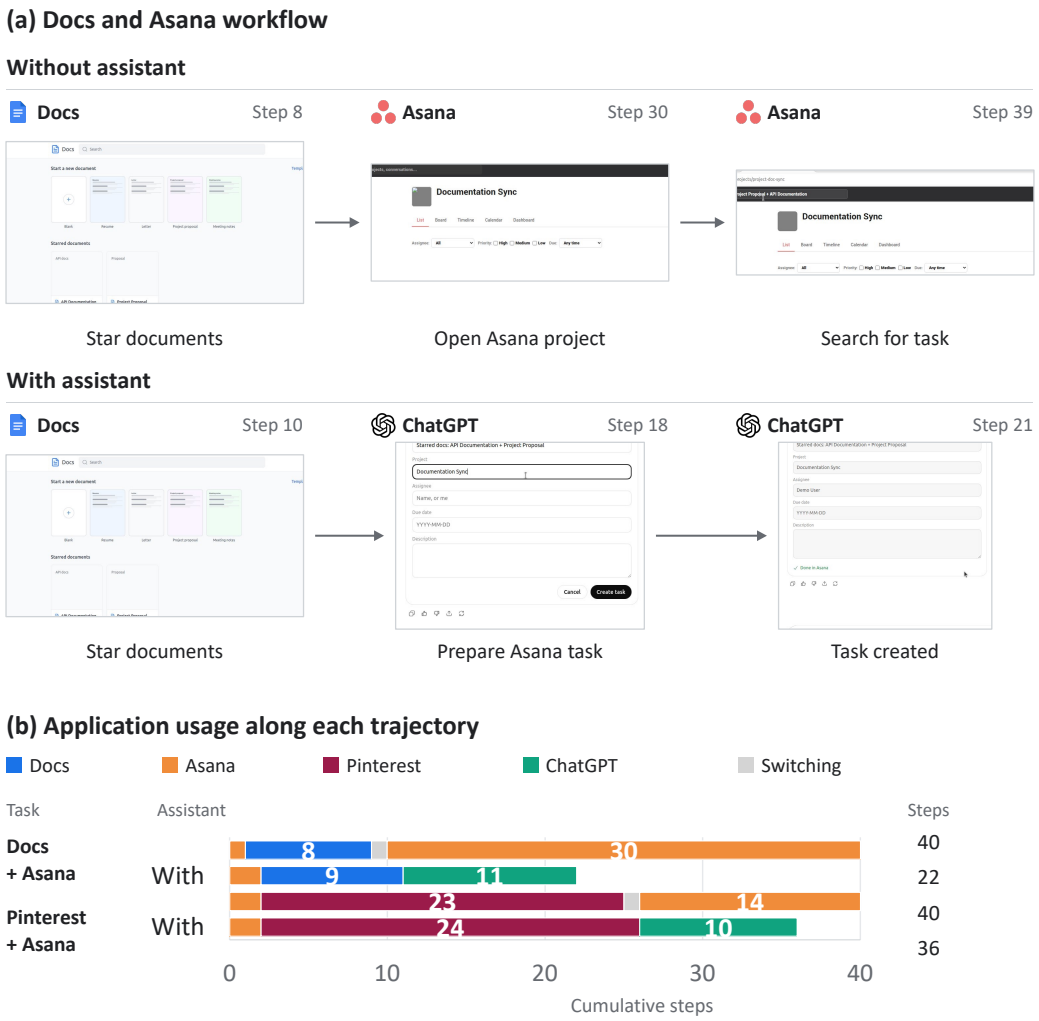


Figure 6: An example of workflow evolution. (a) Three chronological keyframes for a Docs and Asana workflow. Previously, the model started documents in Docs (step 8), opened the target Asana project (step 30), and continued searching for the task (step 39). Now with a new workflow (GPT assistant), it starts documents (step 10), prepares an Asana task through the connector in ChatGPT (step 18), and receives the Done in Asana confirmation (step 21). (b) Comparing application usage and trajectory length with or without new workflows.

Revised cross-application workflows. We start from a task in CUA-Gym (Wang et al., 2026a) and introduce a versioned contract that specifies changes to the user's requirements or preferred execution procedure. A *novel-instruction* contract introduces an authoritative input, such as a policy file containing a structured table that determines item selection and ordering. The required deliverables are revised according to these rules. A *cross-application* contract constructs a workflow involving two to five applications, including the source application in a combination absent from the original task pool.

Novel and unseen workflows. We curate new workflows and the application functions required to execute them. Figure 1(b) illustrates a transition from searching for information in Chrome, manually drafting an email in Outlook, and sending it, to delegating information gathering and drafting to ChatGPT and confirming delivery through an email connector. The underlying objective remains familiar, but the selected applications and workflows of the user change.

A.5 HUMAN VERIFICATION AND TASK ADMISSION

Table 6 summarizes the admission criteria. Application variants must preserve the original task objective and evaluator and exhibit the intended interface change. Workflow candidates are assessed against their revised instructions, deliverables, and required information dependencies. During online RL, admitted candidates augment the retained task pool, and the sampling distribution is updated for subsequent training steps.

Table 6: Admission criteria for application and workflow evolution tasks.

Criterion	**Validation requirement**
Evaluator validity	Application variants retain the base task's reward program and success criteria. Workflow evaluators assess the revised instructions and deliverables.
Task solvability	A reference solution completes the task in the deployed environment using the available inputs and receives full credit from the evaluator.
Evolution validity	The intended change is verified in rendered states or executed interactions, with non-target properties unchanged.
Application Integration	Deliverables can be run and saved on a real machine.

A.6 BENCHMARK COMPOSITION

Table 7 gives the counts of OSEvolveBench per application and workflows. The discriminator validation maintains disjoint training and OSEvolveBench's task instances and audit their underlying assets for overlap, including input files, document contents, and task-specific application states. Newly generated training tasks undergo the overlap check before admission to the training pool. Applications and task categories may be shared across splits, while evaluation tasks use separate instances and assets.

A.7 BENCHMARK STATISTICS

Table 8 summarizes the evolved instances in OSEvolveBench. The application-evolution track contains 99 admitted source tasks and 386 evolved variants, comprising 98 surface, 99 geometry, 98 naming, and 91 interaction variants. The workflow-evolution track contains 98 variants.

Table 7: Counts of OSEvolveBench per application and workflows. The discriminator validation maintains disjoint training and OSEvolveBench's task instances and audit assets for overlap, including input files, document contents. We also hold out five applications exclusively for evaluation: Asana, ChatGPT, Claude, Twitter, and GitLab.

	Application					Workflow
App	Base	S	G	N	I	
Asana	2	2	2	2	2	52
ChatGPT	0	0	0	0	0	11
Claude	0	0	0	0	0	11
Discord	2	2	2	2	2	27
DocuSign	2	2	2	2	2	3
Facebook	2	2	2	2	2	8
GitHub	6	6	6	6	6	3
GitLab	6	6	6	6	6	4
Gmail	2	2	2	2	2	4
Google Calendar	2	2	2	2	2	7
Google Docs	6	6	6	6	6	7
Google Drive	6	6	6	6	6	7
Google Sheets	1	1	1	1	0	7
HubSpot	6	6	6	6	6	5
Instacart	6	6	6	6	6	4
Instagram	1	1	1	1	0	3
Jira	1	1	1	1	1	4
LinkedIn	5	5	5	5	5	3
Microsoft Teams	6	6	6	6	6	3
Monday.com	0	0	0	0	0	3
Notion	1	1	1	1	1	2
Outlook Web	1	1	1	1	0	3
Pinterest	1	1	1	1	1	5
Postman	6	6	6	6	6	5
Reddit	1	1	1	1	0	4
Salesforce	5	5	5	5	5	2
Shopify Admin	6	6	6	6	6	3
Slack	1	1	1	1	1	2
Stripe Dashboard	6	6	6	6	6	2
Trello	1	1	1	1	0	1
Twitter	5	5	5	5	5	1
Uber Eats	1	1	1	0	0	1
WeChat	1	0	1	1	0	1
Xiaohongshu	1	1	1	1	0	0
Total	99	98	99	98	91	98

Table 8: Construction statistics of OSEvolveBench.

Family	Track	Evolution type	Total
Application evolution	Mock applications	Surface appearance, geometry, naming, and interaction	386
Workflow evolution	Workflows	Revised cross-application workflows, novel application combinations and unseen workflows	98

B Policy and Training Details

B.1 Action interface

Table 9 lists every action in the policy's enumeration. Pointer and keyboard actions execute as ordinary input events; control actions wait or end the episode. The coding action is a first-class member of the same enumeration whose execution result returns as a text observation.

Table 9: The action space of the CUA model. Coordinates are integers 0–999 on the screen normalized to a 1000×1000 grid. The coding action is the policy-written one Section 4.1 describes.

Action	Arguments	Effect
Pointer actions (coordinates are integers 0–999 on the normalized screen grid)		
`left_click`	`coordinate`	Click the primary button at the coordinate.
`right_click`	`coordinate`	Click the secondary button at the coordinate.
`middle_click`	`coordinate`	Click the middle button at the coordinate.
`double_click`	`coordinate`	Double-click at the coordinate.
`triple_click`	`coordinate`	Triple-click at the coordinate.
`mouse_move`	`coordinate`	Move the pointer without clicking.
`left_click_drag`	`coordinate`	Drag with the primary button from the current pointer position to the coordinate.
`scroll`	`coordinate`, `pixels`	Scroll at the coordinate by a signed amount.
Keyboard actions		
`type`	`text`	Type the text at the current focus.
`key`	`keys`	Press one key or a chord (a hotkey): press in order, release in reverse order.
`key_down`	`keys`	Press and hold the keys in order without releasing them.
`key_up`	`keys`	Release held keys in reverse order.
Control actions		
`wait`	`time`	Wait the given number of seconds for the interface to settle before the next observation.
`terminate`	`status`	End the episode, declaring success or failure.
Coding action		
`coding`	`language`, `code`	Execute the policy-written Python or Bash block in the environment; the exit status, standard output, and standard error return as the next observation, and the block is persisted with the step.

B.2 RL objective and implementation

Figure 7 shows the reward logged during RL training. For each task, the behavior policy $\pi_{\theta_{\text{old}}}$ generates trajectories from the initial state e_0. Trajectory τ_i contains U_i turns. At turn u, context $c_{i,u}$ comprises the instruction, recent screenshots, and previous actions with execution feedback; response $y_{i,u}$ contains a graphical or coding action. Setup or evaluator failures that prevent reliable scoring are excluded before the group baseline of Eq. (1) is computed; scorable task failures remain in the group with reward zero, and empty groups are omitted. For generated token $y_{i,u,t}$, the importance ratio is

$$\rho_{i,u,t}(\theta) = \frac{\pi_\theta(y_{i,u,t} \mid c_{i,u}, y_{i,u,<t})}{\pi_{\theta_{\text{old}}}(y_{i,u,t} \mid c_{i,u}, y_{i,u,<t})}, \tag{2}$$

where θ_{old} remains fixed during optimization of the collected batch. Let $\mathcal{B}$ denote the current batch of B tasks, and let $m_{i,u,t} \in \{0, 1\}$ indicate generated tokens contributing to the loss. We maximize

a PPO clipped surrogate normalized by the number of loss-bearing tokens in the batch (Schulman et al., 2017):

$$\mathcal{J}(\theta) = \frac{1}{C} \sum_{q \in \mathcal{B}} \sum_{i \in \mathcal{I}_q} \sum_{u=1}^{U_i} \sum_{t=1}^{|y_{i,u}|} m_{i,u,t}\, \ell_{i,u,t}(\theta),$$

$$\ell_{i,u,t}(\theta) = \min\Big(\rho_{i,u,t}(\theta)\hat{A}_i,\ \mathrm{clip}\big(\rho_{i,u,t}(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_i\Big), \tag{3}$$

$$C = \sum_{q \in \mathcal{B}} \sum_{i \in \mathcal{I}_q} \sum_{u=1}^{U_i} \sum_{t=1}^{|y_{i,u}|} m_{i,u,t}.$$

Here, ϵ is the clipping parameter, and the objective applies when $C > 0$. The denominator counts all loss-bearing tokens, including those from zero-advantage trajectories. Instructions, screenshots, and execution outputs provide conditioning information and are excluded from the loss. We use neither group-standard-deviation normalization nor a reference-policy KL penalty.

Identical-reward groups. For a valid group of size $m_q = |\mathcal{I}_q| \geq 1$ with independent Bernoulli rewards of success probability p_q, all rewards agree with probability $p_q^{m_q} + (1-p_q)^{m_q}$, which approaches one as p_q approaches zero or one. Such groups have zero advantages and, without a KL term, contribute no direct policy-gradient signal while their tokens still enter C. This motivates the success-rate-guided task synthesis in Section 4.3: consistently solved tasks prompt harder variants, and persistently unsuccessful tasks prompt simpler execution conditions, aiming to generate tasks with informative variation in rollout outcomes.



Figure 7: Training reward. The task distribution changes during training; reward values reflect both policy performance and curriculum composition.

B.3 GUI–CLI RESULT DISTRIBUTION

Table 10 summarizes trajectories of the saved RL policy on all 369 OSWorld tasks at most 50 steps. Coding steps (%) is the mean over a domain's tasks with at least one action (358 of 369 overall). Avg steps is computed over all domain tasks.

Table 10: Distribution of CUA's GUI–CLI trajectories across 10 OSWorld domains.

Domain	Score	Coding steps (%)	Tasks using coding (%)	Avg. steps
Cross-application	39.6	53.6	83	16.5
Impress	44.7	26.5	57	11.1
Calc	55.3	31.5	81	10.9
Chrome	39.1	9.6	50	26.6
GIMP	80.8	14.5	54	20.2
OS	83.3	32.0	75	12.7
VS Code	69.6	32.5	78	11.9
Writer	52.2	29.8	74	13.7
VLC	70.6	25.3	59	10.5
Thunderbird	60.0	22.9	53	16.1

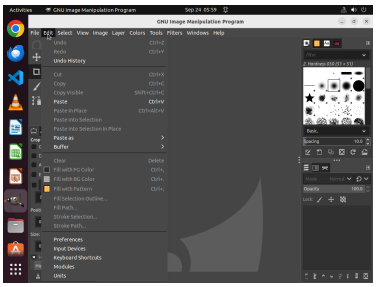
C VISUALIZATION

Three complete six-step OSWorld-Verified trajectories from the same 32B RL checkpoint are shown below. All have evaluator score 1. The recorded model text and actions are reproduced with formatting changes only; screenshots show the state after each action. Termination counts as a step. Code, standard output, and exit codes are shown as logged.

C.1 GUI-ONLY: GIMP PREFERENCES

Task. Set the minimum number of undo steps to 100.

Outcome. Evaluator score: 1; six policy steps.



After step 1

Step 1: left_click

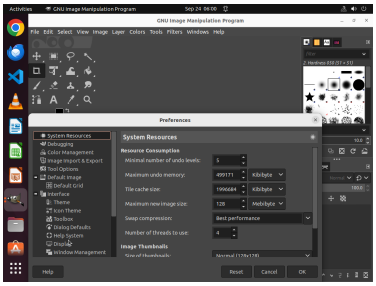
Think. The task is to set the minimum number of undo steps to 100 in GIMP. This is typically found in the Preferences dialog under the Environment section. I need to open the Preferences dialog first.

Let me click on the "Edit" menu to access Preferences.

Action. Left-click at (118,99).

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "left_click", "coordinate":
    [118, 99]}}
```



After step 2

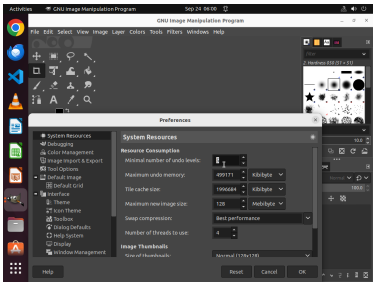
Step 2: left_click

Think. The Edit menu is open. I can see "Preferences" option. I'll click on it to open the preferences dialog.

Action. Left-click at (173,843).

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "left_click", "coordinate":
    [173, 843]}}
```



After step 3

Step 3: triple_click

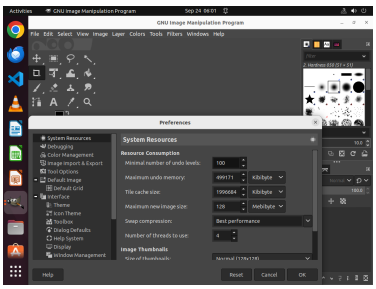
Think. The Preferences dialog is open on the System Resources page. I can see the "Minimal number of undo levels:" field currently shows 5. I need to change this to 100. I'll triple-click on the field to select all the text, then type 100.

Action. Triple-click.

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "triple_click", "coordinate":
    [587, 563]}}
```

GUI-only: GIMP preferences (continued): steps 4–6



After step 4

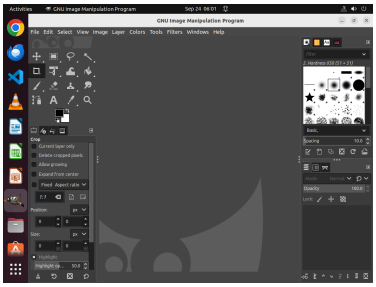
Step 4: type

Think. The field is now selected. I'll type 100 to set the value.

Action. Type: '100'

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "type", "text": "100"}}
```



After step 5

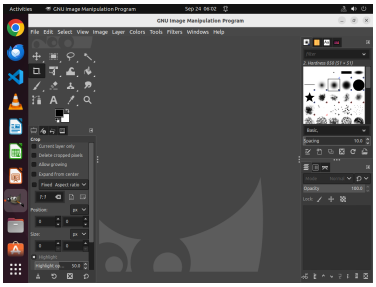
Step 5: left_click

Think. The value is now 100. I need to click OK to save the setting.

Action. Left-click at (803,963).

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "left_click", "coordinate":
    [803, 963]}}
```



After step 6

Step 6: terminate

Think. The preferences dialog has closed and we're back to the main GIMP interface, confirming the settings were saved. The minimal number of undo levels has been successfully set to 100.

Action. Terminate the task (completed).

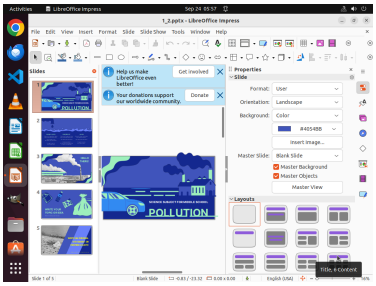
Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "terminate", "status":
    "success"}}
```

C.2 CODING + GUI: PRESENTATION EDITING

Task. Navigate to slide 5 and set the font color of all textboxes to yellow. Use exactly 'yellow'—no variations such as light yellow, dark yellow, or any other color.

Outcome. Evaluator score: 1; six policy steps.



After step 1

Step 1: coding

Think. Let me analyze the task. I need to:
1. Navigate to slide 5
2. Set the font color of all textboxes to exactly 'yellow'

I can see LibreOffice Impress is open with a presentation. I can see slide 5 in the panel on the left. Let me use the coding tool to do this programmatically since it involves precise color setting.

First, let me find the file path. The title bar shows "1_2.pptx - LibreOffice Impress". Let me find the file.

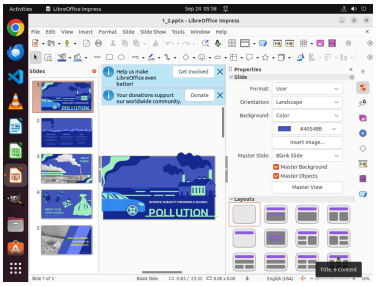
Action. Run bash to Find the presentation file path

Code (bash).

```
find / -name "1_2.pptx" 2>/dev/null
```

Exit code: 1
stdout

```
/home/user/Desktop/1_2.pptx
```



After step 2

Step 2: coding

Think. The file is at '/home/user/Desktop/1_2.pptx'. Now I need to use python-pptx to set all textboxes on slide 5 to yellow font color. Let me write the code to do this.

Action. Run python to Set all textbox font colors to yellow on slide 5

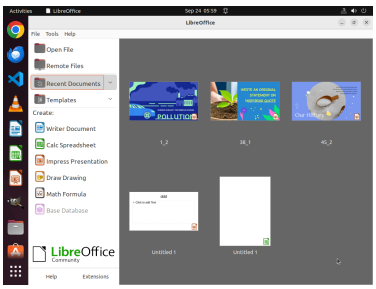
Code (python).

```
from pptx import Presentation
from pptx.util import Pt
from pptx.dml.color import RGBColor

path = "/home/user/Desktop/1_2.pptx"
prs = Presentation(path)

slide = prs.slides[4]  # slide 5 (0-indexed)

for shape in slide.shapes:
    if shape.has_text_frame:
        for para in
    shape.text_frame.paragraphs:
            for run in para.runs:
                run.font.color.rgb =
    RGBColor(0xFF, 0xFF, 0x00)

prs.save(path)
print("Done: all textbox runs on slide 5
    set to yellow.")
```

Exit code: 0
stdout

```
Done: all textbox runs on slide 5 set to
    yellow.
```

Coding + GUI: presentation editing (continued): steps 3–4



After step 3

Step 3: key

Think. The file has been saved programmatically. The LibreOffice Impress still shows the old version. I need to reload the file in LibreOffice to verify the changes, or I can just confirm the file was saved correctly. Since the file was saved directly via python-pptx, the changes are persisted on disk. The task says the artifact needs to be persisted, which it is - the file was saved.

However, the LibreOffice app still has the old version open. If the user saves from LibreOffice, it would overwrite my changes. I should reload the file in LibreOffice to verify, or just confirm the file is saved.

Actually, the task says "save files explicitly (ctrl+s, and handle any 'Keep format' dialog)". Since I saved the file directly via python-pptx, the file on disk is already updated. The LibreOffice display is just showing the cached version.

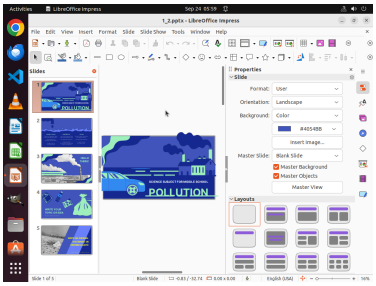
Let me reload the file in LibreOffice to verify the changes are correct, and also to make sure LibreOffice doesn't overwrite the file when it's closed.

I'll close the notification banners first, then reload the file.

Action. Press ctrl+w.

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "key", "keys": ["ctrl", "w"]}}
```



After step 4

Step 4: double_click

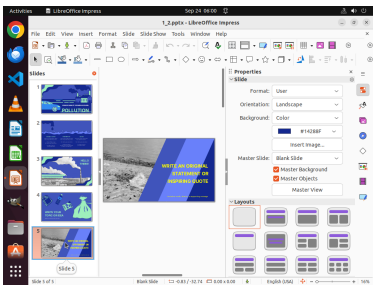
Think. The file was closed and now I'm at the LibreOffice Start Center. I can see the 1_2 presentation in the recent documents. Let me click on it to reopen it and verify the changes.

Action. Double-click.

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "double_click", "coordinate":
    [436, 378]}}
```

Coding + GUI: presentation editing (continued): steps 5–6



After step 5

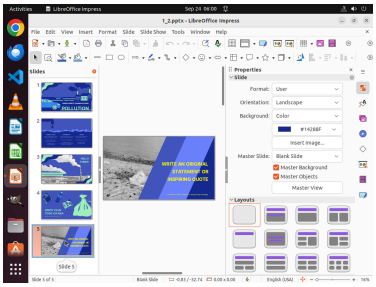
Step 5: left_click

Think. The file is now open. I can see slide 5 in the panel on the left, and it appears to have yellow text. Let me click on slide 5 to verify the changes.

Action. Left-click at (164,842).

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "left_click", "coordinate":
    [164, 842]}}
```



After step 6

Step 6: terminate

Think. Slide 5 is now showing with the text "WRITE AN ORIGINAL STATEMENT OR INSPIRING QUOTE" in yellow color. The task is complete - the font color of all textboxes on slide 5 has been set to yellow (RGB 0xFF, 0xFF, 0x00).

Action. Terminate the task (completed).

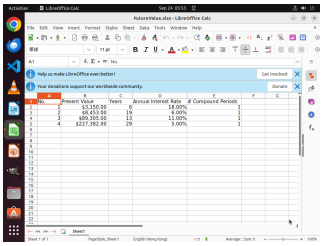
Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "terminate", "status":
    "success"}}
```

C.3 CODING + GUI: SPREADSHEET CONSTRUCTION

Task. Create a new sheet named "Sheet2" and merge cells A1:C1 to write the header "Investment Summary". Beneath that, merge cells A2:B2 to write "High Interest Rate" and merge cells C2:D2 to form "Low Interest Rate".

Outcome. Evaluator score: 1; six policy steps.



After step 1

Step 1: coding

Think. Let me analyze the task:
1. Create a new sheet named "Sheet2"
2. Merge cells A1:C1 and write "Investment Summary"
3. Merge cells A2:B2 and write "High Interest Rate"
4. Merge cells C2:D2 and write "Low Interest Rate"

I can see LibreOffice Calc is open with FutureValue.xlsx. Let me use the coding tool to do this programmatically via Python with openpyxl, which would be more reliable than clicking through the GUI.

First, let me find the file path.

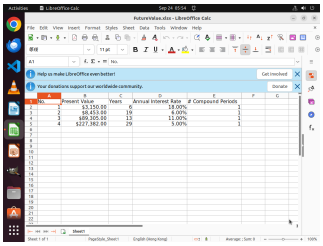
Action. Run bash to Find the FutureValue.xlsx file path

Code (bash).

```
find / -name "FutureValue.xlsx" 2>/dev/null
```

Exit code: 1
stdout

```
/home/user/FutureValue.xlsx
```



After step 2

Step 2: coding

Think. I found the file at '/home/user/FutureValue.xlsx'. Now I'll use openpyxl to create Sheet2 and set up the merged cells with headers.

Action. Run python to Create Sheet2 with merged cells and headers

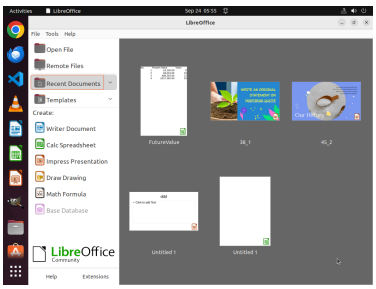
Code (python).

```
import openpyxl
from openpyxl import load_workbook
path = "/home/user/FutureValue.xlsx"
wb = load_workbook(path)
# Create Sheet2
wb.create_sheet("Sheet2")
ws = wb["Sheet2"]
# Merge A1:C1 and write header
ws.merge_cells("A1:C1")
ws["A1"] = "Investment Summary"
# Merge A2:B2
ws.merge_cells("A2:B2")
ws["A2"] = "High Interest Rate"
# Merge C2:D2
ws.merge_cells("C2:D2")
ws["C2"] = "Low Interest Rate"
wb.save(path)
print("Done! Sheet2 created with merged cells.")
print("Sheets:", wb.sheetnames)
```

Exit code: 0
stdout

```
Done! Sheet2 created with merged cells.
Sheets: ['Sheet1', 'Sheet2']
```

Coding + GUI: spreadsheet construction (continued): steps 3–4



After step 3

Step 3: key

Think. The file was saved successfully by the coding tool, but LibreOffice Calc still has the old version open. I need to reload the file in LibreOffice Calc to see the changes. Let me close and reopen the file, or use the "Reload" option.

Actually, the task says the file needs to be persisted. The coding tool already saved the file to disk. But LibreOffice has the old version open and might overwrite it if the user saves from LibreOffice.

I should reload the file in LibreOffice to verify the changes are there, and also to make sure LibreOffice doesn't overwrite the saved file. Let me close the notification bars first and then reload the file.

Actually, the best approach is to just verify the file was saved correctly using the coding tool, and then reload it in LibreOffice. Let me reload the file in LibreOffice by closing and reopening it.

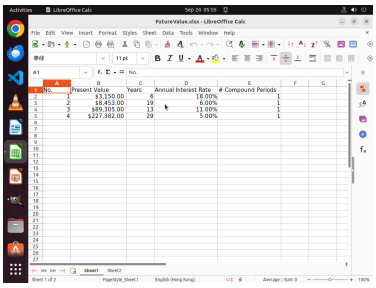
Wait - the task says "save files explicitly (ctrl+s, and handle any 'Keep format' dialog) before finishing." The file was already saved by the coding tool. But LibreOffice still has the old version open. If I close LibreOffice without saving, it will keep the coding tool's version. If I save from LibreOffice, it might overwrite the coding tool's changes.

The safest approach is to reload the file in LibreOffice so it picks up the changes made by the coding tool. Let me do that.

Action. Press ctrl+w.

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "key", "keys": ["ctrl", "w"]}}
```



After step 4

Step 4: double_click

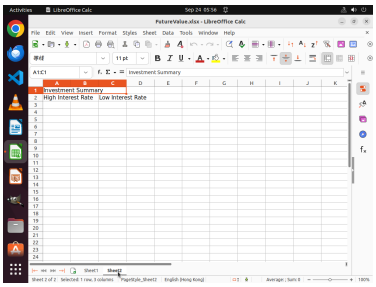
Think. The file was closed and now I'm at the LibreOffice Start Center. I can see FutureValue in the recent documents. Let me click on it to reopen it.

Action. Double-click.

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "double_click", "coordinate":
    [434, 352]}}
```

Coding + GUI: spreadsheet construction (continued): steps 5–6



After step 5

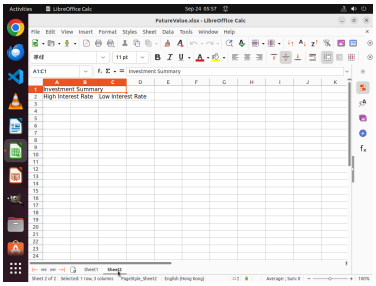
Step 5: left_click

Think. The file is now open and I can see both Sheet1 and Sheet2 tabs at the bottom. Let me click on Sheet2 to verify the merged cells are there.

Action. Left-click at (307,954).

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "left_click", "coordinate":
    [307, 954]}}
```



After step 6

Step 6: terminate

Think. Sheet2 is showing correctly with:
- A1:C1 merged with "Investment Summary"
- A2:B2 merged with "High Interest Rate"
- C2:D2 merged with "Low Interest Rate"

The file is already saved (it was saved by the Python script). The task is complete.

Action. Terminate the task (completed).

Tool call.

```
{"name": "computer_use", "arguments":
    {"action": "terminate", "status":
    "success"}}
```

D Language Models and Prompts in the Self-Improving Loop

CUAEvolve leverages GPT-5.6 Sol (OpenAI, 2026) in SFT and self-improving RL loop: trajectory synthesis for the SFT warm-up (Section 4.1), the diagnoser Φ (Section 4.2), the curriculum generator Ψ, and the discriminator of admission gate Admit (Section 4.3).

D.1 GUI-Coding Trajectory synthesis

GPT-5.6 Sol is the multimodal model used to construct the interleaved SFT trajectories of Section 4.1. In *online synthesis*, the model generates task-conditioned code from the current environment, and this code is executed in a sandbox. The recorded GUI actions, coding actions, and observed outcomes form the trajectory. In *offline transformation*, the model rewrites selected segments of existing GUI-only demonstrations as symbolic programmatic operations, and these rewritten segments are not executed at this stage. When several executed candidates exist for the same task, we keep the trajectory with the highest score and break ties by choosing the one with fewer steps. The policy imitates the demonstrations from both routes, which provides teacher supervision during SFT.

D.2 Diagnoser

At each generation step $s \in \mathcal{U}$ (Algorithm 1), the diagnoser Φ uses GPT-5.6 Sol to analyze the step's training rollouts $\mathcal{F}_s$ together with the success estimates $\{\hat{p}_q\}$. The analysis covers training trajectories, execution feedback, and terminal outcomes. Failure statistics are aggregated by task family and application domain, and application-evolution failures are organized by surface appearance, geometry, naming, and interaction.

```
You are the failure analyst for a computer-use agent under reinforcement learning. You are
     given one training step's rollout record: per-task success rates, the task family and
     application domain of each task, and for each failed rollout its action sequence,
     execution feedback and the evaluator's per-component verdict. Return ONLY a JSON object
     with the key diagnoses, a list. Each entry has: mode, a short name for one failure mode
     actually observed; evidence, the task ids and the specific steps that show it; families,
     the task families it occurs on; axis, one of surface, geometry, naming, interaction or
     workflow, naming the change that would exercise this weakness; and share, the fraction
     of failed rollouts it accounts for. Ground every entry in the record you were given: do
     not report a failure mode you cannot point to, do not propose fixes, and do not rank
     tasks you saw no rollouts for. Distinguish a policy failure from an environment or
     evaluator failure, and put the latter under the separate key infrastructure so it is
     excluded from the curriculum.
```

Listing 1: System prompt for the diagnoser.

D.3 CURRICULUM GENERATOR

The curriculum generator Ψ uses GPT-5.6 Sol to synthesize candidates $\mathcal{C}_s$. Application-evolution candidates modify one axis in $\mathcal{M}$ while preserving the source instruction, target content, and terminal evaluator. Workflow-evolution candidates revise instructions and deliverables, or they introduce new workflow combinations with required information dependencies. Listing 2 includes system prompt from our task-construction pipeline for workflow construction.

```
You are the code author for a verifiable computer-use task. Return ONLY a JSON object with
     the keys task_json, initial_setup_py, reward_py, reference_solution_py and changes_md,
     each a complete file as a single string. No markdown fences, no commentary outside the
     JSON. The code must be correct Python 3.11 that runs with only openpyxl, python-docx,
     python-pptx, pymupdf and the standard library. Never execute processes, network calls,
     signals, eval/exec or os.system. Scope every file operation under the given root or your
     own read-only assets.
```

Listing 2: System prompt for workflow construction.

D.4 ADMISSION GATE

The discriminator behind Admit uses GPT-5.6 Sol to decide whether an application-evolution or workflow-evolution candidate qualifies for the training pool. It checks each candidate against the four criteria: evaluator validity, task solvability, evolution validity, and application integration. Listing 3 gives the system prompt template for these checks. The judge receives the parent and candidate specifications together with reference-run logs and rendered artifacts. It returns a verdict with supporting evidence for each criterion and admits a candidate only when all four criteria pass.

```
You are the admission gate for evolved computer-use training tasks. You judge one candidate
     from family A (application evolution) or family C (workflow evolution) and decide
     whether it qualifies for the training pool. You only check the candidate. You do not
     solve, repair, or rewrite it.

Inputs: (a) the candidate family, A or C; (b) the parent task's specification, assets, and
     evaluator; (c) the candidate task's specification, implementation, and evaluator; (d)
     the intended change and the explicit non-target invariants; (e) reference-run evidence,
     consisting of the actual execution logs of the reference solution, the rendered states,
     and the output artifacts.

(1) Evaluator validity. Family A: the candidate evaluator preserves the parent's goal, target
     content, reward program, and success rule. Family C: the candidate evaluator checks the
     revised instructions, the required deliverables, and the information dependencies they
     introduce.

(2) Task solvability. The logs show a reference solution completing the candidate task in the
     deployed environment using only public or otherwise available inputs, and the candidate
     evaluator gives full credit to the actual output.

(3) Evolution validity. Family A: the intended axis (S, G, N, or I) is realized in rendered
     states or executed interactions, and every listed non-target invariant is preserved.
     Family C: the revised requirement, workflow, or dependency is realized in rendered
     states or executed interactions. Do not require a family C candidate to keep the
     parent's goal or evaluator where the intended change revises them.

(4) Application integration. The evidence shows that the deliverables actually run and save
     on a real machine.
```

```
Evidence rules: Judge only from the recorded evidence. Do not simulate execution, and do not
    infer a pass from code, specifications, or evaluator source alone. A criterion fails if
    its evidence is missing, partial, or inconclusive.
```

Listing 3: System prompt template for the admission gate.

E LIMITATIONS

Our evaluation covers desktop and web applications from the benchmarks in Section 5.1, excluding specialized software such as Blender and computer-aided design (CAD) tools. Application evolution is limited to single-axis changes in appearance, geometry, naming, or interaction, and workflow evolution to revised or new workflows, which cannot capture real software releases that combine changes or modify underlying functionality. The self-improving training loop has not been extended to mobile environments. Extending the self-improving loop to larger-scale, longer-horizon deployment and more diverse multimodal or domain-specific tasks remains future work.