EvoCUA: Evolving Computer Use Agents via Learning from Scalable Synthetic Experience

Taofeng Xue [*†1] **Chong Peng** [*†1] **Mianqiu Huang** [*1 2] **Linsen Guo** [1] **Tiancheng Han** [1 3] **Haozhe Wang** [1 4] **Xiaocheng Zhang** [1] **Xin Yang** [1] **Dengchang Zhao** [1] **Jinrui Ding** [1] **Xiandi Ma** [1] **Yuchen Xie** [1] **Peng Pei** [1] **Xunliang Cai** [1] **Xipeng Qiu** [2]

Abstract

The development of native computer-use agents (CUA) represents a significant leap in multimodal AI. However, their potential is currently bottlenecked by the constraints of static data scaling. Existing paradigms relying primarily on passive imitation of static datasets struggle to capture the intricate causal dynamics inherent in long-horizon computer tasks. In this work, we introduce EvoCUA, a native agentic model that integrates data generation and policy optimization into a self-sustaining evolutionary cycle. This approach employs a verifiable synthesis engine to autonomously generate diverse tasks with executable validators, alongside a scalable infrastructure orchestrating tens of thousands of sandbox rollouts for mass experience acquisition. To internalize this experience, our iterative evolving learning strategy reinforces successful routines while transforming failure trajectories into rich supervision through error analysis and self-correction.

EvoCUA achieves a 56.7% success rate on the OSWorld benchmark, establishing a new open-source state-of-the-art. It outperforms the previous best open-source model, OpenCUA-72B (45.0%), and surpasses leading closed-weights models such as UI-TARS-2 (53.1%). These results demonstrate the generalizability of the evolving paradigm across foundation models of varying scales, establishing a robust and scalable path for advancing native agent capabilities.

[1]Longcat Team, Meituan [2]Fudan University [3]Tongji University [4]The Hong Kong University of Science and Technology. Correspondence to: Taofeng Xue <xuetaofeng@meituan.com>, Chong Peng <pengchong@meituan.com>.

Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

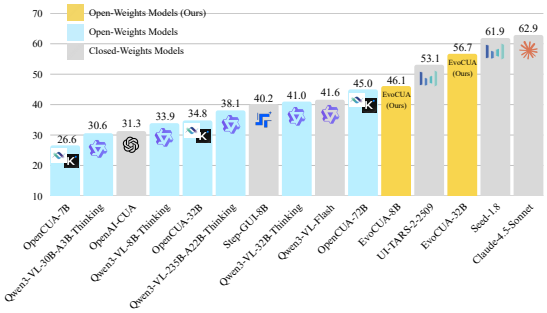


Figure 1. **Performance comparison on the OSWorld-Verified benchmark.** Our **EvoCUA-32B** achieves state-of-the-art performance (56.7%) among open-weights models in Jan 2026.

1. Introduction

The development of generalist agents capable of mastering Graphical User Interfaces (GUIs) represents a pivotal milestone toward artificial general intelligence. Unlike specialized tools, these agents must perceive complex visual contexts and execute long-horizon workflows across heterogeneous applications, effectively emulating human-computer interaction. While recent native vision-language models (VLMs) have successfully integrated perception and action into end-to-end architectures (Bai et al., 2025a; ByteDance Seed Team, 2025), achieving human-level reliability remains a significant challenge. Despite the foundational architectures established by state-of-the-art efforts such as UI-TARS-2 (Wang et al., 2025a), and OpenCUA (Wang et al., 2025b), further progress is increasingly constrained by a critical bottleneck: the diminishing returns of scaling with static datasets.

Existing scaling laws are largely confined to passive imitation of fixed, non-interactive datasets, failing to capture the causal feedback inherent in real-world computer use. Overcoming this limitation necessitates a paradigm shift from data scaling via static traces to experience scaling via massive interactive rollouts. Dynamic experience provides a richer supervisory signal than static text, encompassing environmental feedback and critical insights from both success and failure. However, transforming raw interaction

into a self-improving learning loop presents three primary challenges: 1) *Verifiable data synthesis*. Merely synthesizing textual queries often leads to hallucinations, where the agent generates plausible plans for infeasible tasks. Consequently, a robust framework is essential to ensure that generated queries are strictly grounded in solvable states, aligning with the principles of verifiable rewards. 2) *Scalable interaction infrastructure*: High-throughput experience production demands a unified system that integrates massive environment simulation with high-performance reinforcement learning to support continuous, asynchronous interaction. 3) *Efficient training recipe*: Given an large-scale interaction space, unconstrained exploration is computationally prohibitive. Effective learning requires an on-policy approach that mimics human learning dynamics: consolidating mastered routines while focusing intensely on boundary tasks where the agent oscillates between success and failure.

To address these issues, in this report, we introduce EvoCUA, a native computer use agent that addresses these challenges through the evolving paradigm driven by learning from experience. As illustrated in Figure 2, by unifying verifiable synthesis, high-throughput infrastructure, and evolutionary optimization, EvoCUA establishes a self-sustaining cycle that continuously transforms synthetic compute into high-quality agent capabilities. Our core contributions are threefold:

- **Verified Synthesis Infrastructure.** We develop a unified "Generation-as-Validation" framework that integrates high-performance simulation directly into the task synthesis pipeline. By orchestrating tens of thousands of concurrent sandbox sessions, this engine provides deterministic, state-based supervision signals at scale, ensuring strict grounding and eliminating reward ambiguity through experience acquisition.

- **Evolving Learning from Experience.** We introduce an active learning paradigm centered on the continuous evolution of experience. Through environmental exploration, the model contrasts successful versus failed trajectories to autonomously consolidate effective behavioral patterns. This iterative cycle enables the agent to rectify errors and transcend the limitations of static imitation.

- **Open-Source SOTA Performance.** Empirical evaluations demonstrate that EvoCUA achieves a state-of-the-art success rate of 56.7% on the OSWorld benchmark, significantly outperforming the previous open-source SOTA, OpenCUA-72B (45.0%) (Wang et al., 2025b), and surpassing leading closed-source models UI-TARS-2(53.1%) (Wang et al., 2025a).

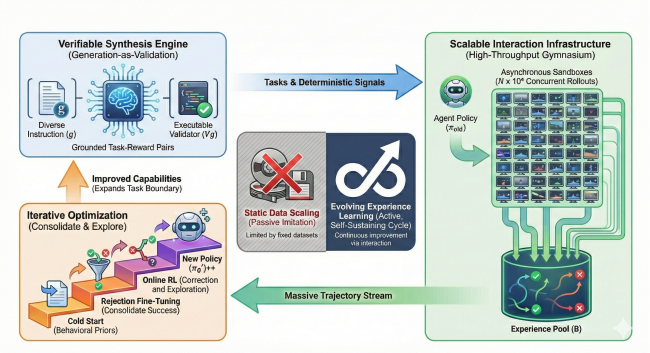


Figure 2. Overview of **EvoCUA**. The approach unifies three core modules: the **Verifiable Synthesis Engine** (top left); the **Scalable Interaction Infrastructure** (right); and **Iterative Optimization** (bottom left).

2. Preliminaries

We model computer use as a Partially Observable Markov Decision Process (POMDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}_{syn})$.

State and Observation. The environment is characterized by an underlying system state $s_t \in \mathcal{S}$ (e.g., kernel registers, background processes), which is not directly accessible to the agent. Instead, the agent perceives a visual observation $o_t \in \mathcal{O}$ rendered from the latent state: $o_t = \mathrm{Render}(s_t)$, typically as a high-resolution screenshot. This partial observability necessitates the use of an interaction history $h_t = \{g, o_0, z_0, a_0, \ldots, o_{t-1}, z_{t-1}, a_{t-1}\}$ to disambiguate the current environmental context.

Action and Thought Space. The agent operates in a unified native action space $\mathcal{A} = \mathcal{A}_{\text{mouse}} \cup \mathcal{A}_{\text{keyboard}} \cup \mathcal{A}_{\text{control}}$. To bridge the gap between raw pixels and symbolic actions, we introduce an explicit thought space $\mathcal{Z}$. At each step t, the policy $\pi_\theta(z_t, a_t \mid h_t, o_t)$ follows a Reason-then-Act sequence: it first generates a linguistic reasoning trace $z_t \in \mathcal{Z}$ to interpret o_t, then selects an executable action $a_t \in \mathcal{A}$.

Verifiable Reward ($\mathcal{R}_{syn}$): Supervision is grounded in execution correctness. For instruction g, a validator V_g provides a sparse binary reward based on the terminal state s_T, defined as $\mathcal{R}_{syn}(s_T; g) \triangleq \mathbb{I}[V_g(s_T) = \text{True}]$.

Objective: We maximize the expected success rate $J(\theta) = \mathbb{E}_{g \sim \mathcal{T}_{syn}, \tau \sim \pi_\theta}[\mathcal{R}_{syn}(s_T; g)]$ over an adaptive task distribution $\mathcal{T}_{syn}(\cdot \mid \pi_{\text{old}})$, where τ denotes trajectories induced by π_θ under instruction g.

3. Verifiable Synthesis Infrastructure

In this section, we introduce a verifiable synthesis infrastructure which focuses on overcoming inherent limitations such as reward hacking and the absence of precise train-

ing signals. We implement a "generation-as-validation" paradigm. For any synthesized instruction g, the infrastructure co-generates a deterministic and executable validator V_g. This ensures that the reward signal $\mathcal{R}_{syn}(s_T; g)$ is derived from a strict verification of the final environment state. This infrastructure combines a three-stage synthesis pipeline with a scalable sandbox environment.

3.1. Agentic Data Synthesis Engine

The synthesis engine is organized into three modules: structured task space construction, agentic dual-stream synthesis, and rigorous quality assurance illustrated in 3.

Structured Task Space Construction. To ensure the synthesized distribution $\mathcal{T}_{syn}$ captures the complexity of real-world computer use, we first establish a structured task space decomposed into domains and resources. We argue that atomic capabilities are inherently transferable and form complex tasks through composition. Guided by this principle, we systematically categorize core desktop applications such as Web Browsers, Excel, and Word, and decompose user behaviors into *atomic capabilities*. This orthogonal decomposition enables the agent to generalize to diverse scenarios through the recombination of primitive skills. For instance, a financial analysis task in Excel is decomposed into sub-skills such as formula manipulation, data sorting, and chart generation. Leveraging a hierarchical domain taxonomy, we synthesized a wide range of task scenarios featuring diverse user personas (Ge et al., 2024) to ensure data diversity. Synthesized scenarios range from educators designing lecture slides to algorithm engineers conducting technical literature surveys.

To bridge the simulation-to-reality gap, we implement a hybrid strategy for the environment initial state. First, for *parametric synthesis*, we use code-based generators to batch-produce documents such as Word, Excel, or PDF files by parameterizing variables like names, prices, and dates. Second, for *non-parametric injection*, we inject public internet data including images, audio, and complex slides. This forces the agent to handle the visual noise and structural diversity inherent in real-world files.

Agentic Dual-Stream Synthesis. The core synthesis process is modeled as a ReAct-based agentic workflow (Yao et al., 2022). Given a sampled scenario tuple, a foundation VLM functions as a *task architect* to execute a dual-stream generation. The **Instruction stream** (g) formulates a natural language query grounded in the specific resource context to ensure user intent is clear and achievable. At the same time, the **Validator stream** (V_g) generates the ground truth and the corresponding executable evaluator code. This code defines the precise success conditions for the task (Yang et al., 2025).

To guarantee executability, we enforce a **closed-loop feedback mechanism**. The generated code is immediately executed in a real sandbox environment. The execution results, including output files from successful runs and error messages from failed executions, are fed back to the model to evaluate the quality of the files and the evaluator. This process iterates for multiple rounds until the execution succeeds and passes quality checks. To further enhance stability, we abstract frequently used verification logic into a standardized tool library. Finally, the valid tuple is formatted into a standardized JSON structure compatible with established benchmarks like OSWorld.

Quality Assurance. To eliminate false labels and data leakage, we implement a multi-stage filtering protocol. We first employ **consistency-based filtering** via sandbox rollouts; tasks failing due to configuration issues are returned for agentic refinement. For successful rollouts, we cross-verify outcomes using both a reward model and an executable evaluator. Significant discrepancies between these signals trigger manual spot checks based on our domain taxonomy. This cross-verification loop—integrating execution results, reward models, and human audit—ensures only high-fidelity, grounded tasks remain in the final dataset.

Furthermore, we enforce a **Tri-fold decontamination** process to prevent inflated metrics. This includes (1) *semantic decontamination* to remove instructions semantically equivalent to benchmark queries, (2) *configuration decontamination* to prune tasks with identical application initialization settings, and (3) *evaluator decontamination* to verify that the generated success conditions do not overlap with existing evaluation scripts. Through this pipeline, we have successfully scaled verifiable training data to tens of thousands of instances.

3.2. Scalable Sandbox Environment

To support massive-scale interaction and validation, we develop a high-throughput, deterministic sandbox infrastructure designed for industrial-level experience acquisition.

Elastic Management: The environment employs an asynchronous gateway based on the reactor pattern to decouple control logic from execution. This architecture, managed by a distributed scheduler, can bootstrap $10,000$ instances per minute and stably sustain over $100,000$ concurrent sandboxes.

Deterministic Calibration: We utilize a hybrid virtualization approach (QEMU-KVM within Docker) to balance strict isolation with near-native GUI performance. To ensure validation stability across millions of runs, the environment is calibrated for input determinism and rendering consistency through kernel-level patches and unified font caching. This ensures that agent actions and system feedback remain

precise and reproducible. For a comprehensive discussion of the infrastructure's technical stack, please refer to Appendix C.

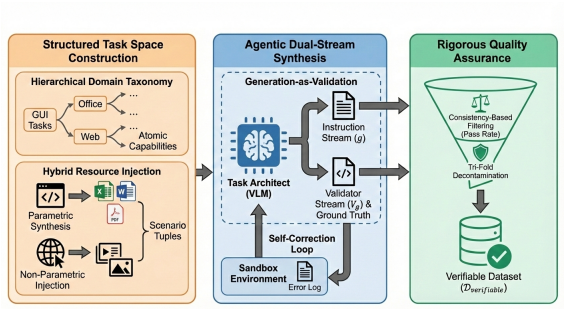


Figure 3. Architecture of the Verifiable Synthesis Engine: (1) **Structured Task Space** to define diverse scenarios from domain taxonomies and hybrid resources; (2) **Agentic Dual-Stream Synthesis**, where a VLM co-generates instructions (g) and executable validators (V_g); and (3) **Quality Assurance** to filter outputs for high consistency and ensure decontamination

4. Evolving Learning from Experience

To bridge the gap between atomic imitation and generalist problem-solving, we propose the evolving paradigm via learning from experience. This paradigm shifts from static data scaling to a dynamic capability evolution cycle. The process is structured into three progressive stages: a supervised cold-start to establish behavioral priors, rejection sampling fine-tuning to consolidate successful experiences via adaptive scaling, and reinforcement learning to rectify failures and explore complex dynamics through interaction.

4.1. Cold-Start

To initialize the policy π_{init} with a robust behavioral prior, we first formally define the unified action and thought spaces to establish the structural bounds of the agent, and subsequently leverage these definitions to synthesize and format grounded interaction data.

Unifying the Action Space ($\mathcal{A}$). We implement **Semantic Action Mapping** to construct a unified action space $\mathcal{A} = \mathcal{A}_{\text{mouse}} \cup \mathcal{A}_{\text{keyboard}} \cup \mathcal{A}_{\text{control}}$ as illustrated in Appendix D. We categorize raw event streams into two primary components:

- *Physical Interaction ($\mathcal{A}_{mouse} \cup \mathcal{A}_{keyboard}$)*: This component encompasses coordinate-based mouse events and keyboard inputs. Crucially, to support complex, multi-step operations, we implement a stateful interaction mechanism. By decoupling discrete key presses into `key_down` and `key_up` events, the policy can maintain active states (e.g., holding modifiers like `Shift` for multi-selection) required for complex tasks.
- *Control Primitives ($\mathcal{A}_{control}$)*: We introduce meta-actions to manage the execution flow distinct from physical I/O. Specifically, the `wait` primitive allows the agent to handle asynchronous UI rendering, while `terminate` serves as a formal signal to conclude the task.

Structuring the Thought Space ($\mathcal{Z}$). To enable interpretable and robust decision-making, we define a reasoning schema for the latent thought space $\mathcal{Z}$. This schema imposes a structured format to ensure the reasoning process strictly aligns with execution logic:

- *Goal Clarification (z_0)*: At the initial step ($t = 0$), the agent is required to explicitly paraphrase the user's objective. This clarifies ambiguous instructions and grounds the subsequent planning process.
- *Observation Consistency (z_{obs})*: To minimize hallucinations, the reasoning trace must include a concise summary of key visual elements. We enforce strict semantic consistency between this textual summary and the actual observed state.
- *Self-Verification (z_{check})*: Before issuing the final termination signal, the agent is prompted to execute auxiliary interaction steps (e.g., checking a file status) to visually confirm that the execution result aligns with the user's instruction.
- *Reflection and Correction ($z_{reflect}$)*: We leverage failed rollouts for error correction. Upon identifying a critical error step in a failed trajectory, we restore the environment to the pre-error state. To account for sandbox non-determinism, we strictly filter for **state consistency** between the restored environment and the original trace. From this valid restored state, we induce self-correction using high-temperature sampling to generate successful remedial paths.
- *Reasoning-Augmented Termination (z_T)*: To prevent the model from overfitting to the termination label, the `terminate` action must be strictly conditional on a preceding reasoning trace. This trace requires the agent to explicitly synthesize visual evidence to justify task completion, ensuring the decision is grounded in logic rather than memorized patterns.

Based on these formalized definitions, we synthesize the prior dataset $\mathcal{D}_{\text{prior}}$ by leveraging foundational vision-language models (e.g., Qwen3-VL, OpenCUA) within a modular framework. Crucially, to ensure alignment between reasoning and action, we employ a **Hindsight Reasoning Generation** strategy. Treating the ground-truth execution path as known future information, we retrospectively generate reasoning traces z_t that explain the observed actions,

thereby augmenting physical trajectories with coherent cognitive chains.

Training Details. For model training, we decompose these multi-turn trajectories into single-turn samples. To balance information density with memory constraints, the input context retains full multimodal details (screenshots, reasoning, and actions) only for the most recent five steps, while earlier history is compressed into text-only semantic actions. The training loss is computed exclusively on the current step's reasoning and action.

Finally, to preserve general foundation capabilities, we incorporate a diverse mixture of general-purpose data, covering STEM, OCR, visual grounding, and text-based reasoning. The volume of this general data is balanced to match the scale of the decomposed single-turn trajectory samples.

Qualitative Analysis. We synthesize trajectory data adhering to this schema. Following cold start training, qualitative analysis confirms the agent effectively masters atomic capabilities as illustrated in Appendix H. However, a critical robustness gap persists in complex scenarios. While the agent can execute standard long-horizon workflows, it exhibits fragility in boundary cases. To address these limitations, we move to the next stage: internalizing scalable, high-quality experiences.

4.2. Rejection Sampling Fine-Tuning

The objective of Rejection Sampling Fine-Tuning (RFT) (Ahn et al., 2024) is to consolidate the agent's ability to solve tasks by learning exclusively from high-quality, successful executions. This process involves two key components: efficiently generating successful trajectories via dynamic compute, and denoising them to maximize the signal-to-noise ratio.

Dynamic Compute Budgeting. To optimize the generation of high-quality experience under computational constraints, we propose dynamic compute budgeting. Instead of uniformly allocating rollout resources, this mechanism adapts the exploration budget to the agent's current proficiency level for each specific task.

We establish a hierarchical budget spectrum $\mathcal{K} = \{k_1, \ldots, k_n\}$ paired with descending success rate thresholds $\Lambda = \{\tau_1, \ldots, \tau_n\}$. For a given task query g drawn from the synthesis engine $\mathcal{T}_{\text{syn}}$, the system identifies the optimal rollout budget K^* that satisfies the sufficiency condition:

$$K^* = k_{i^*} \quad \text{where} \quad i^* = \min\{i \mid \text{SR}(k_i) \geq \tau_i\} \tag{1}$$

Here, $\text{SR}(k_i)$ represents the pass rate observed with budget k_i. This strategy effectively prunes efficiently solved tasks and concentrates computational power on boundary queries—tasks where the policy exhibits high variance.

Step-Level Denoising. Although successful rollouts demonstrate the model's capability, they often contain significant noise. We use a judge model to analyze the trajectories and mask out redundant steps. This filtering is especially important for infeasible tasks; for these, we remove all intermediate actions and strictly keep the reasoning trace and the final `terminate=failure` action. This process refines the raw data into high-quality supervision, which is then aggregated into the experience pool $\mathcal{B}$.

Through this generation and filtering pipeline, we scale our high-fidelity experience pool $\mathcal{B}$ to tens of thousands of trajectories. We interleave this domain-specific experience with a balanced corpus of general-purpose multimodal data to prevent catastrophic forgetting.

4.3. Reinforcement Learning

While RFT consolidates what the agent *can* do, it does not explicitly correct what it *does wrong*. To push the capability boundary, we employ RL to learn from failures.

Standard trajectory-level preference optimization is ill-suited for long-horizon tasks due to state misalignment. We instead propose a Step-Level Direct Preference Optimization strategy (Lai et al., 2024) that targets *Critical Forking Points* illustrated in Figure 4.

Causal Deviation Discovery. Given a failed rollout τ^- and a successful reference τ^+ (retrieved from the same or a semantically equivalent task), we employ a Reference-Guided Diagnosis mechanism. We identify the Critical Deviation Step t^* as the first timestamp where the agent's action diverges from the reference, despite the environmental states remaining functionally equivalent. This isolates the specific response $(z_{t^*}^-, a_{t^*}^-)$ that caused the agent to leave the optimal solution manifold.

Structured Preference Construction. Once the critical error $(z_l, a_l) = (z_{t^*}^-, a_{t^*}^-)$ is identified, we construct preference pairs to provide comprehensive supervision.

Paradigm I: Action Correction (At Step t^*). The objective is to replace the rejected error (z_l, a_l) with an optimal chosen response (z_w, a_w). We obtain (z_w, a_w) via window-based reference alignment (migrating thoughts and actions from τ^+ via VLM semantic matching) or visual-grounded synthesis (synthesizing fresh traces via a general model when no alignment exists).

Paradigm II: Reflection and Recovery (At Step t^*+1). To improve robustness, we address the state immediately *after* the error $(t^* + 1)$. We treat the agent's blind continuation as the rejected sample. For the chosen sample, we synthesize a reflection trace. Instead of acting blindly, the agent is trained to halt and generate a reasoning chain that: (1) observes the unexpected screen state, and (2) formulates a remedial plan.

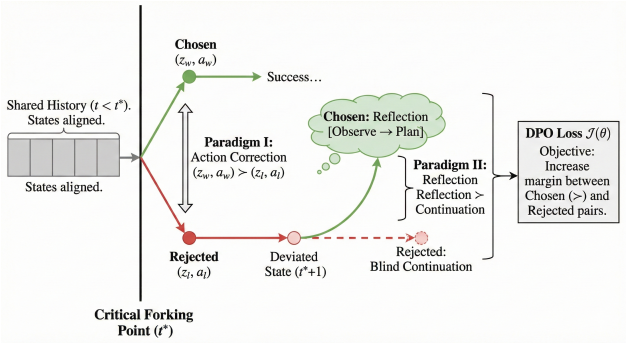


Figure 4. **Dual-Paradigm DPO Overview.** At forking point t^*, **Paradigm I** (Action Correction) optimizes for the correct action a_w over a_l. **Paradigm II** (Reflection) optimizes the subsequent state $t^* + 1$ by prioritizing reflective reasoning over blind continuation.

Optimization Objective. We optimize the policy π_θ using Direct Preference Optimization (DPO). Consistent with our formulation where the policy generates a reasoning trace z and an action a conditioned on history h_t and observation o_t, the loss function is defined as:

$$\mathcal{J}(\theta) = -\mathbb{E}_{(h_t, o_t, (z,a)_w, (z,a)_l) \sim \mathcal{D}} \log \sigma\left(\beta H_t\right)$$
$$H_t = \log \frac{\pi_\theta(z_w, a_w | h_t, o_t)}{\pi_{\text{ref}}(z_w, a_w | h_t, o_t)} - \log \frac{\pi_\theta(z_l, a_l | h_t, o_t)}{\pi_{\text{ref}}(z_l, a_l | h_t, o_t)} \quad (2)$$

By iteratively updating the policy with these structured preferences, EvoCUA continuously expands its capability boundary, effectively converting transient interaction experience into robust model parameters.

5. Evaluation

In this section, we conduct a comprehensive empirical evaluation of EvoCUA.

5.1. Experimental Setup

Our training pipeline evolves from a lightweight cold-start (1k trajectories) to a continuous iterative cycle. We expand the training distribution via large-scale rejection sampling with step-level denoising, while refining the policy through preference learning from errors.

We validate our approach across varying scales by post-training on the **Qwen3-VL-Thinking** (Bai et al., 2025a) (8B, 32B) and **OpenCUA** (Wang et al., 2025b) (7B, 32B, 72B) foundation models.

5.2. Main Results

We evaluate EvoCUA on the **OSWorld** benchmark, which serves as a representative testbed for open-ended computer use tasks. As summarized in Table 1, our results highlight the effectiveness of the proposed method:

SOTA Open-Weights Performance: EvoCUA-32B achieves a **56.7%** success rate. It delivers substantial gains over baselines, improving by **+15.7%** over its Qwen3-VL-32B-Thinking backbone (41.0%) and surpassing the previous best open-weights OpenCUA-72B (45.0%) by **+11.7%**. These results are achieved within a 50-step budget, demonstrating efficiency compared to the 100-step baselines.

Competitive with Closed-Weights Frontiers: The model effectively closes the gap with proprietary models. It outperforms the strong baseline UI-TARS-2-2509 (53.1%) by **+3.6%** and narrows the performance margin to the industry-leading Claude-4.5-Sonnet (58.1%) to a mere **1.4%** under equivalent constraints.

Training and Scaling Superiority: EvoCUA-8B (46.1%) proves highly parameter-efficient, outperforming specialized 72B models. In a controlled comparison using the Qwen3-VL-8B backbone, it achieves a **+5.9%** improvement over Step-GUI-8B (40.2%), strictly isolating and validating the effectiveness of our evolving learning paradigm.

We further evaluate EvoCUA across GUI grounding benchmarks and general multimodal tasks. As detailed in Appendix G, while some thinking-based variants (e.g., EvoCUA-32B) show a slight shift in general metrics due to training data distribution differences, the core agentic capabilities remain robust.

5.3. Cross-Environment Generalization

To evaluate the zero-shot cross-OS generalization capabilities of our approach, we test EvoCUA on the **WindowsAgentArena** (WAA) (Bonatti et al., 2024) benchmark. Because our training pipeline relies entirely on a Linux-based environment, evaluating on a Windows OS presents a rigorous test of out-of-distribution, real-world applicability.

As demonstrated in Table 5.3, EvoCUA exhibits strong cross-environment transferability. EvoCUA-32B achieves a success rate of **56.48%**, significantly outperforming its base model, Qwen3-VL-32B-Thinking (42.9%), by **+13.58%**. Furthermore, it surpasses the current leading frontier GUI agent, UI-TARS-2 (50.6%), by nearly 6 absolute points. These results confirm that the structured action schemas and reasoning priors acquired through our evolutionary learning paradigm are not overfitted to a specific platform, but instead generalize effectively to entirely novel operating systems and visual interfaces.

5.4. Ablation Study

To rigorously verify the contribution of each component within the EvoCUA, we conducted extensive ablation studies on two distinct foundation models, Qwen3-VL-32B-Thinking and OpenCUA-72B.

Table 1. **Performance comparison on the OSWorld-Verified benchmark.** Models are categorized by accessibility (Closed-Weights vs. Open-Weights). **Max Steps** denotes the interaction budget per task. **EvoCUA-32B** achieves state-of-the-art performance among open models, significantly outperforming larger baselines.

Model	Type	Max Steps	Success Rate (Pass@1)
Closed-Weights Models			
OpenAI CUA (OpenAI, 2025)	Specialized	50	31.3%
Step-GUI-8B (Yan et al., 2025)	Specialized	100	40.2%
Qwen3-VL-Flash (Bai et al., 2025a)	General	100	41.6%
UI-TARS-2-2509 (Wang et al., 2025a)	General	100	53.1%
Claude-4.5-Sonnet (Anthropic, 2025)	General	50	58.1%
Seed-1.8 (ByteDance Seed Team, 2025)	General	100	61.9%
Claude-4.5-Sonnet (Anthropic, 2025)	General	100	**62.9%**
Open-Weights Models			
Qwen2.5-VL-32B-Instruct (Bai et al., 2025b)	General	100	5.9%
Qwen2.5-VL-72B-Instruct (Bai et al., 2025b)	General	100	8.8%
ScaleCUA-32B (Liu et al., 2025)	Specialized	50	17.7%
UI-TARS-72B-DPO (Qin et al., 2025)	Specialized	50	24.6%
OpenCUA-7B (Wang et al., 2025b)	Specialized	100	26.6%
UI-TARS-1.5-7B (Qin et al., 2025)	Specialized	100	27.5%
Qwen3-VL-8B-Thinking (Bai et al., 2025a)	General	100	30.6%
OpenCUA-32B (Wang et al., 2025b)	Specialized	100	34.8%
GUI-Owl-7B-Desktop-RL (Ye et al., 2025)	Specialized	15	34.9%
Qwen3-VL-235B-A22B Thinking (Bai et al., 2025a)	General	100	38.1%
Qwen3-VL-32B-Thinking (Bai et al., 2025a)	General	100	41.0%
OpenCUA-72B (Wang et al., 2025b)	Specialized	100	45.0%
EvoCUA-8B (Ours)	**General**	**50**	**46.1%**
EvoCUA-32B (Ours)	**General**	**50**	**56.7%**

Table 2. Zero-shot performance on WindowsAgentArena (WAA). This evaluates the models' ability to generalize from a Linux-based training distribution to a Windows OS environment.

Model	Success (%)
Qwen3-VL-32B-Instruct (Bai et al., 2025a)	30.90
Qwen3-VL-32B-Thinking (Bai et al., 2025a)	42.90
UI-TARS-2 (Wang et al., 2025a)	50.60
EvoCUA-32B (Ours)	**56.48**

5.4.1. Component Analysis on EvoCUA-32B

We adopt Qwen3-VL-32B-Thinking as our base checkpoint to dissect the cumulative gains from the Unified Action Space, Cold Start, Rejection Fine-Tuning (RFT), and RL. As shown in Table 3, each stage of the evolutionary cycle yields significant monotonic improvements.

Action Space & Cold Start. Controlled experiments show that the unified action space contributes a **+4.84%** improvement. Incorporating behavioral priors through cold-start training on synthesized traces adds **+2.62%**, suggesting that structured action schemas and reasoning patterns facilitate subsequent experience learning.

RFT & DPO. RFT increases performance by **+3.13%** through successful experience consolidation. DPO further improves the success rate by **+3.21%** by addressing identified failure modes. A second iteration of RFT and DPO yields an additional **+1.90%**, indicating that the model continues to refine its capability boundary through recursive synthesis and correction.

5.4.2. Generalizability on OpenCUA-72B

To verify the universality of our approach, we applied the same paradigm to the larger OpenCUA-72B model as detailed in Table 3.

Table 3. Detailed ablation study across different model scales. We report the absolute performance gain (Δ) relative to the preceding stage.

Stage	Improvement (Δ)	
	EvoCUA-32B	**OpenCUA-72B**
+ Unified Action Space	**+4.84%**	–
+ Cold Start	**+2.62%**	**+2.14%**
+ RFT	**+3.13%**	**+3.69%**
+ Offline DPO	**+3.21%**	**+3.02%**
+ Iterative Training	**+1.90%**	**+1.82%**

The results on OpenCUA-72B echo our findings on Qwen3-VL, with DPO (+3.02%) and RFT (+3.69%) providing strong contributions. In addition, OpenCUA-72B adopts the standard pyautogui format. This action space natively sup-

ports stateful operations (such as shift+click) and possesses no obvious functional deficiencies.

5.5. Scaling Analysis

We investigate the scalability of EvoCUA by analyzing the performance gain ($\Delta\%$) across varying Pass@k values, max inference steps and data volume.

Scaling with Pass@k. In Figure 5a, EvoCUA maintains a consistent performance lead over the base model (Qwen3-VL-Thinking) across all Pass@k metrics. The 32B model sustains a positive gain, peaking at +4.93% at $k = 16$ and maintaining a significant advantage even at higher k values. This consistent gap demonstrates that our training strategy optimizing the action space and reasoning priors fundamentally elevates the model's performance ceiling.

Scaling with Max Steps. In Figure 5b, We observe that performance steadily improves as the maximum step limit increases. Increasing the inference capacity from 15 to 50 steps leads to consistent gains, with the 32B model achieving a +16.25% improvement over the baseline. Beyond 50 steps, the rate of improvement moderates, primarily due to the scarcity of trajectories exceeding 50 steps in the current training distribution.

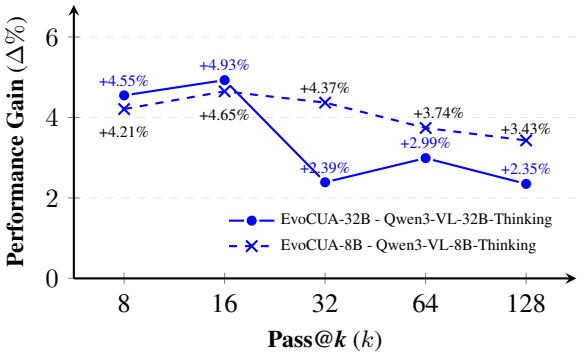
Experience Scaling. We evaluate the scalability of RFT through an ablation study on OpenCUA-72B, focusing on multi-round iteration without cold-start or DPO phases (Table 4). The performance gains scale consistently across three stages: *Round 1* yields a **+2.61 pp** gain using 20k samples; *Round 2*, initialized from the Round 1 checkpoint, increases the gain to **+6.79 pp** with 226k samples; and *Round 3* achieves a **+8.12 pp** improvement by training the OpenCUA-72B base model from scratch on 1M samples aggregated from all iterations.

Our analysis reveals a trade-off between data scale and the signal-to-noise ratio (SNR). As models scale, their noise tolerance drops, bottlenecking iterative methods. Continued scaling is feasible only if data quality, on-policy alignment, and SNR are prioritized.

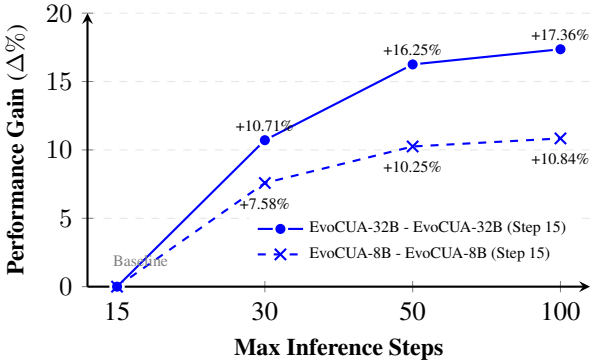
Given the extensive scale of our experiments, we have distilled several **key insights** in Appendix B.

Table 4. Experience scaling results on RFT. The absolute gains are all relative to the baseline.

Stage	Data Size	Gain ($\Delta\%$)
RFT Round 1	20k	+2.61
RFT Round 2	226k	+6.79
RFT Round 3	1M	+8.12



(a) Performance Gain across Pass@k



(b) Scaling with Inference Steps

Figure 5. **Performance analysis of EvoCUA models.** (a) Improvement over the base model across varying Pass@k metrics. Legends indicate the specific backbone models used. (b) Performance scaling with increased maximum inference steps. The legends denote the performance gain relative to the Step 15 baseline.

6. Related Work

Foundation VLMs and Computer Use Capabilities. The landscape of Large Visual Language Models (VLMs) has rapidly evolved to support complex agentic tasks. Proprietary frontier models, most notably Claude 4.5 Sonnet (Anthropic, 2025) and Seed 1.8 (ByteDance Seed Team, 2025), have set the industry standard, demonstrating human-level proficiency in zero-shot instruction following and long-horizon planning. In the open-weight domain, Qwen3-VL (Bai et al., 2025a) has emerged as a robust backbone, introducing next-generation dynamic resolution and enhanced OCR capabilities. EvoCUA builds directly on the Qwen3-VL architecture, enhancing it via a specialized evolutionary post-training curriculum to transcend general-purpose pre-training limitations.

Generalist GUI Agents and Benchmarks. To evaluate online agent performance, OSWorld (Xie et al., 2024) serve as primary testbeds. OpenCUA (Wang et al., 2025b) establishes a critical foundation with the AgentNet dataset, while state-of-the-art efforts like UI-TARS-2 (Wang et al.,

2025a) and Step-GUI (Yan et al., 2025) utilize multi-turn RL and step-wise visual reasoning, respectively. Unlike these demonstration-heavy approaches, EvoCUA utilizes autonomously synthesized, verifiable experiences to reduce annotation costs while achieving superior performance on the OSWorld leaderboard.

Visual Grounding and Action Execution. Precise GUI grounding is essential for action execution. Following early works like Aguvis (Xu et al., 2024), recent models such as ShowUI (Lin et al., 2025) and UGround (Gou et al., 2024) have optimized architectures for high-resolution layouts. EvoCUA integrates these grounding-specialized insights to establish robust primitives before optimizing high-level planning.

From Imitation to Experiential Learning. Training paradigms are shifting from Behavior Cloning toward Reinforcement Learning (RL). Beyond standard PPO (Schulman et al., 2017), the Reinforcement Learning with Verifiable Rewards (RLVR) paradigm—pioneered by DeepSeek-R1 (Guo et al., 2025)—enables verifying reasoning chains without dense supervision. Recent advances include Group-in-Group optimization (Feng et al., 2025) and reward-free "Early Experience" (Zhang et al., 2025). EvoCUA extends this by addressing data scarcity through a verifiable synthesis engine. This creates a self-sustaining cycle of rejection sampling and preference learning on ground-truth-verified synthetic trajectories, enabling the agent to iteratively learn from experience.

7. Conclusion

In this work, we present EvoCUA, a native computer use agent developed through the evolving paradigm via learning from experience. By integrating verifiable synthesis with a scalable interaction infrastructure, we demonstrate the efficacy of converting synthetic compute into high-quality training signals. Empirical evaluations on the OSWorld benchmark validate this approach, with EvoCUA achieving a success rate of 56.7%, establishing a new state-of-the-art among open-weights models.

Despite these advancements, a performance gap persists between current open models and leading closed-weights systems or human-level reliability. This disparity highlights the limits of offline learning from synthesized traces alone. To address this, our preliminary investigation into online reinforcement learning identifies active environmental interaction as a critical driver for further improvement, evidenced by a consistent upward trend in reward accumulation shown in Appendix A. Future work will focus on systematically expanding this online evolutionary boundary, aiming to achieve fully autonomous computer use capabilities.

References

Ahn, J., Verma, R., Lou, R., Liu, D., Zhang, R., and Yin, W. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

Anthropic. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, 2025. Accessed: 2025-10-31.

Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., Ge, W., Guo, Z., Huang, Q., Huang, J., Huang, F., Hui, B., Jiang, S., Li, Z., Li, M., Li, M., Li, K., Lin, Z., Lin, J., Liu, X., Liu, J., Liu, C., Liu, Y., Liu, D., Liu, S., Lu, D., Luo, R., Lv, C., Men, R., Meng, L., Ren, X., Ren, X., Song, S., Sun, Y., Tang, J., Tu, J., Wan, J., Wang, P., Wang, P., Wang, Q., Wang, Y., Xie, T., Xu, Y., Xu, H., Xu, J., Yang, Z., Yang, M., Yang, J., Yang, A., Yu, B., Zhang, F., Zhang, H., Zhang, X., Zheng, B., Zhong, H., Zhou, J., Zhou, F., Zhou, J., Zhu, Y., and Zhu, K. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., Zhong, H., Zhu, Y., Yang, M., Li, Z., Wan, J., Wang, P., Ding, W., Fu, Z., Xu, Y., Ye, J., Zhang, X., Xie, T., Cheng, Z., Zhang, H., Yang, Z., Xu, H., and Lin, J. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Bonatti, R., Zhao, D., Bonacci, F., Dupont, D., Abdali, S., Li, Y., Lu, Y., Wagle, J., Koishida, K., Bucker, A., et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

ByteDance Seed Team. Seed 1.8. https://github.com/ByteDance-Seed/Seed-1.8/, 2025. GitHub repository.

Chen, L., Li, J., Dong, X., Zhang, P., Zang, Y., Chen, Z., Duan, H., Wang, J., Qiao, Y., Lin, D., and Zhao, F. Are we on the right way for evaluating large vision-language models?, 2024. URL https://arxiv.org/abs/2403.20330.

Feng, L., Xue, Z., Liu, T., and An, B. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Ge, T., Chan, X., Wang, X., Yu, D., Mi, H., and Yu, D. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*, 2024.

Gou, B., Wang, R., Zheng, B., Xie, Y., Chang, C., Shu, Y., Sun, H., and Su, Y. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Lai, X., Tian, Z., Chen, Y., Yang, S., Peng, X., and Jia, J. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*, 2024.

Li, K., Ziyang, M., Lin, H., Luo, Z., Tian, Y., Ma, J., Huang, Z., and Chua, T.-S. Screenspot-pro: GUI grounding for professional high-resolution computer use. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=XaKNDIAHas.

Lin, K. Q., Li, L., Gao, D., Yang, Z., Wu, S., Bai, Z., Lei, S. W., Wang, L., and Shou, M. Z. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 19498–19508, 2025.

Liu, Y., Li, Z., Huang, M., Yang, B., Yu, W., Li, C., Yin, X.-C., Liu, C.-L., Jin, L., and Bai, X. Ocrbench: on the hidden mystery of ocr in large multimodal models. *Science China Information Sciences*, 67 (12), December 2024. ISSN 1869-1919. doi: 10.1007/s11432-024-4235-6. URL http://dx.doi.org/10.1007/s11432-024-4235-6.

Liu, Z., Xie, J., Ding, Z., Li, Z., Yang, B., Wu, Z., Wang, X., Sun, Q., Liu, S., Wang, W., Ye, S., Li, Q., Dong, X., Yu, Y., Lu, C., Mo, Y., Yan, Y., Tian, Z., Zhang, X., Huang, Y., Liu, Y., Su, W., Luo, G., Yue, X., Qi, B., Chen, K., Zhou, B., Qiao, Y., Chen, Q., and Wang, W. Scalecua: Scaling open-source computer use agents with cross-platform data. 2025. URL https://github.com/OpenGVLab/ScaleCUA.

Lu, P., Bansal, H., Xia, T., Liu, J., Li, C., Hajishirzi, H., Cheng, H., Chang, K.-W., Galley, M., and Gao, J. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts, 2024. URL https://arxiv.org/abs/2310.02255.

OpenAI. Computer-using agent (cua). https://openai.com/index/computer-using-agent/, 2025. Accessed: 2025-10-01.

Qin, Y., Ye, Y., Fang, J., Wang, H., Liang, S., Tian, S., Zhang, J., Li, J., Li, Y., Huang, S., et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Wang, H., Zou, H., Song, H., Feng, J., Fang, J., Lu, J., Liu, L., Luo, Q., Liang, S., Huang, S., et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025a.

Wang, X., Wang, B., Lu, D., Yang, J., Xie, T., Wang, J., Deng, J., Guo, X., Xu, Y., Wu, C. H., et al. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025b.

Wu, Z., Wu, Z., Xu, F., Wang, Y., Sun, Q., Jia, C., Cheng, K., Ding, Z., Chen, L., Liang, P. P., et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Xie, T., Deng, J., Li, X., Yang, J., Wu, H., Chen, J., Hu, W., Wang, X., Xu, Y., Wang, Z., Xu, Y., Wang, J., Sahoo, D., Yu, T., and Xiong, C. Scaling computer-use grounding via user interface decomposition and synthesis, 2025. URL https://arxiv.org/abs/2505.13227.

Xu, Y., Wang, Z., Wang, J., Lu, D., Xie, T., Saha, A., Sahoo, D., Yu, T., and Xiong, C. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

Yan, H., Wang, J., Huang, X., Shen, Y., Meng, Z., Fan, Z., Tan, K., Gao, J., Shi, L., Yang, M., et al. Step-gui technical report. *arXiv preprint arXiv:2512.15431*, 2025.

Yang, Y., Yang, Z., Dou, Z.-Y., Nguyen, A., You, K., Attia, O., Szot, A., Feng, M., Ramrakhya, R., Toshev, A., et al. Ultracua: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*, 2025.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Ye, J., Zhang, X., Xu, H., Liu, H., Wang, J., Zhu, Z., Zheng, Z., Gao, F., Cao, J., Lu, Z., et al. Mobile-agent-v3: Foundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

Yue, X., Ni, Y., Zhang, K., Zheng, T., Liu, R., Zhang, G., Stevens, S., Jiang, D., Ren, W., Sun, Y., Wei, C., Yu, B., Yuan, R., Sun, R., Yin, M., Zheng, B., Yang, Z., Liu, Y., Huang, W., Sun, H., Su, Y., and Chen, W. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi, 2024. URL https://arxiv.org/abs/2311.16502.

Yue, X., Zheng, T., Ni, Y., Wang, Y., Zhang, K., Tong, S., Sun, Y., Yu, B., Zhang, G., Sun, H., Su, Y., Chen, W., and Neubig, G. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark, 2025. URL https://arxiv.org/abs/2409.02813.

Zhang, K., Chen, X., Liu, B., Xue, T., Liao, Z., Liu, Z., Wang, X., Ning, Y., Chen, Z., Fu, X., et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025.

Statement

During the paper's preparation, we utilized generative AI tools for text structuring and language refinement. All content was manually reviewed, revised, and validated by the authors to guarantee accuracy, logical coherence, and academic rigor. The paper's core ideas, research design, experimental validation, and conclusion derivation are the authors' independent work. Additionally, Figures 2, 3, 4, and 7 were generated using generative AI tools, with the authors overseeing parameter configuration, content screening, and finalization to align with the research's expressive and academic requirements.

This research seeks to advance machine learning by enriching its theoretical framework and optimizing technical approaches. Notably, the authors acknowledge its potential social implications: Computer Use Agents developed based on this research may introduce risks such as data security breaches, privacy leaks, and unclear permission boundaries in real-world production scenarios.

A. Future Work on Online Agentic RL

Reinforcement Learning with Verifiable Rewards (RLVR) (Guo et al., 2025) has become a crucial framework for boosting the reliability, generalization, and performance of model. Building on this, our future work aims to explore online agentic reinforcement learning in GUI-based agent tasks. Constrained by time limitations, we have not yet conducted sufficient model training and comprehensive benchmark evaluations. Accordingly, the subsequent parts of this section will first conduct an in-depth analysis of the training-inference discrepancy issue, and then discuss the future research directions to advance this work.

Training-Inference Discrepancy in Trajectory-Level Training Algorithms such as GRPO (Shao et al., 2024) have been shown to be effective on a wide range of reasoning tasks. These algorithms collect a set of trajectories for a single query, calculate the advantage function within the trajectory group, and conduct training at the trajectory granularity. However, trajectory-level training will cause training-inference discrepancy in GUI tasks. During the rollout phase, GUI model does not retain all complete context information, but only preserves the complete information of recent steps (including screenshots, reasoning and actions), while earlier historical information is compressed into text-only semantic actions. If the trajectory of the final step is directly used for training, the model will not be able to learn the supervision signals of intermediate steps.

Step-Level Policy Optimization To address the training-inference discrepancy in trajectory-level training, we propose namely **Ste**p-Level **P**olicy **O**ptimization (**STEPO**), a simple yet effective policy optimization algorithm.

For a trajectory τ with length T, each step $t \in \{1, 2, \ldots, T\}$ contains K_t tokens. We denote the k-th token in step t as $x_{t,k}$ ($k \in \{1, 2, ..., K_t\}$), and the full token sequence of step t is represented by $x_t = (x_{t,1}, x_{t,2}, \ldots, x_{t,K_t})$. For the trajectory set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, the token at position k of step t in the i-th trajectory is denoted as $x_{i,t,k}$.

For each question q, similar to GRPO, STEPO samples a group G of trajectories $\{\tau_1, \tau_2, \ldots, \tau_n\}$ and calculates the advantages within the trajectory group:

$$\hat{A}_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{\text{std}(\{R_j\}_{j=1}^G)} \tag{3}$$

Where R_i represents the reward of the trajectory τ_i. Subsequently, the advantage value $\hat{A}_i$ corresponding to each trajectory τ_i is uniformly allocated to all steps contained in the trajectory, that is:

$$\hat{A}_{i,t} = \hat{A}_i / T_i, t \in \{1, 2, \ldots, T_i\}, \tag{4}$$

where T_i denotes the number of steps contained in the trajectory τ_i. All tokens within the same step share the corresponding advantage value $A_{i,t}$ of this step. On this basis, we conduct model training using all step-level samples. The optimization objective of the proposed algorithm can be demonstrated as:

$$\begin{aligned} &\mathcal{J}_{\text{STEPO}}(\theta) = \mathbb{E}[q \sim P(Q), \{\tau_i\}_{i=1}^G \sim \pi_{\theta_{old}}(\mathcal{T}|q)] \\ &\frac{1}{G}\sum_{i=1}^{G}\sum_{t=1}^{T_i}\frac{1}{K_t}\sum_{k=1}^{K_t}\{\min[r_{i,t,k}(\theta)\hat{A}_{i,t}, \text{clip}(r_{i,t,k}, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\hat{A}_{i,t}] - \beta\mathbb{D}_{KL}(\pi_\theta \| \pi_{\text{ref}})\}, \end{aligned} \tag{5}$$

where

$$r_{i,t,k}(\theta) = \frac{\pi_\theta(\tau_{i,t,k}|q, \tau_{i,t,<k})}{\pi_{\theta_{\text{old}}}(\tau_{i,t,k}|q, \tau_{i,t,<k})}, \tag{6}$$

denotes the importance sampling ratio. ϵ denotes the clipping parameter, $\mathbb{D}_{KL}$ denotes a KL penalty term and β controls the KL divergence regularization. By uniformly allocating the advantage value of a trajectory to all steps it comprises, this strategy achieves two core optimization effects: first, it drives high-advantage-value trajectories to complete tasks with fewer steps, thereby reducing redundant execution steps; second, it prompts low-advantage-value trajectories to expand the number of exploration steps, so as to improve the task completion rate. By the step-level policy optimization mechanism, STEPO can effectively circumvent the training-inference discrepancy issue.

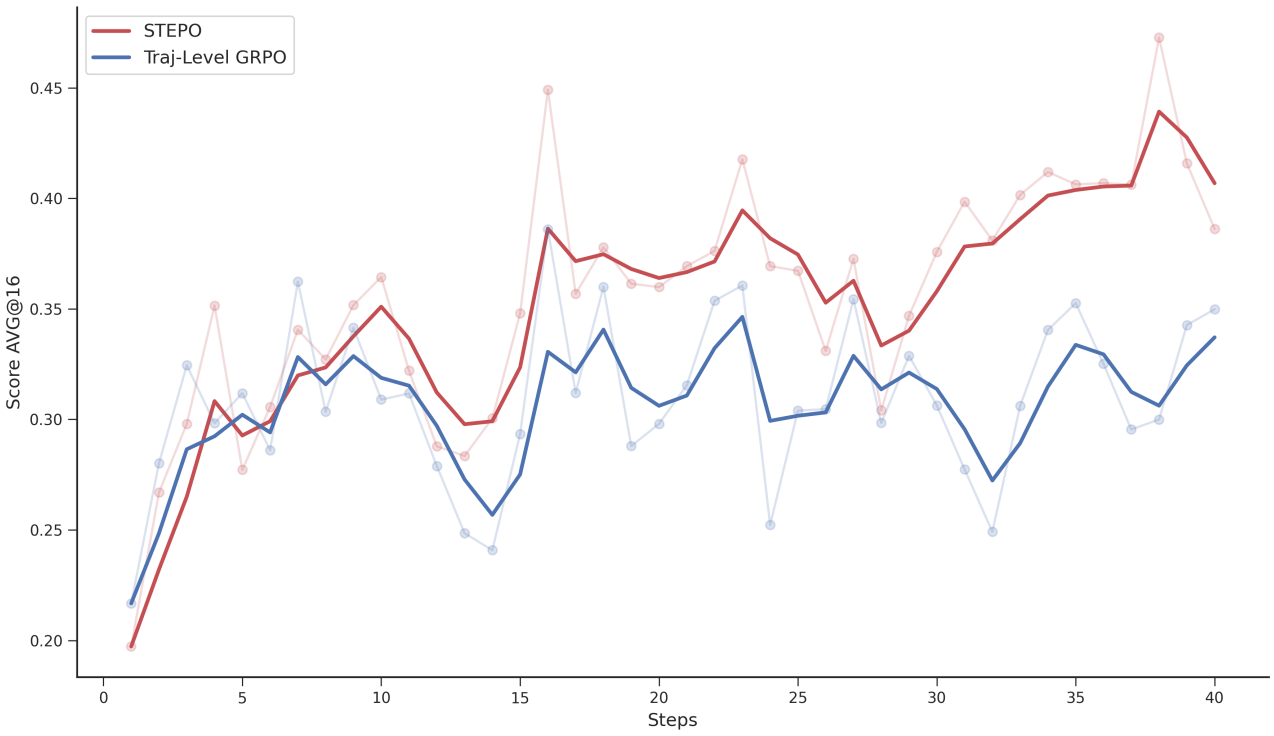


Figure 6. The values illustrated in this figure denote the 16-time average scores of the two methods, respectively.

Experiments and Analysis To clarify the impact of train-inference discrepancy and verify the effectiveness of STEPO, we conduct online RL training on the OpenCUA-32B model. As illustrated in the figure 6, the training performance of STEPO is significantly superior to that of GRPO trained with final trajectories, which fully confirms the effectiveness of STEPO.

However, STEPO suffers from the issue of high training cost, as the number of updates to the policy model multiplies significantly. Accordingly, we hypothesize that the requirements for step-level training may not be uniform across different training phases, and training only specific key steps might also achieve comparable performance to training all steps. In the future, we will explore directions such as scaling up online RL and developing more effective RL training recipes.

B. Detailed Discussions on Agentic Training

Drawing from over **a thousand individual experiments totaling more than 1 million accelerator hours**, we categorize our observations regarding the training dynamics of native computer use agents into five critical dimensions.

1. The Dual Nature of Experiences. Our analysis reveals that the signal-to-noise ratio varies fundamentally between success and failure trajectories, necessitating distinct processing strategies.

- *Success trajectories:* Trajectories generated by the model represent known knowledge characterized by low noise but limited information gain. While the final outcome is correct, step-level redundancy constitutes a major noise source. Without aggressive filtering of these inefficient steps, the model becomes fragile, leading to phenomena such as action aliasing (outputting conflicting actions for a single state) and cyclic repetition (endlessly clicking the same coordinates). Effective filtering is thus a prerequisite for multi-round rejection sampling fine-tuning.
- *Failure trajectories:* Conversely, failure trajectories are high-noise but high-information. They delineate the model's capability boundaries and contain corner cases that the current policy cannot handle. While raw failure data is too noisy for direct learning, identifying critical error steps allows for the construction of preference pairs. This transforms failed attempts into a high-value source for boundary alignment.

2. Foundational Constraints and Initialization. The initialization phase substantially influences the agent's potential performance.

- *Completeness of action space:* A comprehensive definition of the action space is a prerequisite. Missing high-efficiency operations (e.g., triple click, shift-based shortcuts) renders specific tasks, such as complex spreadsheet editing, effectively unsolvable. Post-hoc additions to the action space are inefficient compared to a correct initial definition.
- *Pattern-centric cold start:* The cold start phase should prioritize pattern diversity over data volume. We observed that a lightweight cold start is sufficient to establish a latent alignment—grounding the action space and stabilizing output formatting. A heavy cold start often yields high supervised metrics but creates a checkpoint that is harder to refine later. A lightweight initialization, followed by rigorous rejection sampling and preference optimization, consistently produces superior final performance.

3. Dynamics of Iterative Optimization. Computer use tasks are inherently long-horizon, often requiring dozens of interaction turns. Optimizing for this requires strict adherence to specific dynamic properties.

- *The on-policy imperative:* We emphasize the necessity of using strictly on-policy data during iterative learning. We hypothesize that off-policy data disrupts the principal direction of the optimization vector established during supervision. Once the model's weights diverge from the optimal manifold due to distribution shifts, recovering the correct optimization path is computationally prohibitive.
- *Termination asymmetry:* The distribution of the terminate action is the most critical control variable. We observed a distinct asymmetry: the model converges rapidly on failure recognition, whereas recognizing success requires a carefully calibrated density of positive samples. An excessive concentration of success signals leads to premature termination, while a deficit prevents the agent from stopping.
- *Self-correction and future potential:* To mitigate error accumulation in long-horizon tasks, we utilize preference optimization focused on state checking and reflection. By targeting steps where the agent fails to perceive errors, we enhance robustness. These improvements suggest that the logical evolution is a transition to online reinforcement learning, where advanced credit assignment mechanisms can further optimize performance in complex, multi-step environments.

4. Visualization-Driven Diagnosis and Iteration. We argue that achieving SOTA performance in long-horizon tasks requires more than algorithmic novelty; it demands a transparent debugging infrastructure. We developed a comprehensive suite of trajectory analysis and visualization tools that served as the "eyes" of our evolutionary cycle. These tools played a pivotal role in three critical phases:

- *Quality Assurance for Synthesis*: They allowed us to visualize synthesized samples alongside their ground-truth states, enabling rapid identification of "hallucinated validators" or executable logic errors in our Synthesis Engine before they polluted the training pool.
- *Cold-Start Data Construction*: By visually contrasting the trajectory characteristics of different foundation models, we identified superior reasoning patterns and action sequences. This guided the curation of our high-quality Cold Start dataset, ensuring the agent learned robust behavioral priors rather than noisy imitation.
- *Failure Analysis for Refinement*: Our Pass@k Differential Analysis tool aggregates successful and failed trajectories for the same query. This granular comparison helped us pinpoint specific failure modes—such as coordinate drift or reasoning-action misalignment—directly informing the design of our step-level policy optimization to rectify these specific weaknesses.

5. Environment Scaling. It is critical to distinguish the role of Pass@k in agentic tasks versus standard LLM benchmarks. In traditional text generation, the "environment" (the prompt) is static and deterministic; thus, Pass@k solely measures the diversity of the model's internal capacity. In contrast, GUI environments introduce inherent environmental stochasticity. Factors such as system latency, network fluctuations, and minor rendering variations mean that identical action sequences

can yield different state transitions. Consequently, in this context, Pass@k serves a dual purpose: it evaluates not only the model's generative diversity but also its robustness against environmental noise. We observe that even with deterministic sampling (temperature=0), success rates exhibit variance due to these system perturbations. This finding highlights a critical limitation of pure data scaling. To achieve human-level reliability, future research must prioritize environment scaling—expanding environmental diversity and modeling dynamic uncertainties to ensure robustness across real-world systems.

C. Technical Implementation of the Scalable Infrastructure

The transition from static data scaling to evolving experience learning necessitates a fundamental shift in infrastructure capabilities. Unlike passive training pipelines, our active learning paradigm requires a high-throughput gymnasium capable of generating continuous, diverse, and interactive feedback at a massive scale. To address the challenges of heterogeneity, high concurrency, and strict session isolation inherent in large-scale reinforcement learning, we developed a unified environment sandbox platform. This platform, illustrated in Figure 7, serves as the bedrock for EvoCUA, orchestrating hundreds of thousands of daily sandbox sessions and processing millions of interaction requests per day with industrial-grade stability.

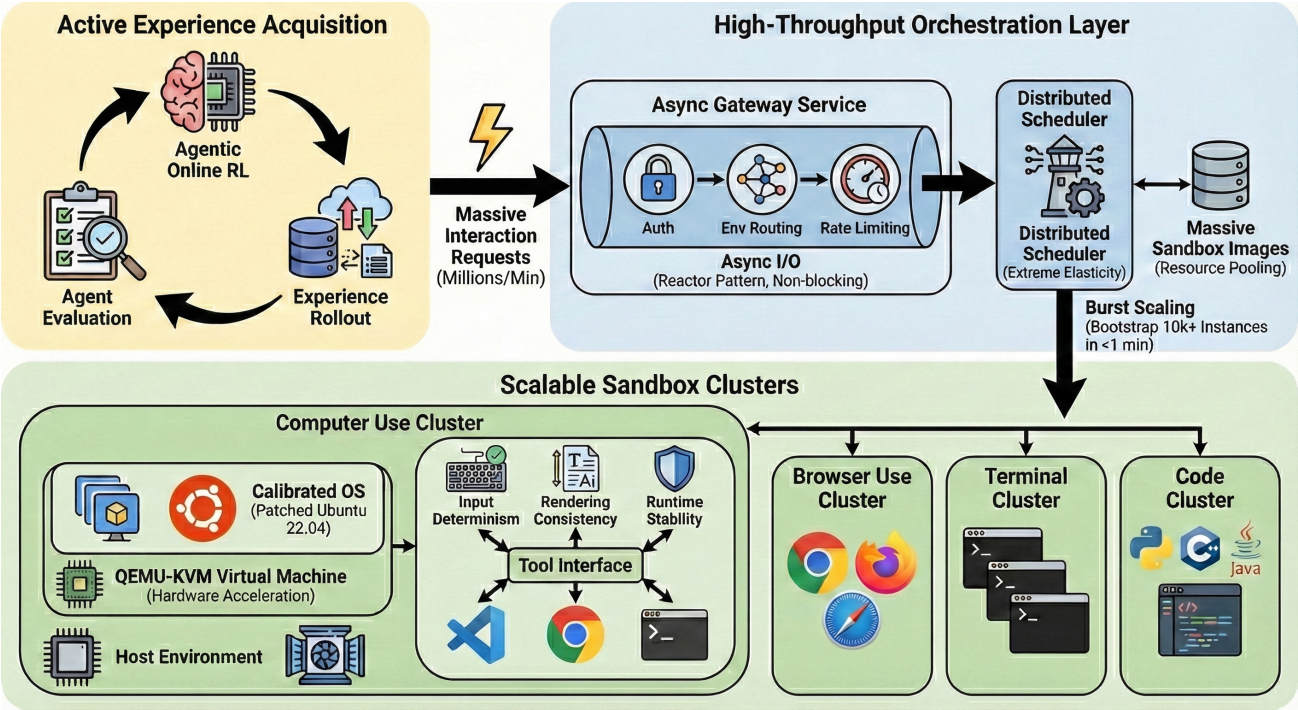


Figure 7. **Scalable Infrastructure.** The architecture orchestrates massive interaction requests from the online RL loop (top-left) through an asynchronous gateway and distributed scheduler (top-right). The bottom layer deploys parallel sandbox clusters, highlighting the **Computer Use Sandbox**, which utilizes QEMU-KVM virtualization and a calibrated OS to ensure input determinism, rendering consistency, and runtime stability for high-fidelity environments.

C.1. Architecture and Abstractions

To manage the complexity of diverse interaction tasks, the platform is architected around two core abstractions: *tools* and *clusters*.

Tools. A tool encapsulates the immutable definition of a simulation environment, including version-controlled system images and exposed interaction APIs. The platform currently supports hundreds of distinct environment types, ranging from generic benchmarks to specialized agentic environments. This design decouples environment iteration from experimentation, ensuring backward compatibility and reproducibility.

Clusters (Dynamic Scaling Units). A cluster represents the runtime instantiation of a tool and serves as the fundamental unit for environment scaling. By specifying tool types and configuring resource quotas, users can instantly provision customized environment services for distinct workloads. This abstraction allows the infrastructure to dynamically scale environment instances—from a handful of debugging sessions to tens of thousands of concurrent training nodes—without resource contention or cross-contamination.

C.2. High-Throughput Orchestration

The capability to support massive-scale exploration hinges on the efficiency of our microservices architecture, specifically designed to eliminate I/O bottlenecks and enable rapid environment scaling.

The infrastructure relies on an asynchronous gateway service based on the reactor pattern for non-blocking I/O. This service achieves a routing throughput at the scale of hundreds of thousands of requests per minute. By decoupling the control plane (lifecycle management) from the data plane (environment interaction), the gateway prevents long-running environment executions from blocking critical routing logic.

Complementing the gateway, the distributed scheduler is engineered for extreme elasticity, managing the lifecycle of massive sandbox images. Leveraging distributed sharding and resource pooling, the scheduler achieves high-efficiency node scheduling. More critically, it supports burst scaling capabilities, bootstrapping tens of thousands of sandbox instances within one minute. This rapid instantiation ensures that the environment scaling strictly matches the training demand of on-policy reinforcement learning, minimizing the latency between policy updates and experience collection. Ultimately, this resilient scheduling backbone enables the infrastructure to stably sustain over 100,000 concurrent sandboxes.

C.3. High-Fidelity Environment Instantiation

To support the rigorous requirements of computer use tasks, we implement a hybrid virtualization architecture that encapsulates QEMU-KVM virtual machines within Docker containers.

Hybrid virtualization. While Docker provides compatibility with our orchestration layer, the internal execution relies on QEMU with KVM hardware acceleration. We construct a customized QEMU launch sequence that explicitly disables non-essential peripherals while optimizing I/O performance. This nested design ensures strict kernel-level isolation—crucial for security when agents execute arbitrary code—while maintaining near-native performance for GUI rendering and I/O operations.

Deterministic environment calibration. We constructed a customized OS image based on Ubuntu 22.04 to address the gap between simulation and real-world deployment, implementing specific kernel and userspace patches:

- **Input determinism (HID patching):** Standard virtualization often suffers from key mapping collisions. We calibrated the human interface device mapping at the `xkb` kernel level. Specifically, we modified the `/usr/share/x11/xkb/symbols/pc` definitions to resolve symbolic collisions (e.g., the < vs > shift-state error in US layouts), ensuring that the agent's symbolic intent strictly matches the realized character input.

- **Rendering consistency:** To prevent layout shifts in office software that confuse visual agents, we injected a comprehensive suite of proprietary fonts directly into the system font cache (`fc-cache`). This guarantees that documents render identically to their native counterparts.

- **Runtime stability:** The image is hardened with system-level proxy configurations to resolve network instabilities and includes pre-installed dependencies like `xsel` and `qpdf` to eliminate common runtime errors during clipboard operations and PDF processing.

D. Unified Action Space

The following table details the unified native action space $\mathcal{A}$ implemented in EvoCUA. The agent interacts with the environment by invoking the `computer_use` function with a specific `action` and its corresponding arguments.

Table 5. Detailed Definition of the EvoCUA Native Action Space

Category	Action Primitive	Description	Required Arguments
Keyboard	`key`	Performs a key press and release sequence on the specified keys.	`keys` (array)
	`key_down`	Presses and **holds** the specified key(s). Used for stateful operations (e.g., holding Shift).	`keys` (array)
	`key_up`	Releases the specified key(s) in reverse order.	`keys` (array)
	`type`	Types a string of text on the keyboard.	`text` (string)
Mouse	`mouse_move`	Moves the cursor to the specified pixel coordinates.	`coordinate` (x, y)
	`left_click`	Clicks the left mouse button at the specified coordinates.	`coordinate` (x, y)
	`right_click`	Clicks the right mouse button at the specified coordinates.	`coordinate` (x, y)
	`middle_click`	Clicks the middle mouse button at the specified coordinates.	`coordinate` (x, y)
	`double_click`	Double-clicks the left mouse button at the specified coordinates.	`coordinate` (x, y)
	`triple_click`	Triple-clicks the left mouse button (useful for text selection).	`coordinate` (x, y)
	`left_click_drag`	Clicks and drags the cursor to the target coordinates.	`coordinate` (x, y)
	`scroll` / `hscroll`	Performs a vertical or horizontal scroll.	`pixels` (number)
Control	`wait`	Pauses execution for a specified duration to allow UI rendering.	`time` (number)
	`terminate`	Terminates the current task and reports the final status.	`status` (”success”—”failure”)

E. Cold Start: Hindsight Reasoning Generation

To construct high-quality data for the supervised cold-start phase, we transform raw physical interaction traces into training samples augmented with explicit cognitive chains. We employ a **Hindsight Reasoning Generation** strategy to achieve this. By treating the ground-truth execution path as known future information, we utilize a general model to retrospectively generate reasoning traces (z_t) that explain the observed actions, thereby establishing a causal alignment between cognition and execution.

The generation process is driven by a series of context-aware prompt templates that enforce the structural schemas defined in our **Thought Space ($\mathcal{Z}$)**. Depending on the execution phase, the generation logic adapts as follows:

1. Goal Clarification (z_0) At the initial step of a trajectory ($t = 0$), the reasoning generation focuses on resolving ambiguity and establishing a global plan.

- **Context:** The general model is provided with the user instruction, the initial screenshot, and the first executable code block.
- **Generation Logic:** We utilize a specific template that enforces a first-person perspective. The model must explicitly state the current environment state, clarify the task goal, and articulate a high-level plan (e.g., “*I need to open the browser to search for...*”) before justifying the specific action taken. This ensures that the subsequent physical execution is grounded in a clear intent.

2. Observation Consistency (z_{obs}) For intermediate steps, the objective is to maintain semantic consistency between the visual observation and the reasoning trace.

- **Context:** The model analyzes the transition from the previous state to the current state.
- **Generation Logic:** The prompt instructs the model to identify *“What changed”* in the environment and explain *“Why this action is needed”* to advance the workflow.
- **Semantic Abstraction:** To prevent overfitting to specific screen resolutions, the prompt explicitly constrains the generation to avoid mentioning raw pixel coordinates. Instead, the model is guided to describe target UI elements

semantically (e.g., "*Click on the 'File' menu*" rather than "*Click at (100, 200)*"), ensuring the reasoning remains robust to layout variations.

3. Reflection and Correction ($z_{reflect}$) For trajectories involving error recovery ("resume" traces), we implement a specialized **Reflection Mechanism**.

- **Context:** When processing a trajectory segment that recovers from a failure, the synthesis engine injects the specific `analysis_reason` (the root cause of the prior failure) into the prompt context.
- **Generation Logic:** The model is enforced to begin the thought trace with a dedicated header: "**Reflection:** ". It must retrospectively analyze the failure (e.g., "*Reflection: I realize that my previous attempt to click the icon failed because...*").
- **Self-Correction:** Following the reflection, the model must naturally transition to a corrected plan (e.g., "*Now I will try a different approach...*"), effectively internalizing the logic of self-correction into the training data.

4. Reasoning-Augmented Termination (z_T) To mitigate premature or delayed stopping, the termination action is conditioned on a rigorous visual verification process.

- **Context:** The generation is triggered at the final step of a trajectory.
- **Generation Logic:** The general model is required to assess the final screenshot against the initial instruction. It must generate a reasoning trace that provides visual evidence of task completion (or failure) before emitting the final `terminate` signal. This ensures that the agent's termination decision is grounded in logical verification rather than memorized trajectory lengths.

Algorithm 1 Hindsight Reasoning Generation

Input: Instruction g; Raw Trajectory $\tau = \{(o_t, a_t)\}_{t=0}^{T}$; Error Context c_{err} (optional)
Output: Reasoning Traces $\mathcal{Z} = \{z_t\}_{t=0}^{T}$

$\mathcal{Z} \leftarrow \emptyset$
$h_{prev} \leftarrow \emptyset$ *// Initialize interaction history*
for $t \leftarrow 0$ **to** T **do**
  $prompt \leftarrow$ NULL
  if $t = 0$ ▷ **Phase 1: Initialization then**
    if $c_{err} \neq$ NULL **then**
      // Trigger Reflection Mechanism for error recovery
      $prompt \leftarrow$ CONSTRUCTREFLECTPROMPT(g, c_{err}, o_0, a_0)
    else
      // Standard Goal Clarification
      $prompt \leftarrow$ CONSTRUCTGOALPROMPT(g, o_0, a_0)
    end if
  else if $t = T$ ▷ **Phase 3: Termination then**
    // Reasoning-Augmented Termination Verification
    $prompt \leftarrow$ CONSTRUCTTERMPROMPT(g, h_{prev}, o_T)
    else ▷ **Phase 2: Intermediate Execution**
      // Ensure Observation Consistency
    $prompt \leftarrow$ CONSTRUCTOBSPROMPT(g, h_{prev}, o_t, a_t)
  end if
  // Query General Model
  $z_t \leftarrow$ GeneralLLM$(prompt)$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{z_t\}$
  $h_{prev} \leftarrow h_{prev} \cup \{(z_t, a_t)\}$
end for
return $\mathcal{Z}$

F. Algorithm for DPO

In this section, we present the algorithmic implementation of Step-Level Direct Preference Optimization (DPO). This method focuses on two core processes: **Key Error Identification** and **Preference Pair Construction**. Algorithm 2 details how we identify Critical Forking Points from failure trajectories and construct paired data for both Action Correction and Reflection.

Algorithm 2 Step-Level DPO Pair Construction

Input: Target Trajectory $\mathcal{T}_{tgt}$ (Failure Case), Reference $\mathcal{T}_{ref}$ (Success Case), VLM $\mathcal{M}$
Output: DPO Dataset $\mathcal{D}_{dpo}$

$E \leftarrow$ AnalyzeErrorSteps$(\mathcal{T}_{tgt}, \mathcal{T}_{ref})$ ▷ **Step 1: Error Identification**
$\mathcal{D}_{dpo} \leftarrow \emptyset$
for each error step index t in E **do**
  $o_t, a_{rejected} \leftarrow$ GetStateAndAction$(\mathcal{T}_{tgt}, t)$ *// Extract failure context*
  $S_{aligned} \leftarrow$ None ▷ **Step 2: Critical Forking Point Discovery**
  for $k \leftarrow t - w$ **to** $t + w$ **do**
    $o_{ref}, a_{ref} \leftarrow$ GetStateAndAction$(\mathcal{T}_{ref}, k)$
    if CheckAlignment$(\mathcal{M}, o_t, a_{rejected}, a_{ref})$ **then**
      $S_{aligned} \leftarrow$ NormalizeCoords(a_{ref}, o_t)
      break
    end if
  end for
  if $S_{aligned} \neq$ None **then**
    $z_{enhanced} \leftarrow \mathcal{M}$.SynthesizeThought$(o_t, S_{aligned})$ ▷ **Step 3: Paradigm I (Correction)**
    $\tau_{chosen} \leftarrow (z_{enhanced}, S_{aligned})$
    $\mathcal{D}_{dpo}$.add$(\{$state : o_t, chosen : τ_{chosen}, rejected : $a_{rejected}\})$
    if $t + 1 <$ Length$(\mathcal{T}_{tgt})$ ▷ **Step 4: Paradigm II (Reflection) then**
      $o_{next}, a_{blind} \leftarrow$ GetStateAndAction$(\mathcal{T}_{tgt}, t + 1)$
      // Observe Error → Stop → Plan
      $z_{reflect} \leftarrow \mathcal{M}$.GenerateReflection$(o_{next}, a_{rejected}, S_{aligned})$
      $\tau_{chosen_ref} \leftarrow (z_{reflect}, S_{aligned})$
      $\mathcal{D}_{dpo}$.add$(\{$state : o_{next}, chosen : τ_{chosen_ref}, rejected : $a_{blind}\})$
    end if
  end if
end for
return $\mathcal{D}_{dpo}$

G. Extended Evaluation: Grounding and General Capabilities

In this section, we provide the full results and detailed analysis of EvoCUA on fine-grained GUI grounding and general multimodal benchmarks.

We assess EvoCUA's performance across two critical dimensions: fine-grained GUI grounding (ScreenSpot-v2 (Wu et al., 2024), ScreenSpot-Pro (Li et al., 2025), OSWorld-G (Xie et al., 2025)) and general multimodal robustness (MMMU (Yue et al., 2024), MMMU-Pro (Yue et al., 2025), MathVista (Lu et al., 2024), MMStar (Chen et al., 2024), OCRBench (Liu et al., 2024)). Table 6 summarizes the results across different model scales and backbones.

Analysis. We observe distinct behaviors depending on the base model used. For the OpenCUA-72B backbone, our post-training strategy maintains performance parity or yields slight improvements across both grounding and general benchmarks (e.g., preserving MMMU scores while improving OSWorld-G). This stability confirms that our training method effectively preserves the base model's knowledge when the data distribution is aligned.

Conversely, the EvoCUA-32B variant exhibits performance decline in specific metrics, notably on ScreenSpot-Pro and MMMU, compared to the Qwen3-VL-32B-Thinking baseline. We attribute this performance drop primarily to discrepancies in data distribution and patterns. Due to time constraints, the general dataset used for fine-tuning EvoCUA was directly

Table 6. Performance comparison on the offline grounding and general benchmarks. Values marked with * are sourced from other public reports.

Model	GUI Grounding			General Multimodal Capabilities				
	ScreenSpot v2	**ScreenSpot Pro**	**OSWorld-G**	**MMMU**	**MMMU-Pro**	**MathVista**	**MMStar**	**OCRBench**
OpenCUA-72B	92.90*	60.80*	66.95	60.67	43.04	70.90	66.47	83.8
Qwen3-VL-8B-Thinking	90.09	46.40*	56.70*	74.10*	60.40*	81.40*	75.30*	81.9*
Qwen3-VL-32B-Thinking	91.11	57.10*	64.00*	78.10*	68.10*	85.90*	79.40*	85.5*
EvoCUA-OpenCUA-72B	93.47	63.24	67.65	59.22	46.51	69.40	67.80	84.05
EvoCUA-8B	85.21	45.39	55.08	62.11	53.30	75.80	69.07	80.30
EvoCUA-32B	90.40	49.76	63.86	68.11	59.16	80.40	73.20	85.35

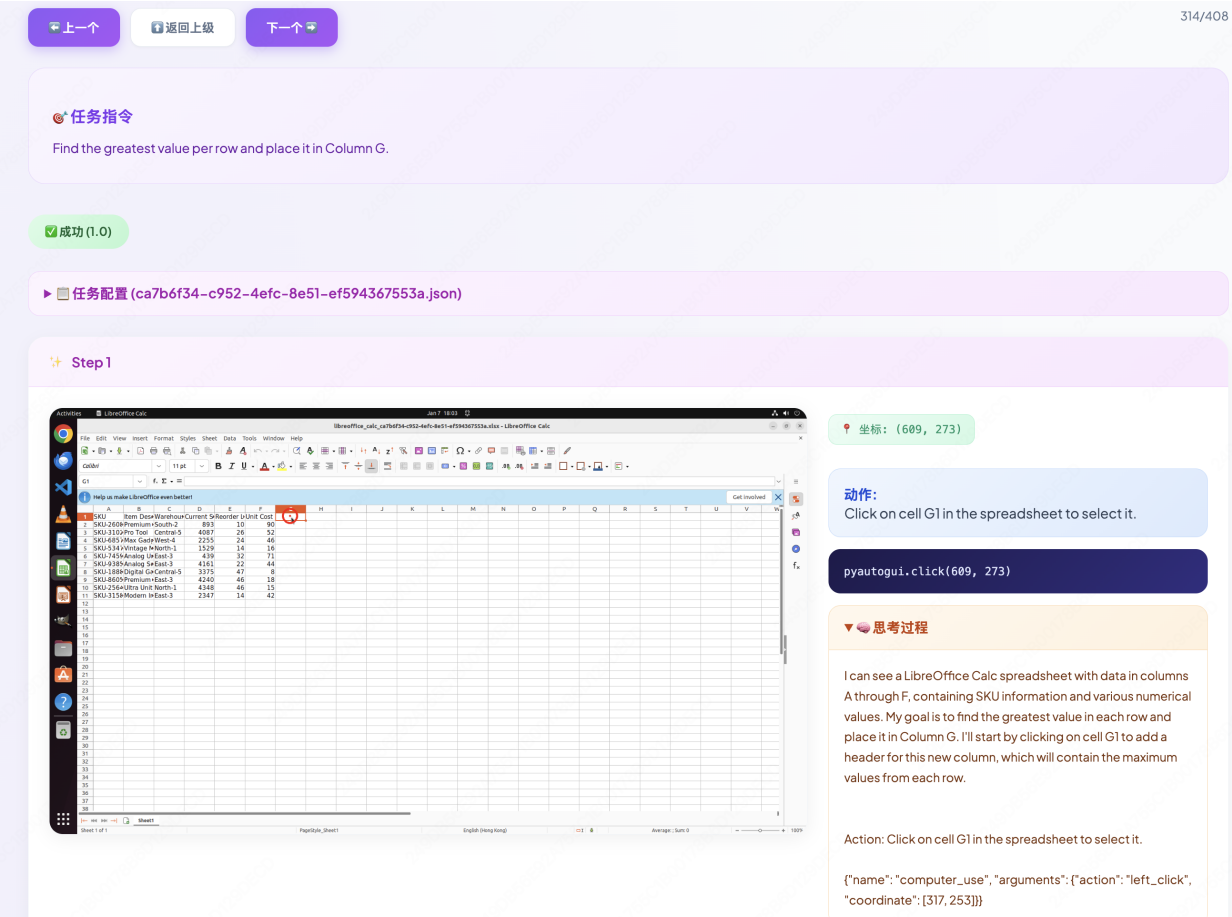
adopted from OpenCUA-72B variants experiments. However, this dataset is non-thinking, creating a significant mismatch with the thinking-based distribution of the Qwen3-VL-32B-Thinking model. We further analyzed the output lengths of Qwen3-VL-32B-Thinking and EvoCUA on general benchmarks. The results reveal a significant reduction in EvoCUA's token count compared to Qwen3-VL-32B-Thinking (2,514 vs 3,620), accompanied by a shift in output style.

Conclusion. The consistent performance on the OpenCUA backbone validates the effectiveness of our training strategy. The performance decline observed in the Qwen3-VL-Thinking-based variants is primarily attributed to a shift in general data distribution and patterns. Future updates of the EvoCUA models will incorporate an upgraded thinking-based general dataset. This alignment is expected to resolve the current discrepancy and further improve the model generalization performance.

H. Trajectory Analysis and Visualization

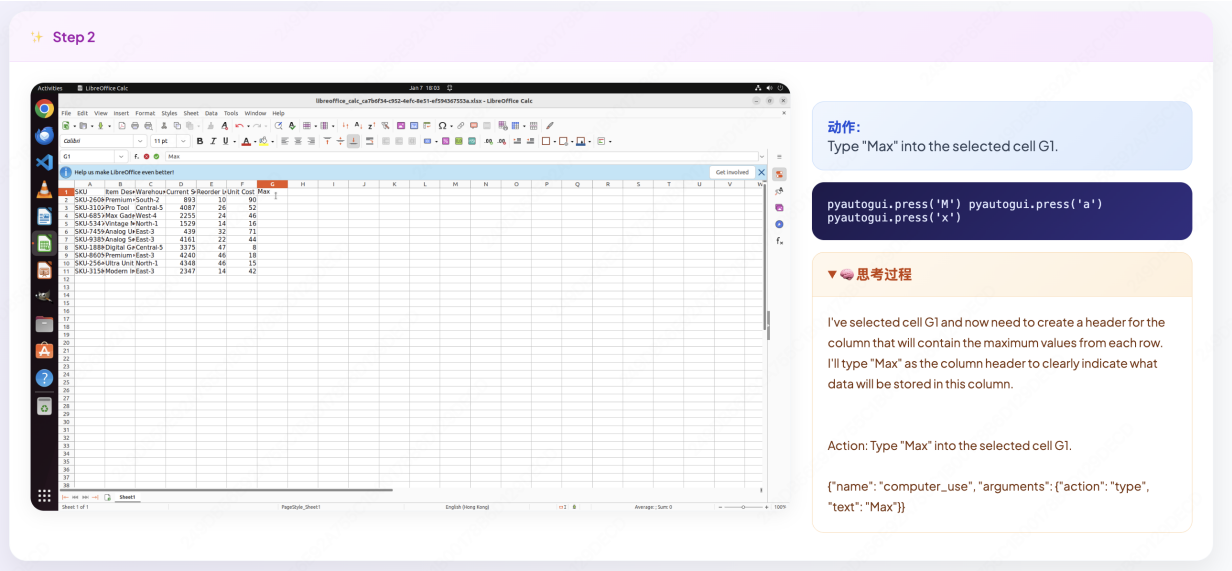
To enable granular diagnosis of agent behaviors and strictly validate the quality of our **synthetically generated experience**, we developed the **EvoCUA Trajectory Inspector**. This visualization system allows us to examine the frame-by-frame alignment between the agent's visual observation (o_t), internal reasoning trace (z_t), and the executable code action (a_t).

We illustrate the utility of this system using a representative synthetic task from the spreadsheet domain: *"Find the greatest value per row and place it in Column G."* This long-horizon task serves as a rigorous testbed for validating the logical consistency of our synthesis engine. Figure 8 presents the visualization of these key timestamps.



(a) Step 1: Goal Clarification ($t = 1$)**.** The inspector visualizes the initial state. The reasoning panel displays the agent's explicit paraphrasing of the instruction ("Find the greatest value... place it in Column G"), validating the goal grounding mechanism in our synthetic data.

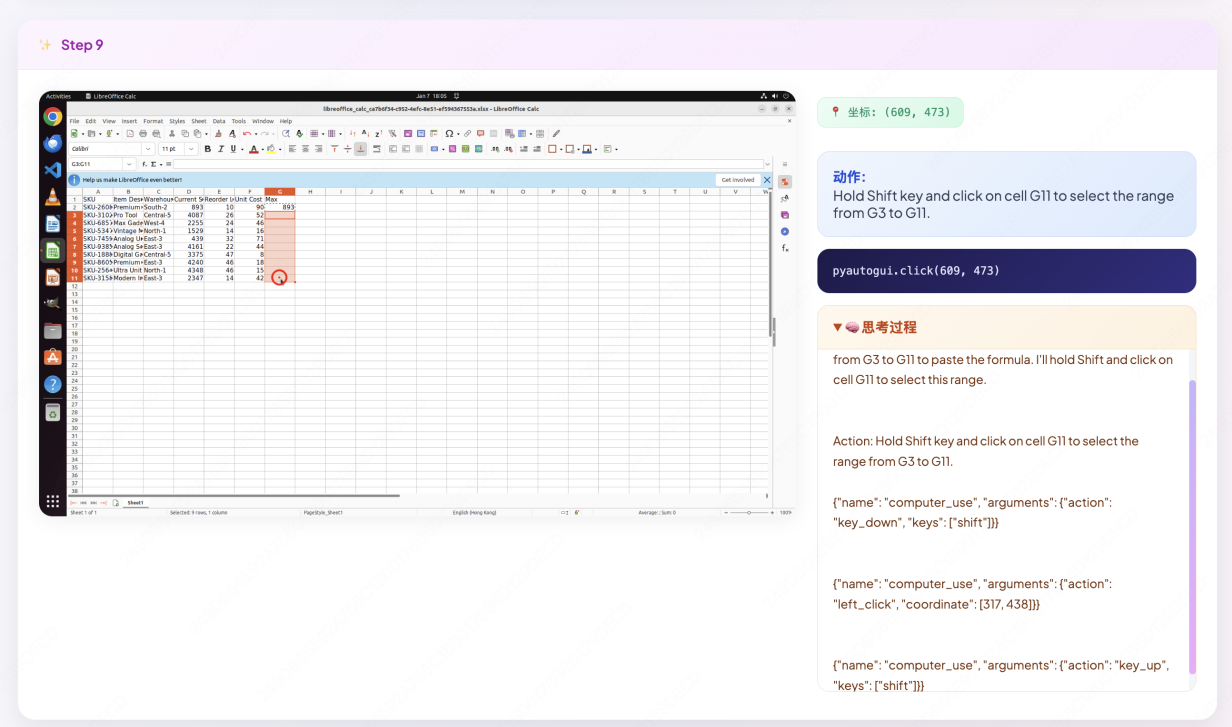
↓



(b) Step 2: Text Entry ($t = 1$)**.** The system captures the transition from planning to execution. The agent's reasoning ("Type 'Max'...") is perfectly aligned with the generated atomic action sequence (`press('M')`, `press('a')`, `press('x')`).

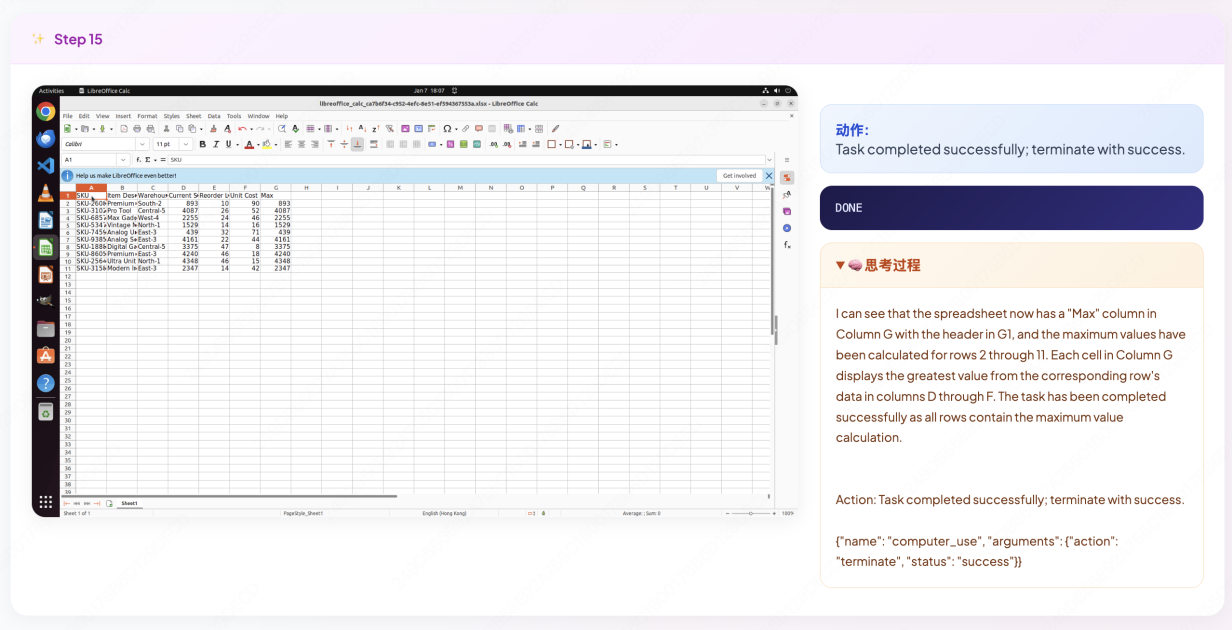
Figure 8. **Visualization of Synthesized Trajectory (Part I).** The EvoCUA Trajectory Inspector validates the logical consistency of synthetic training data: (a) Clear alignment between user instruction and agent planning; (b) Precise correspondence between reasoning and atomic keyboard actions.

⋮



(c) Step 9: Stateful Interaction ($t = 9$)**.** This view validates the *Unified Action Space*. The synthetic ground truth requires a stateful operation (Shift-Select). The inspector confirms the agent correctly executes the `key_down: shift` → `click` → `key_up: shift` sequence.

⋮



(d) Step 15: Verified Termination ($t = 15$)**.** The final frame validates the *Reasoning-Augmented Termination* schema. The tool highlights that the agent generates visual evidence ("I can see... Max column... calculated") to justify the successful termination status.

Figure 8. **Visualization of Synthesized Trajectory (Part II) (Cont.).** (c) Validation of complex stateful primitives (Shift+Click) essential for GUI manipulation. (d) Validation of the termination logic to ensure task completeness.