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]

[1]Meituan [2]Fudan University
[3]Tongji University
[4]The Hong Kong University of Science and Technology

[*]Equal contribution. [†]Corresponding authors.

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 OSWorld-Verified, competitive with state-of-the-art open-weights models in the 30B parameter class. It significantly outperforms the larger open-weights model, OpenCUA-72B (45.0%), and surpasses leading closed-weights models such as UI-TARS-2 (53.1%), offering a highly parameter-efficient alternative to massive closed-weights models. 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. Code and model checkpoints are available at `https://github.com/meituan/EvoCUA`.

1 Introduction

Building generalist agents capable of mastering computer use through complex visual perception and long-horizon workflow execution represents a pivotal milestone in artificial general intelligence. While recent native vision-language models (VLMs) have integrated perception and action into end-to-end architectures [1, 2], and foundational efforts such as UI-TARS-2 [3] and OpenCUA [4] have established strong baselines, further progress is increasingly constrained by the diminishing returns of scaling with static, non-interactive datasets, which fail to capture the causal feedback inherent in real-world computer use.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

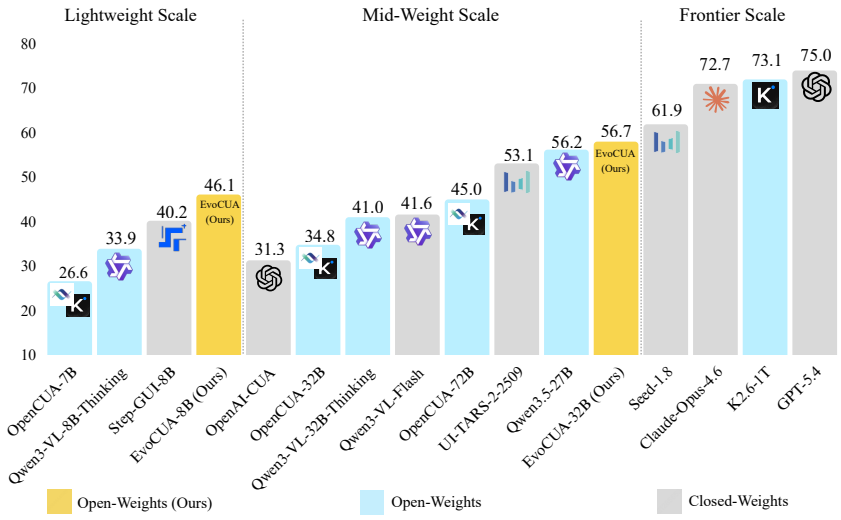


Figure 1: **Performance comparison on the OSWorld-Verified benchmark.** Models are categorized by parameter scale. EvoCUA achieves a 56.7% success rate, competitive with state-of-the-art open-weights models in the 30B parameter class.

Overcoming this limitation necessitates a paradigm shift from data scaling via static traces to experience scaling via massive interactive rollouts, which provide richer supervisory signals encompassing environmental feedback and insights from both success and failure. However, transforming raw interaction into a self-improving learning loop presents three primary challenges: 1) *Verifiable data synthesis*: synthesizing textual queries alone often leads to hallucinations where the agent generates plausible plans for infeasible tasks, requiring a framework that strictly grounds generation in solvable states with verifiable rewards. 2) *Scalable interaction infrastructure*: high-throughput experience production demands a unified system integrating massive environment simulation with reinforcement learning to support continuous, asynchronous interaction. 3) *Efficient training recipe*: given a vast interaction space, unconstrained exploration is computationally prohibitive; effective learning requires an on-policy approach that consolidates mastered routines while focusing on boundary tasks where the agent oscillates between success and failure.

To address these issues, we introduce EvoCUA, a native computer use agent driven by an evolving paradigm. As illustrated in Figure 2, EvoCUA unifies verifiable synthesis, high-throughput infrastructure, and evolutionary optimization into 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 reducing reward ambiguity through experience acquisition.

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

- **Parameter-Efficient Performance.** EvoCUA is competitive with state-of-the-art models in the 30B parameter class on OSWorld with a 56.7% success rate, surpassing both the larger OpenCUA-72B (45.0%) [4] and leading closed-weights models like UI-TARS-2 (53.1%) [3].

Evaluation code and model checkpoints are available at `https://github.com/meituan/EvoCUA`.

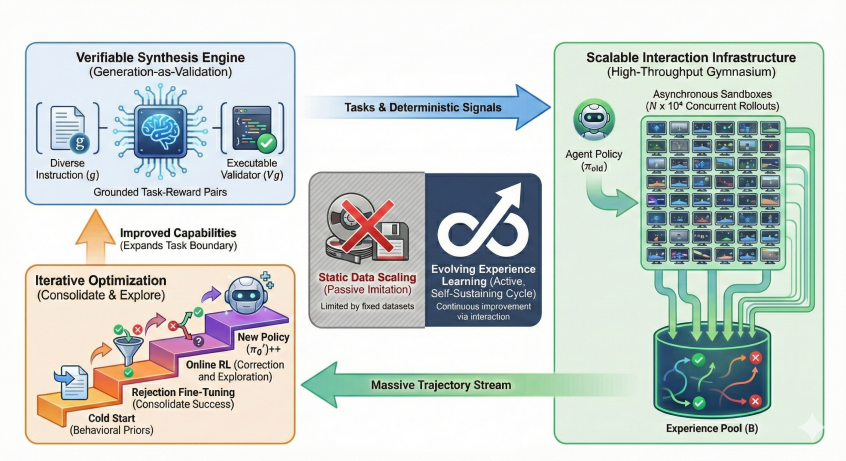


Figure 2: EvoCUA unifies three core modules: the **Verifiable Synthesis Engine** (top left); the **Scalable Interaction Infrastructure** (right); and **Iterative Optimization** (bottom left).

2 Preliminaries

Problem Formulation. We model computer use as a POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}_{syn})$. At each step t, the agent perceives a visual observation $o_t = \text{Render}(s_t)$ of the environment state s_t, and maintains an interaction history $h_t = \{g, o_0, z_0, a_0, \ldots, o_{t-1}, z_{t-1}, a_{t-1}\}$, where g is the task instruction and z_t denotes the chain-of-thought reasoning. The policy $\pi_\theta(z_t, a_t \mid h_t, o_t)$ follows a Reason-then-Act paradigm over a unified action space $\mathcal{A} = \mathcal{A}_{\text{mouse}} \cup \mathcal{A}_{\text{keyboard}} \cup \mathcal{A}_{\text{control}}$. Supervision uses a verifiable binary reward $\mathcal{R}_{syn}(s_T;\, g) \triangleq \mathbb{I}[V_g(s_T) = \text{True}]$, where V_g is a task-specific verification function that checks whether the final state s_T satisfies the goal g. The training objective is $J(\theta) = \mathbb{E}_{g\sim\mathcal{T}_{syn},\tau\sim\pi_\theta}[\mathcal{R}_{syn}(s_T;\, g)]$, with tasks sampled from a synthetic task distribution $\mathcal{T}_{syn}$. A complete formulation is provided in Appendix A.

Overview. Optimizing $J(\theta)$ requires addressing three challenges: (1) constructing a grounded task distribution $\mathcal{T}_{syn}$ with deterministic validators V_g (Section 3); (2) collecting on-policy trajectories τ at scale (Section 3); and (3) optimizing π_θ over the vast interaction space (Section 4).

3 Verifiable Synthesis Infrastructure

We introduce a verifiable synthesis infrastructure that mitigates reward hacking and imprecise training signals via a "generation-as-validation" paradigm: for any synthesized instruction g, we co-generate a deterministic, executable validator V_g such that the reward $\mathcal{R}_{syn}(s_T; g)$ derives from strict verification of the final environment state. The 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 Figure 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. Building on the principle that atomic capabilities are inherently transferable and form complex tasks through composition, 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 this hierarchical domain taxonomy, we synthesize a wide range of task scenarios featuring diverse user

personas [5], ranging 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. 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. For *non-parametric injection*, we inject public internet data including images, audio, and complex slides, forcing 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 [6]. 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. In parallel, the **Validator stream (V_g)** generates the ground truth and the corresponding executable evaluator code, which defines the precise success conditions for the task [7].

To guarantee executability, we enforce a **closed-loop feedback mechanism**. The generated code is immediately executed in a real sandbox environment, and the execution results are fed back to the model to evaluate the quality of both the generated files and the evaluator. This process iterates for multiple rounds until 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. We implement a multi-stage filtering and tri-fold decontamination protocol to ensure data integrity and prevent leakage. This pipeline successfully scales our verifiable dataset to tens of thousands of instances (see Appendix D for detailed procedures).

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. For **elastic management**, the environment employs an asynchronous gateway based on the reactor pattern to decouple control from execution; managed by a distributed scheduler, this architecture can bootstrap $10{,}000$ instances per minute and sustain over $100{,}000$ concurrent sandboxes. For **deterministic calibration**, we utilize a hybrid virtualization approach (QEMU-KVM within Docker) to balance strict isolation with near-native GUI performance, and ensure validation stability across millions of runs through kernel-level patches for input determinism and unified font caching for rendering consistency. A comprehensive discussion of the technical stack is provided in Appendix F.

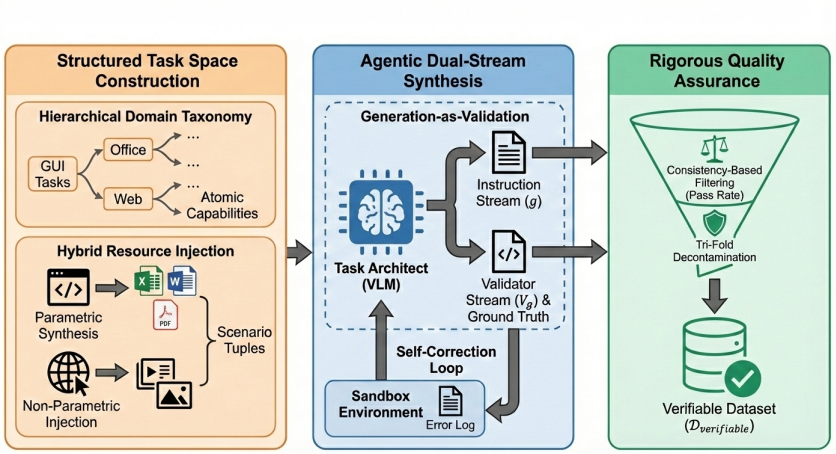


Figure 3: Architecture of the Verifiable Synthesis Engine, comprising **Structured Task Space**, **Agentic Dual-Stream Synthesis** (co-generating instructions g and validators V_g), and **Quality Assurance**.

4 Evolving Learning from Experience

To bridge atomic imitation and generalist problem-solving, we propose an evolving paradigm comprising three stages: cold-start to establish behavioral priors, rejection sampling fine-tuning to consolidate successful experiences, and offline preference optimization to learn from failures. The complete policy-conditioned training loop is summarized in Appendix I.

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 construct $\mathcal{A} = \mathcal{A}_{\text{mouse}} \cup \mathcal{A}_{\text{keyboard}} \cup \mathcal{A}_{\text{control}}$ via **Semantic Action Mapping** (full schema in Appendix G). For *Physical Interaction* ($\mathcal{A}_{\text{mouse}} \cup \mathcal{A}_{\text{keyboard}}$), we encompass coordinate-based mouse events and keyboard inputs; crucially, we decouple key presses into `key_down` and `key_up` events, enabling stateful operations like holding modifiers (e.g., `Shift`) for multi-selection. For *Control Primitives* ($\mathcal{A}_{\text{control}}$), we introduce meta-actions distinct from physical I/O: `wait` handles asynchronous UI rendering, while `terminate` formally signals task completion.

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

- *Goal Clarification (z_0)*: At $t = 0$, the agent paraphrases the user objective to clarify ambiguous instructions and ground subsequent planning.
- *Observation Consistency (z_{obs})*: The reasoning trace summarizes key visual elements, enforcing strict semantic consistency with the observed state to minimize hallucinations.
- *Self-Verification (z_{check})*: Before issuing termination, the agent executes auxiliary steps (e.g., checking file status) to visually confirm alignment with the user instruction.
- *Reflection and Correction ($z_{reflect}$)*: From failed rollouts, we restore the environment to the pre-error state (filtered for state consistency under sandbox non-determinism), then induce self-correction via high-temperature sampling to generate remedial paths.
- *Reasoning-Augmented Termination (z_T)*: The `terminate` action is conditioned on a reasoning trace that synthesizes visual evidence justifying completion, preventing overfitting to the termination label.

Based on these definitions, we synthesize the prior dataset $\mathcal{D}_{\text{prior}}$ via hindsight reasoning generation: treating ground-truth execution paths as known future information, we retrospectively generate reasoning traces z_t that explain the observed actions, augmenting physical trajectories with coherent cognitive chains (see Appendix H for synthesis and training details). After cold-start training, the agent masters atomic capabilities (Appendix N); however, due to fragility in complex boundary cases, our next stage focuses on internalizing scalable, high-quality experiences.

4.2 Rejection Sampling Fine-Tuning

Rejection Sampling Fine-Tuning (RFT) [8] consolidates the agent's task-solving ability by learning exclusively from successful executions, via two components: efficient trajectory generation under dynamic compute, and denoising for high 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, $\mathrm{SR}(k_i)$ 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 trajectories and mask redundant steps. This is especially important for infeasible tasks; for these, we remove all intermediate actions and keep only the reasoning trace and final `terminate=failure` action. This process refines raw data into high-quality supervision, aggregated into the experience pool $\mathcal{B}$.

This pipeline scales the experience pool $\mathcal{B}$ to tens of thousands of trajectories, interleaved with general-purpose multimodal data to prevent catastrophic forgetting.

4.3 Step-Level Offline Preference Optimization

While RFT consolidates successful behaviors, it does not explicitly correct failures. Since trajectory-level preference optimization suffers from state misalignment in long-horizon tasks, we propose a Step-Level Direct Preference Optimization strategy [9] targeting *Critical Forking Points* (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 environmental states remaining functionally equivalent. This isolates the response $(z_l, a_l) = (z_{t^*}^-, a_{t^*}^-)$ that caused the agent to leave the optimal solution manifold, from which we construct preference pairs along two paradigms.

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.

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:

$$\mathcal{J}(\theta) = -\mathbb{E}_{\mathcal{D}} \log \sigma\Big(\beta \log \tfrac{\pi_\theta(z_w, a_w|\cdot)}{\pi_{\text{ref}}(z_w, a_w|\cdot)} - \beta \log \tfrac{\pi_\theta(z_l, a_l|\cdot)}{\pi_{\text{ref}}(z_l, a_l|\cdot)}\Big) \tag{2}$$

where $\mathcal{D} = \{(h_t, o_t, (z, a)_w, (z, a)_l)\}$ and $\cdot$ abbreviates the conditioning context (h_t, o_t).

By iteratively updating the policy with structured preferences, EvoCUA continuously expands its capability boundary, 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) through large-scale rejection sampling with step-level denoising and preference learning from errors. Training-data statistics are summarized in Appendix C. We validate our approach across varying scales by post-training on the **Qwen3-VL-Thinking** [1] (8B, 32B) and **OpenCUA** [4] (7B, 32B, 72B) foundation models.

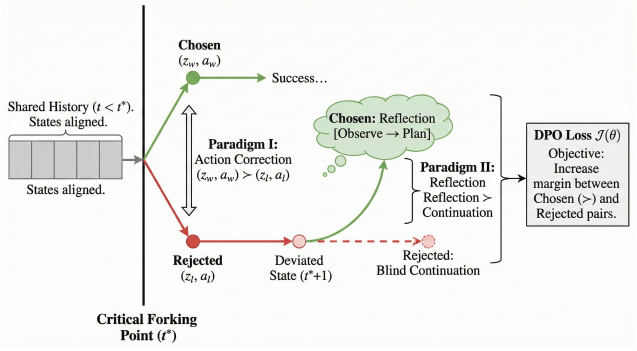


Figure 4: **Dual-Paradigm DPO.** At forking point t^*, **Paradigm I** corrects the action (a_w over a_l); **Paradigm II** optimizes step $t^* + 1$ via reflective reasoning over blind continuation.

5.2 Main Results

We evaluate EvoCUA on the **OSWorld** benchmark, a representative testbed for open-ended computer use tasks. As summarized in Table 1, EvoCUA-32B achieves a **56.7%** success rate within a 50-step budget, competitive with state-of-the-art open-weights models in the 30B parameter class. It delivers a **+15.7%** gain over its Qwen3-VL backbone (41.0%) and substantially outperforms the larger OpenCUA-72B (45.0%), validating the efficacy of our evolving learning paradigm. Per-domain results are provided in Appendix K.

While concurrent works such as the open-weights K2.6 (73.1%) and proprietary GPT-5.4 (75.0%) push absolute boundaries via massive scaling, EvoCUA-32B offers a superior balance of performance and accessibility: with significantly fewer parameters, it surpasses UI-TARS-2 (53.1%) and rivals Claude-Sonnet-4.5 (58.1%), serving as a parameter-efficient alternative for computer-use tasks.

We further compare performance on the Qwen3-VL-8B backbone: EvoCUA-8B achieves **46.1%**, outperforming Step-GUI-8B (40.2%) by **+5.9%** and surpassing specialized 72B baselines, demonstrating exceptional parameter efficiency.

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

6 Cross-Environment Generalization

To evaluate zero-shot cross-OS generalization, we test EvoCUA on **WindowsAgentArena** (WAA) [20], providing a rigorous out-of-distribution test given our Linux-only training pipeline. As shown in Table 2, EvoCUA-32B achieves **56.48%** in 50 steps, surpassing its Qwen3-VL-32B-Thinking base (42.9%) by **+13.58%** and the leading UI-TARS-2 (50.6%) by nearly 6 points. These results confirm that the action schemas and reasoning priors acquired through our evolutionary paradigm generalize across operating systems without overfitting to a specific platform.

6.1 Ablation Study

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

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 offline DPO. As shown in Table 3, each stage of the evolutionary cycle yields significant monotonic improvements. Controlled experiments show that the unified action space contributes a **+4.84%** improvement, and 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 then increases performance by **+3.13%** through successful

Table 1: **Comparison on OSWorld-Verified. EvoCUA-32B** achieves performance competitive with state-of-the-art models in the 30B-parameter class, significantly outperforming larger baselines.

Model	Type	Max Steps	Success Rate (Pass@1)
Closed-Weights Models			
OpenAI CUA [10]	Specialized	50	31.3%
Step-GUI-8B [11]	Specialized	100	40.2%
Qwen3-VL-Flash [1]	General	100	41.6%
UI-TARS-2-2509 [3]	General	100	53.1%
Claude-Sonnet-4.5 [12]	General	50	58.1%
Seed-1.8 [2]	General	100	61.9%
Claude-Sonnet-4.5 [12]	General	100	62.9%
Claude Opus 4.6 [13]	General	100	72.7%
GPT-5.4 [14]	General	100	**75.0%**
Open-Weights Models			
Qwen2.5-VL-32B-Instruct [15]	General	100	5.9%
Qwen2.5-VL-72B-Instruct [15]	General	100	8.8%
UI-TARS-72B-DPO [16]	Specialized	50	24.6%
OpenCUA-7B [4]	Specialized	100	26.6%
UI-TARS-1.5-7B [16]	Specialized	100	27.5%
Qwen3-VL-8B-Thinking [1]	General	100	33.9%
OpenCUA-32B [4]	Specialized	100	34.8%
GUI-Owl-7B-Desktop-RL [17]	Specialized	15	34.9%
Qwen3-VL-32B-Thinking [1]	General	100	41.0%
OpenCUA-72B [4]	Specialized	100	45.0%
EvoCUA-8B (Ours)	**General**	**50**	**46.1%**
Qwen3.5-27B [18]	General	100	56.2%
EvoCUA-32B (Ours)	**General**	**50**	**56.7%**
K2.6 [19]	General	100	**73.1%**

Table 2: Zero-shot performance on WindowsAgentArena (Linux→Windows transfer).

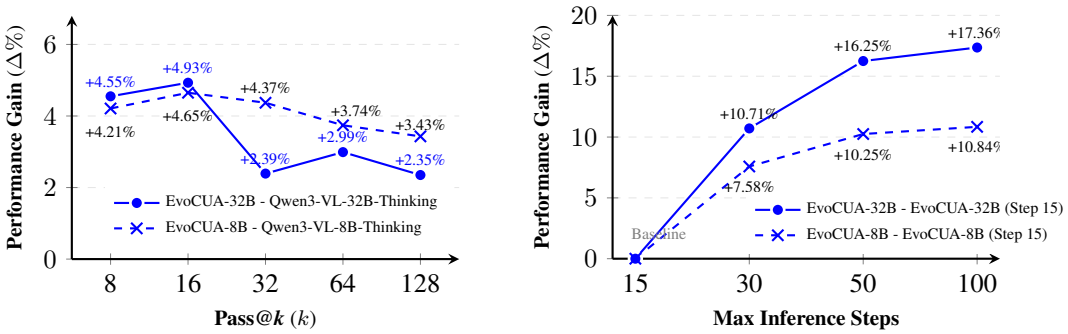
Model	Success (%)
Qwen3-VL-32B-Instruct [1]	30.90
Qwen3-VL-32B-Thinking [1]	42.90
UI-TARS-2 [3]	50.60
EvoCUA-32B (Ours)	**56.48**

experience consolidation, while 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.

Generalizability on OpenCUA-72B. To verify the universality of our approach, we apply the same paradigm to the larger OpenCUA-72B model (Table 3). The results echo our findings on Qwen3-VL, with DPO (+3.02%) and RFT (+3.69%) providing strong contributions. Notably, OpenCUA-72B adopts the standard pyautogui format, which natively supports stateful operations (e.g., shift+click) and exhibits no obvious functional deficiencies in its action space.

Table 3: Ablation across model scales. Δ denotes absolute gain over 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%**



(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. (b) Performance scaling with increased maximum inference steps.

6.2 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. As shown in Figure 5a, EvoCUA maintains a consistent performance lead over the Qwen3-VL-Thinking baseline across all Pass@k metrics, with the 32B variant peaking at $k = 16$. This persistent gap demonstrates that our training strategy improves single-shot reliability while elevating the absolute capability ceiling, as evidenced by +4.93% gain at $k = 16$.

Scaling with Max Steps. As shown in Figure 5b, performance steadily improves with increased step budgets, with the 32B model gaining over 16% from 15 to 50 steps. Beyond 50 steps, the rate of improvement moderates, primarily reflecting 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). 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, reaches **+6.79 pp** with 226k samples; and *Round 3* achieves **+8.12 pp** by training the OpenCUA-72B base from scratch on 1M aggregated samples.

Our analysis reveals a trade-off between data scale and signal-to-noise ratio (SNR): as models scale, noise tolerance drops, bottlenecking iterative methods. Continued scaling is feasible only when data quality, on-policy alignment, and SNR are jointly prioritized. Given the extensive scope of our experiments, we distill several **key insights** in Appendix E.

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

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

7 Related Work

Foundation VLMs and Computer Use Capabilities. The landscape of Large Visual Language Models (VLMs) has rapidly evolved to support complex agentic tasks. While Claude 4.5 Sonnet [12] and Seed 1.8 [2] established the standard, recent concurrent works—including ultra-large-scale models like GPT-5.4 [14] (75.0%), Opus 4.6 [13] (72.7%), and K2.6 [19] (73.1%)—have pushed boundaries via extensive scaling. To offer a highly parameter-efficient alternative, EvoCUA builds upon Qwen3-VL [1]. Through a specialized evolutionary post-training curriculum, it transcends pre-training limitations to deliver robust agentic capabilities at the 30B scale.

Generalist CUA Agents and Benchmarks. OSWorld [21] serves as the primary online evaluation testbed. OpenCUA [4] provides a foundation with AgentNet and a unified action schema; UI-TARS-2 [3] and Step-GUI [11] leverage multi-turn RL and step-wise reasoning; UltraCUA [7] proposes programmatic validator generation. These works address data, infrastructure, or optimization in isolation. EvoCUA instead contributes their *closed-loop co-design*: policy-aware synthesis calibrated to the capability boundary, on-policy sandbox rollouts as an integral training component, and failure-driven feedback guiding subsequent synthesis into a self-sustaining cycle.

Visual Grounding and Action Execution. Precise GUI grounding is essential for action execution. Following early works like Aguvis [22], recent models such as ShowUI [23] and UGround [24] 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 is shifting from Behavior Cloning toward Reinforcement Learning (RL). Beyond standard PPO [25], the Reinforcement Learning with Verifiable Rewards (RLVR) paradigm—pioneered by DeepSeek-R1 [26]—enables verifying reasoning without dense supervision. Recent advances include GIGPO [27] and reward-free "Early Experience" [28]. EvoCUA extends this by addressing data scarcity through a verifiable synthesis engine creating a self-sustaining cycle of rejection sampling and preference learning on verified trajectories.

8 Conclusion

In this work, we present EvoCUA, a native computer use agent developed through an evolving paradigm of learning from experience. By integrating verifiable synthesis with a scalable interaction infrastructure, we convert synthetic compute into high-quality training signals, achieving **56.7%** on OSWorld and achieving performance competitive with state-of-the-art models in the 30B-parameter class—a parameter-efficient alternative to massive closed-weights models.

A gap nonetheless persists between current open models and leading closed-weights systems, highlighting the limits of offline learning from synthesized traces. Our preliminary investigation into online reinforcement learning identifies active environmental interaction as a critical driver for further improvement, evidenced by consistent reward accumulation gains (Appendix B). Future work will systematically expand this online evolutionary boundary toward fully autonomous computer use.

Acknowledgments and Disclosure of Funding

We acknowledge Meituan for providing computational resources for this research. We thank the open-source community for advancing research on computer use agents. In particular, we are grateful to Xinyuan Wang and Tianbao Xie, the core authors of OpenCUA and OSWorld, respectively, for their insightful discussions, valuable feedback on evaluation, and continued support throughout this project.

We also thank our colleagues and family members for their support, encouragement, and helpful discussions: Chen Gao, Daorun Pan, Jiahui Wang, Jiangke Fan, Jianing Wang, Jiarong Shi, Kefeng Zhang, Rumei Li, Wenlong Zhu, Xuejia Shi, Xuezhi Cao, Ying Ouyang, Yerui Sun, Yuchao Zhu, Yufei Zhang, and Yuwei Jiang.

The authors declare no competing interests.

References

[1] 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.

[2] ByteDance Seed Team. Seed 1.8. `https://github.com/ByteDance-Seed/Seed-1.8/`, 2025. GitHub repository.

[3] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

[4] Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, et al. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025.

[5] Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*, 2024.

[6] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[7] Yuhao Yang, Zhen Yang, Zi-Yi Dou, Anh Nguyen, Keen You, Omar Attia, Andrew Szot, Michael Feng, Ram Ramrakhya, Alexander Toshev, et al. Ultracua: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*, 2025.

[8] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

[9] Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*, 2024.

[10] OpenAI. Computer-using agent (cua). `https://openai.com/index/computer-using-agent/`, 2025.

[11] Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, et al. Step-gui technical report. *arXiv preprint arXiv:2512.15431*, 2025.

[12] Anthropic. Introducing claude sonnet 4.5. `https://www.anthropic.com/news/claude-sonnet-4-5`, 2025.

[13] Anthropic. Introducing claude opus 4.6. `https://www.anthropic.com/news/claude-opus-4-6`, February 2026.

[14] OpenAI. Introducing gpt-5.4. `https://openai.com/index/introducing-gpt-5-4`, March 2026.

[15] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

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

[17] Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Foundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

[18] Qwen. Qwen 3.5. `https://huggingface.co/Qwen/Qwen3.5-27B`, February 2026.

[19] Moonshot AI. Kimi k2.6. https://www.kimi.com/ai-models/kimi-k2-6, April 2026.

[20] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

[21] 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.

[22] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

[23] Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508, 2025.

[24] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

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

[26] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[27] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

[28] Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025.

[29] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[30] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

[31] Kaixin Li, Meng Ziyang, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 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.

[32] Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. Scaling computer-use grounding via user interface decomposition and synthesis, 2025. URL https://arxiv.org/abs/2505.13227.

[33] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi, 2024. URL https://arxiv.org/abs/2311.16502.

[34] Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark, 2025. URL https://arxiv.org/abs/2409.02813.

[35] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts, 2024. URL https://arxiv.org/abs/2310.02255.

[36] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models?, 2024. URL https://arxiv.org/abs/2403.20330.

[37] Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xu-Cheng Yin, Cheng-Lin Liu, Lianwen Jin, and Xiang Bai. 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.

A Problem Formulation

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 the task distribution $\mathcal{T}_{syn}(\cdot \mid \pi_{\text{old}})$, where τ denotes trajectories induced by π_θ under instruction g.

B Future Work on Online Agentic RL

Reinforcement Learning with Verifiable Rewards (RLVR) [26] has become a crucial framework for boosting the reliability, generalization, and performance of models. 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 [29] 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 **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.

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.

C Training Data Card

Table 5 summarizes the initial synthetic task pool and the trajectory statistics across training stages. We distinguish task instances from trajectories: multiple rollouts can be collected for the same task, and each trajectory can subsequently be decomposed into multiple single-step training samples.

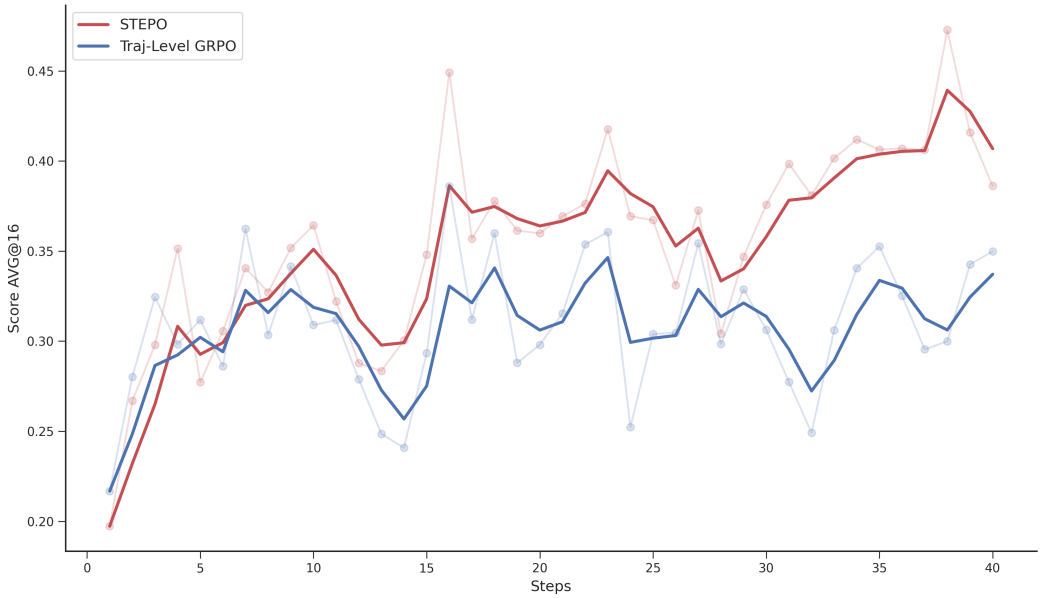


Figure 6: The values illustrated in this figure denote the 16-time average scores of the two methods, respectively.

Table 5: Summary of training-data statistics. Domain proportions and trajectory statistics are approximate.

Attribute	**Statistics**
Initial synthetic task pool	More than 10,000 tasks
Domain distribution	OS utilities: 35%; web navigation: 35%; office/productivity: 30%
Cold-start dataset	1,000 trajectories
Initial RFT rollout pass fraction	Approximately 60% pass the sandbox execution filters
Early-round successful trajectories	Approximately 10 steps on average
Later-round successful trajectories	Predominantly 20–30 steps

Successful rollouts provide candidates for RFT after further denoising, while failed rollouts can be paired with successful references for step-level DPO. As the policy evolves, task selection shifts toward its capability boundary, with later-round successful trajectories typically involving longer interaction horizons.

D Data Filtering and Decontamination

Execution-based filtering. We implement a multi-stage filtering protocol to reduce false labels and data leakage. 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 process combines execution results, auxiliary judgments, and human inspection to improve the quality of the retained training data.

Manual validator audit. Deterministic execution does not by itself guarantee that a validator correctly captures the semantic intent of an instruction. To assess this distinction, we randomly sampled 500 generated tasks across application domains and conducted a manual audit of instruction–validator alignment.

The audit examined whether each validator measured the intended task goal. A false-positive error occurs when the validator accepts an outcome that does not satisfy the instruction. A false-negative error occurs when the validator rejects an outcome that satisfies the instruction, including a valid alternative completion that is not recognized by an overly restrictive checking rule.

Table 6 summarizes the audit results. Strict semantic alignment was observed in 92.4% of the audited tasks. False-positive and false-negative errors accounted for approximately 0.6% and 7.0% of the audited sample, respectively. These percentages describe the composition of the audited sample, rather than conditional false-positive or false-negative rates.

Table 6: Manual audit of instruction–validator alignment on 500 randomly sampled generated tasks. Percentages are relative to the full audited sample.

Audit outcome	**Proportion**
Strict semantic alignment	92.4%
False-positive errors	$\sim$0.6%
False-negative errors	$\sim$7.0%

The observed errors were predominantly false negatives, reflecting cases in which strict validators failed to recognize valid alternative completions. False-positive errors were less frequent but remained nonzero. The audit therefore supports the use of executable validators as a scalable supervision mechanism while identifying residual semantic errors that require further filtering and refinement.

Benchmark decontamination. We apply a three-part decontamination process to reduce overlap between synthetic training tasks and evaluation benchmarks:

1. **Semantic decontamination:** remove instructions that are semantically equivalent to benchmark queries.
2. **Configuration decontamination:** prune tasks with initial application configurations that duplicate those of benchmark tasks.
3. **Evaluator decontamination:** check generated task-specific success conditions for overlap with existing benchmark evaluation scripts.

Together with execution-based filtering and manual inspection, these checks support the construction of a verifiable training pool containing tens of thousands of task instances.

E 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.

F 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.

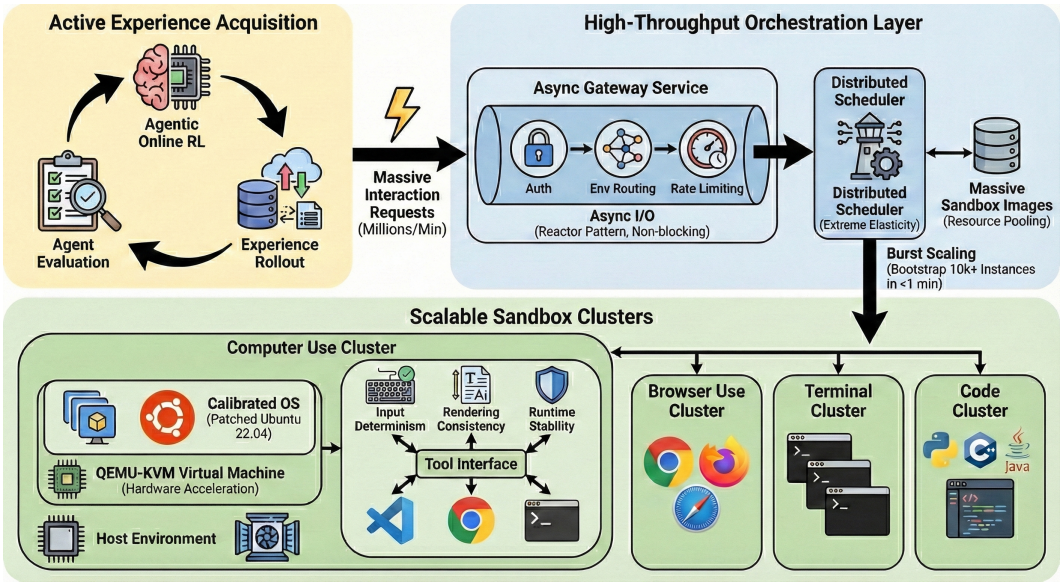


Figure 7: **Scalable Infrastructure.** The architecture orchestrates massive interaction requests from the online RL loop (top-left) through an asynchronous gateway and distributed scheduler (top-right). The bottom layer deploys parallel sandbox clusters, highlighting the **Computer Use Sandbox**, which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

F.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.

F.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.

F.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.

G 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 7: 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")

H Details Technical Implementation of Cold Start

H.1 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 $\mathcal{Z}$

H.2 Training Details

For model training, we decompose 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.

I Overall Training Loop

We describe the complete policy-conditioned training loop of EvoCUA in Algorithm 2. The pipeline begins with cold-start supervised fine-tuning and subsequently alternates between task-pool updates, interaction data collection, rejection sampling fine-tuning (RFT), and step-level offline DPO.

Task-pool updates and adaptive collection. At each iteration, the current policy's observed task-solving behavior guides task selection and the allocation of synthesis resources. Reliably solved tasks are pruned from active collection, while newly synthesized tasks expand the pool toward the policy's capability boundary. New tasks undergo the filtering and decontamination procedures described in Appendix D.

For each selected task, rollout collection follows the dynamic compute budgeting scheme in Section 4.2. We progressively increase the cumulative rollout budget through $\mathcal{K} = \{k_1, \ldots, k_n\}$ and evaluate the observed success rate $\mathrm{SR}(k_i)$ at each budget level. Collection stops at the first level satisfying $\mathrm{SR}(k_i) \geq \tau_i$; if no threshold is met, collection ends at the maximum budget k_n. Tasks that produce no successful trajectories within this budget do not contribute successful examples to RFT.

Sequential RFT and offline DPO. Successful trajectories are denoised to construct the RFT data. Failed trajectories are analyzed against successful references to identify critical error steps and construct preference pairs. RFT first updates the current policy using successful experience; offline DPO then updates the resulting intermediate policy using the constructed preference dataset. The updated policy is used for the next round of interaction.

The DPO pairs in this loop are constructed from the collected trajectories. Thus, policy-conditioned data collection does not imply that the DPO update itself is online reinforcement learning. The detailed preference-pair construction procedure is given in Algorithm 3.

In Algorithm 2, s_T^τ denotes the terminal state of trajectory τ.

Relation to the ablation experiments. The "Iterative Training" row in Table 3 corresponds to an additional complete iteration of task-pool updates, policy-conditioned rollout collection, RFT, and offline DPO. In contrast, Table 4 examines RFT-only training without the cold-start and DPO stages. These experiments evaluate different aspects of the pipeline and should be distinguished when interpreting the effect of iteration and data scale.

J 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 3 details how we identify Critical Forking Points from failure trajectories and construct paired data for both Action Correction and Reflection.

Algorithm 2 Policy-conditioned training loop of EvoCUA

Input: Base policy π_{base}; cold-start dataset $\mathcal{D}_{\text{cs}}$; initial task pool $\mathcal{T}_0$; number of iterations R; increasing cumulative rollout budgets $\mathcal{K} = \{k_1, \ldots, k_n\}$; descending success-rate thresholds $\Lambda = \{\tau_1, \ldots, \tau_n\}$
Output: Updated policy π_R

$\pi_0 \leftarrow \text{SFT}(\pi_{\text{base}}, \mathcal{D}_{\text{cs}})$
for $r = 0, \ldots, R-1$ **do**
  Update $\mathcal{T}_r$ using observed solve rates: prune reliably solved tasks and incorporate newly synthesized, filtered tasks
  $\mathcal{S}_r \leftarrow \emptyset$; $\mathcal{F}_r \leftarrow \emptyset$
  for each selected task $g \in \mathcal{T}_r$ **do**
    $n_g \leftarrow 0$; $k_0 \leftarrow 0$
    for $i = 1, \ldots, n$ **do**
      for $b = k_{i-1} + 1, \ldots, k_i$ **do**
        $\tau \leftarrow \textsc{Rollout}(\pi_r, g)$
        if $V_g(s_T^\tau) = 1$ **then**
          $\mathcal{S}_r \leftarrow \mathcal{S}_r \cup \{(g, \tau)\}$
          $n_g \leftarrow n_g + 1$
        else
          $\mathcal{F}_r \leftarrow \mathcal{F}_r \cup \{(g, \tau)\}$
        end if
      end for
      $\text{SR}_g(k_i) \leftarrow n_g / k_i$
      if $\text{SR}_g(k_i) \geq \tau_i$ **then**
        break
      end if
    end for
  end for
  $\mathcal{D}^r_{\text{RFT}} \leftarrow \textsc{DenoiseAndFormat}(\mathcal{S}_r)$
  $\pi^r_{\text{RFT}} \leftarrow \text{RFT}(\pi_r, \mathcal{D}^r_{\text{RFT}})$
  $\mathcal{D}^r_{\text{DPO}} \leftarrow \textsc{ConstructPreferencePairs}(\mathcal{F}_r, \mathcal{S}_r)$
  $\pi_{r+1} \leftarrow \textsc{OfflineDPO}(\pi^r_{\text{RFT}}, \mathcal{D}^r_{\text{DPO}})$
  Carry forward the task pool and observed solve-rate statistics for the next iteration
end for
return π_R

Algorithm 3 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 $\mathcal{D}_{dpo}$

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

K Per-Domain Results on OSWorld-Verified

Table 8 reports the per-domain results of EvoCUA-32B under a 50-step budget, using the EvoCUA-20260105 entry on the official OSWorld-Verified leaderboard.[1] Domain success rates are calculated from the published domain-level scores and task counts.

Table 8: Per-domain performance of EvoCUA-32B on OSWorld-Verified with a 50-step budget. Task counts follow the published leaderboard entry.

Domain	Tasks	Success rate (%)
Chrome	46	65.13
GIMP	26	76.92
LibreOffice Calc	47	55.32
LibreOffice Impress	47	59.49
LibreOffice Writer	23	69.57
Multi-app workflows	93	27.89
OS	23	78.26
Thunderbird	15	80.00
VLC	17	49.35
VS Code	23	86.96
Overall	**360**	**56.73**

Across LibreOffice Calc, Impress, and Writer, the combined task-weighted success rate is 59.79%. Performance varies across domains, with multi-app workflows remaining particularly challenging.

L 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 [30], ScreenSpot-Pro [31], OSWorld-G [32]) and general multimodal robustness (MMMU [33], MMMU-Pro [34], MathVista [35], MMStar [36], OCRBench [37]). Table 9 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 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.

[1] `https://osworld-v1.xlang.ai/`

Table 9: 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

M Computational Cost Breakdown

To facilitate reproducibility, we report the detailed computational footprint of training a converged EvoCUA-32B model.

Sandbox infrastructure. Our infrastructure is architected to scale to tens of thousands of concurrent sessions. For routine daily iterations, we typically deploy 5,000–10,000 sandboxes for mass experience acquisition. Each sandbox is allocated 2–4 CPU cores and 8 GB of RAM, with allocation scaled by task complexity. A single physical machine with 192 CPU cores comfortably instantiates 50–100 concurrent sandboxes.

Training pipeline. All training is performed on NVIDIA H800 GPUs. Table 10 summarizes the breakdown of GPU hours per training stage for one complete evolutionary iteration.

Table 10: Per-stage GPU-hour breakdown for one complete evolutionary iteration of EvoCUA-32B on NVIDIA H800 hardware.

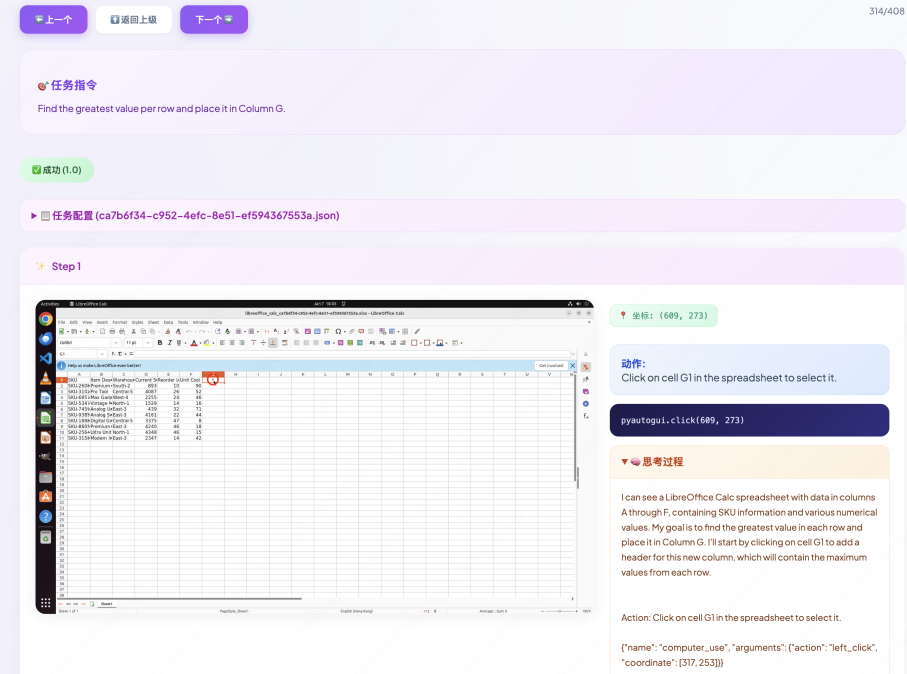
Stage	Sub-task	Wall time $\times$ #GPUs	GPU hours
Cold-start SFT	Training	12 h $\times$ 32	384
RFT phase	Rollout	24 h $\times$ 64	1,536
	Judge filtering	24 h $\times$ 64	1,536
	Training	12 h $\times$ 32	384
Step-level DPO	Training	3 h $\times$ 32	96
Total			**3,936**

A complete evolutionary iteration thus requires approximately 3,936 H800 GPU hours (~4 k GPU hours). The rollout and judge-filtering stages dominate the cost, reflecting the experience-centric nature of the proposed training paradigm; the targeted, error-driven sample budget of step-level DPO keeps its cost an order of magnitude smaller than the rollout phase. We note that the project-level total of more than one million accelerator hours reported in Appendix E accumulates over all preliminary, ablation, and failed experiments across both EvoCUA-8B and EvoCUA-32B variants.

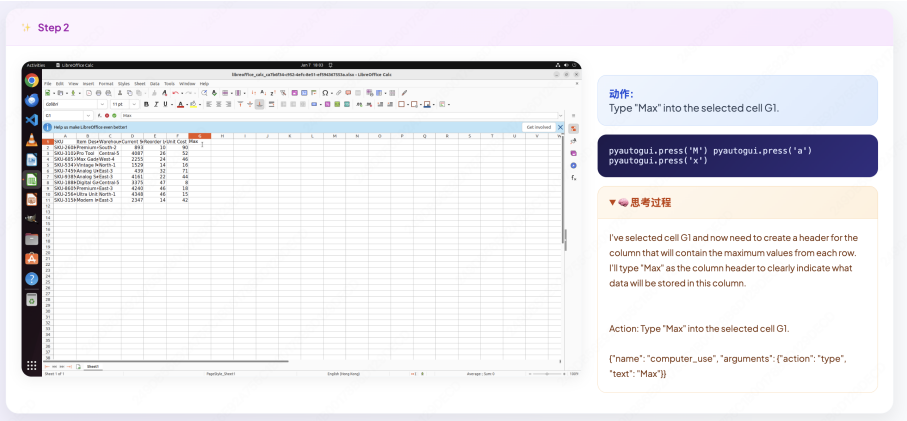
N Trajectory Analysis and Visualization

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.

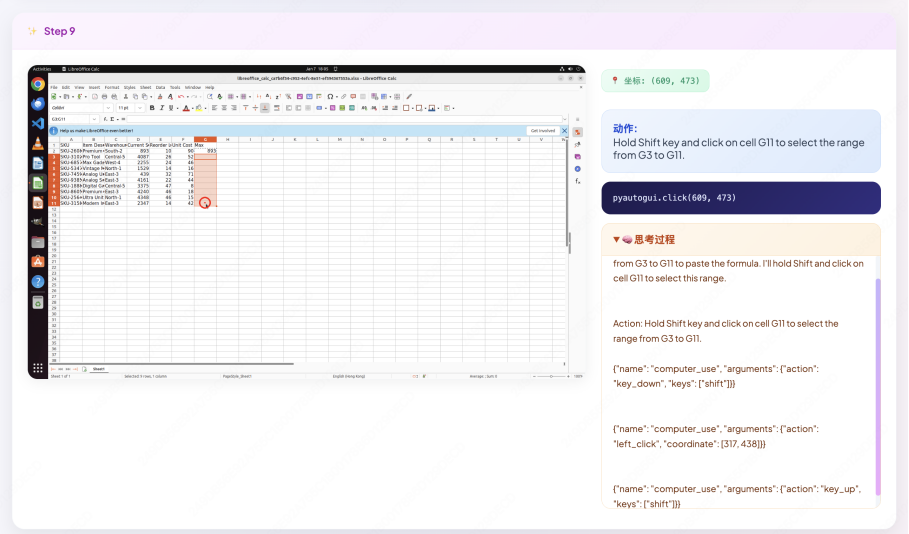


(b) Step 2: Text Entry ($t = 1$)**.** The system captures the transition from planning to execution. The agent's reasoning ("Type 'Max'...") is perfectly aligned with the generated atomic action sequence (`press('M'), press('a'), press('x')`).

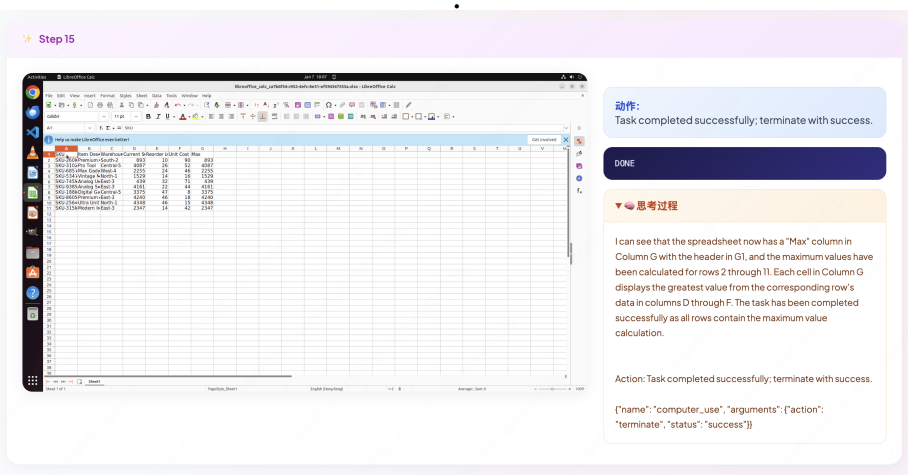
Figure 8: **Visualization of Synthesized Trajectory (Part I).** The EvoCUA Trajectory Inspector validates the logical consistency of synthetic training data: (a) Clear alignment between user instruction and agent planning; (b) Precise correspondence between reasoning and atomic keyboard actions.

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,



(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.

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.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state three contributions (verifiable synthesis, evolving learning from experience, open-source SOTA) and specific empirical claims (e.g., 56.7% on OSWorld), all of which are substantiated in Section 5 and the appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Conclusion discusses the residual gap to closed-weights systems and the limits of offline learning from synthesized traces; Section 6.1 (Scaling Analysis) further analyzes the trade-off between data scale and signal-to-noise ratio.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper presents an empirical methodology without formal theoretical results: there are no theorems, convergence proofs, or sample-complexity bounds claimed. We note that formal convergence analysis is intractable for multimodal GUI agents in open-ended OS environments due to the combinatorial state space and non-stationary transition dynamics; instead, the design relies on principled mechanisms (e.g., deterministic code validators, adaptive task sampling) whose effects are validated empirically in Sections 5 and 6.1.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We release model checkpoints (EvoCUA-8B and EvoCUA-32B) for both reproduction and follow-up research. The training pipeline, foundation models (Qwen3-VL-Thinking, OpenCUA), unified action space (Appendix G), hindsight reasoning generation procedure (Appendix H), and DPO pair construction (Algorithm 3) are described in sufficient detail to reproduce the main OSWorld and WindowsAgentArena results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [N/A]

   Justification: The paper's contribution is a training methodology rather than a specific dataset or benchmark. Reproducing the main results relies on publicly available foundation models (Qwen3-VL, OpenCUA) and the algorithmic details provided in the paper and appendices.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 reports training stages (cold start, RFT, DPO), foundation models, and evaluation budgets (e.g., 50 vs 100 max steps); per-stage details and hyperparameters appear in Sections 4.1–4.2 and Appendices B–H.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Main results in Table 1 report Pass@1 averaged across 3 independent rollouts on the official OSWorld-Verified evaluation harness, and we verified that the aggregate Pass@1 remain stable across runs (max-min spread < 2% on EvoCUA-32B). Additional variability evidence is provided in the Pass@k analysis (Fig. 5a, k=8–128).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix M provides a per-stage GPU-hour breakdown for one complete evolutionary iteration of EvoCUA-32B on NVIDIA H800 GPUs: cold-start SFT (384 GPU hours), RFT rollout (1,536 h) and judge filtering (1,536 h), RFT training (384 h), and step-level DPO (96 h), totaling ~3,936 H800 GPU hours per iteration. The sandbox infrastructure budget (5,000–10,000 concurrent sandboxes per daily iteration with 2–4 CPU cores and 8 GB RAM each) is documented in Appendix F. As required by the checklist guideline, we also disclose that the full research project including preliminary studies, ablations and failed experiments consumed substantially more compute (more than one million accelerator hours cumulative, Appendix E) than what is reported in the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: We have reviewed and adhered to the NeurIPS Code of Ethics; the work uses only synthetic sandbox data and publicly available foundation models, with no human-subjects data collection.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The Statement section explicitly discusses both positive impacts (advancing autonomous GUI agents) and negative impacts (data security, privacy leakage, ambiguous permission boundaries when deployed in real production).

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [Yes]

Justification: We mitigate misuse risk through two structural and documentation-level safeguards: (i) the released action space (Appendix G) is restricted to generic GUI primitives (mouse, keyboard, wait, terminate) and contains no privileged system-level, network or credential-handling operations, structurally bounding the misuse surface; (ii) the released checkpoints are distributed with a model card documenting intended research use, known failure modes (e.g., coordinate drift, premature termination) and recommended sandboxed deployment.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All foundation models (Qwen3-VL, OpenCUA), benchmarks (OSWorld, WindowsAgentArena, ScreenSpot, MMMU, MathVista, etc.) and supporting libraries are cited at first use; we use them under their respective open-source licenses (Apache-2.0/Qwen-license/research-only as applicable) and within their stated terms.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We release two new model checkpoints, EvoCUA-8B and EvoCUA-32B, accompanied by a model card that documents (i) the training pipeline (cold-start SFT, RFT, step-level DPO) referenced in Sections 4.1–4.2; (ii) intended research use and known failure modes (coordinate drift, premature termination, sensitivity to environmental stochasticity); (iii) the unified action space and input/output schema (Appendix G); and (iv) the detailed compute footprint (Appendix M). Licensing information is reported in the response to Question 12.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: All trajectories are synthesized inside automated sandboxes; no crowdworkers or human subjects participate in data collection or evaluation.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work involves no human-subjects research, so IRB or equivalent ethical review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: VLMs/LLMs are central, non-standard components of the methodology: a foundation VLM acts as the "task architect" that co-generates instructions and executable validators (Section 3); a general LLM is also used for hindsight reasoning generation (Appendix H) and as a judge in step-level denoising (Section 4.2). LLM-assisted text refinement of the manuscript itself is purely editorial and, per the NeurIPS LLM policy, does not require declaration.

    Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.