ToolCUA: Towards Optimal GUI-Tool Path Orchestration for Computer Use Agents

Xuhao Hu[1,2,*], **Xi Zhang**[2,*], **Haiyang Xu**[2,†], **Kyle Qiao**[2], **Jingyi Yang**[1], **Xuanjing Huang**[1], **Jing Shao**[3], **Ming Yan**[2,†], **Jieping Ye**[2]

[1]Fudan University [2]Alibaba Token Hub, Alibaba Group [3]Shanghai Artificial Intelligence Laboratory

[*]Equal Contribution [†]Corresponding Author

xuhaohu08@gmail.com; zx443053@alibaba-inc.com

Abstract

Computer Use Agents (CUAs) can act through both atomic GUI actions (*e.g.*, clicking and typing) and high-level tool calls (*e.g.*, API-based file operations), but they are often confused by this hybrid action space: they fail to determine when to continue with GUI actions and when to switch to tools, leading to suboptimal execution paths. This difficulty stems from two issues. First, high-quality interleaved GUI-Tool trajectories are scarce, and collecting real tool trajectories is expensive and brittle. Second, existing supervision provides limited guidance for GUI-Tool path selection, as most methods focus on step-level action imitation or final task completion and offer little trajectory-level feedback on whether GUI-Tool switching leads to a more effective execution path. In this paper, we propose **ToolCUA**, an end-to-end agent designed to learn optimal GUI-Tool path selection through a staged training paradigm. We first introduce an **Interleaved GUI-Tool Trajectory Scaling Pipeline** that repurposes abundant static GUI trajectories and synthesizes a grounded library of simulated tools, making it possible to scale diverse GUI-Tool trajectories without manual engineering or real tool-trajectory collection. Based on this data, we perform Tool-Bootstrapped GUI RFT, which combines warmup SFT with single-turn RL to improve decisions at critical GUI-Tool switching points. Finally, we further optimize ToolCUA with **Online Agentic RL** in a high-fidelity GUI-Tool environment, using a **Tool-Efficient Path Reward** that encourages both appropriate tool use and shorter execution paths. Experiments on OSWorld-MCP show that ToolCUA achieves 46.85% accuracy, a relative improvement of approximately 66% over the baseline, establishing a new state of the art among models of comparable scale. It also improves by 3.9 percentage points over the GUI-only setting, demonstrating effective GUI-Tool orchestration. The results further suggest that training in a hybrid action space is a promising paradigm for real-world digital agents.

1 Introduction

The rapid evolution of Multimodal Large Language Models (MLLMs) [2–4, 8, 37, 62] toward agentic capabilities [17, 44, 47, 58] has established Computer Use Agents (CUAs) [21, 27, 31, 40, 42, 43, 49, 52, 65] as a frontier topic for automating native desktop workflows. Conventionally, CUAs primarily rely on atomic GUI actions (*e.g.*, clicking and scrolling), which offer broad generalizability but are susceptible to cascading errors in long-horizon tasks. In contrast, structured tool calls [10, 30, 38, 47] provide agents with superior efficiency and precision [64, 65]. For example, in Figure 1 (a), modifying an entire column in LibreOffice can be completed by a single API call, whereas a pure GUI solution requires a long sequence of clicks and keystrokes. However, tool-based APIs are constrained by

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

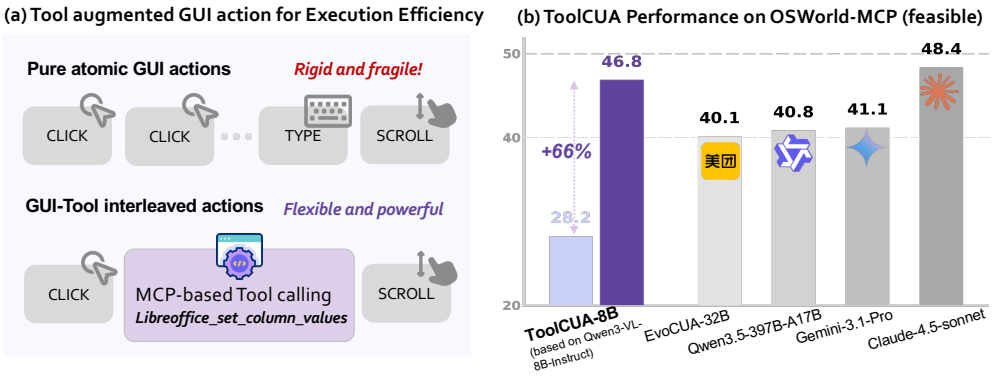


Figure 1: (a) The advantage of Tool-augmented actions compared with pure GUI actions. (b) The performance of our **ToolCUA** compared with the baselines, agentic CUAs, and general models.

service coverage and stability, limiting applicability in diverse scenarios. Therefore, given their complementary strengths, a hybrid GUI-Tool action space is essential for next-generation CUAs.

Although GUI actions and tool calls are complementary, simply exposing both action spaces to an MLLM does not solve the problem. In practice, agents are often **confused by the hybrid action space**. As shown in Table 1, some models invoke tools frequently on successful tool-beneficial tasks yet suffer lower overall task success, while others (*e.g.*, Qwen3VL-8B) remain overly GUI-centric and almost never invoke tools even when more efficient tool calls are available. We formalize this challenge, illustrated in Figure 2, as **optimal GUI-Tool path selection**: dynamically determining when to use GUI actions and when to invoke tools so as to form an efficient and reliable task trajectory. Unlike step-level action selection, this is inherently a trajectory-level policy learning problem, as each GUI-to-Tool or Tool-to-GUI switching decision not only solves the immediate step, but reshapes the entire subsequent trajectory in terms of efficiency and reliability.

To this end, existing approaches fall short in two fundamental aspects. First, **current CUAs are often undertrained on tool use, exhibiting a deficit in tool-calling knowledge**. This limitation is rooted in the lack of high-quality interleaved GUI-Tool trajectories. In real computer-use environments, usable tools are difficult to obtain and maintain. Specifically, APIs are often application-specific, incomplete, or unstable, and collecting GUI-Tool data requires expensive environment instrumentation. Existing efforts [52, 57] partly address this by generating tools from code, but such pipelines remain costly to scale and do not fully exploit the large amount of existing GUI-only trajectory corpora [15, 21, 26, 43, 48, 63]. Second, even when basic tool-use ability is available, existing supervision provides **limited guidance for learning effective hybrid action orchestration**. In practice, current training signals usually come from either step-level imitation or final task-completion rewards. The former captures only local action plausibility, while the latter does not distinguish between a timely tool-call switch and a long, brittle GUI-only workaround. As a result, the model cannot reliably learn whether switching between GUI actions and tool calls improves the full trajectory.

To address these challenges, we introduce **ToolCUA**, a unified agentic model trained through a staged paradigm. It first builds hybrid-action foundations with scalable interleaved GUI-Tool data and then improves trajectory-level GUI-Tool decisions through reinforcement learning. First, we propose an **interleaved GUI-Tool trajectory scaling pipeline** built on existing static GUI corpora. It employs MLLMs to synthesize a trajectory-aware library of tools from recurrent GUI procedures and converts GUI-only data into interleaved GUI-Tool trajectories through tool-step generation and next-state grounding. By repurposing existing GUI corpora and synthesizing tools instead of collecting expensive real tool trajectories, this pipeline enables scalable data construction without manual engineering, while covering varied tool granularities and switching contexts. Building on this data, we perform Tool-Bootstrapped GUI Reinforcement Fine-Tuning (RFT). It first applies warmup SFT to establish basic hybrid-action capabilities, and then uses single-turn RL to improve decisions at explicit GUI-Tool switching points. Finally, we optimize ToolCUA with **Online Agentic RL** in a realistic GUI-Tool environment with a **Tool-Efficient Path Reward**, which includes a tool appropriateness term R_{tool} and path efficiency term R_{length}: R_{tool} incentivizes the agent to invoke tools when beneficial and abstain when unnecessary, while R_{length} encourages shorter execution

Table 1: Performance comparison between pure GUI and hybrid GUI-Tool action spaces on the matched feasible task set of OSWorld [48] and OSWorld-MCP [14]. "Steps" is the average number of trajectory steps. For the hybrid setting, "Tool-calls" is averaged over successful trajectories in the Tool-Beneficial subset. See details in Appendix C.1.

Model	Action	Accuracy ↑	Steps ↓	Tool-calls
Qwen3VL-8B	GUI	29.0	19.2	-
	+ Tools	28.2 (−0.8)	19.3	0.01
Qwen3VL-235B	GUI	41.1	25.9	-
	+ Tools	38.1 (−3.0)	18.0	3.62
EvoCUA-32B	GUI	52.6	25.0	-
	+ Tools	40.5 (−12.1)	26.2	6.23
Claude-4-Sonnet	GUI	47.7	23.6	-
	+ Tools	43.5 (−4.2)	19.8	4.30
Claude-4.5-Sonnet	GUI	61.9	23.3	-
	+ Tools	48.4 (−13.5)	19.1	3.81
ToolCUA-8B (Ours)	GUI	42.9	19.4	-
	+ Tools	**46.8** (+3.9)	**14.9**	**2.11**



Figure 2: Current computer use agents suffer from optimal path confusion under GUI-Tool hybrid actions.

paths by replacing redundant GUI actions with tool calls. Together, they provide trajectory-level feedback that drives the model toward globally optimal GUI-Tool path selection.

Experimental results demonstrate that ToolCUA achieves 46.85% accuracy on OSWorld-MCP [14], establishing state-of-the-art performance among models of comparable scale. This represents an approximately 66% relative improvement over the Qwen3-VL-8B-Instruct baseline [3] and is competitive with leading proprietary models [2, 12]. Furthermore, ToolCUA achieves 42.9% accuracy in the pure-GUI setting and gains another 3.9 percentage points when tools are available, demonstrating effective GUI-Tool path selection. Additionally, ToolCUA shows out-of-distribution generalization across tasks and platforms, reaching 23.9% on unseen *multi_apps* Linux tasks and achieving 33.8% on unseen *Windows* desktop apps in WindowsAgentArena [6]. These results confirm that operating in a hybrid GUI-Tool action space is essential for achieving generalizable and efficient real-world digital automation. Our main contributions are summarized as follows:

- We propose an **Interleaved GUI-Tool trajectory scaling pipeline** that repurposes existing pure GUI corpora into scalable hybrid-action training data through tool synthesis, obviating the need for manual tool-environment construction and tool trajectory collection.
- We propose a staged training paradigm consisting of **Tool-Bootstrapped GUI RFT**, which establishes hybrid-action foundations and calibrates GUI-Tool switching decisions, followed by **Online Agentic RL** with a **Tool-Efficient Path Reward** (R_{tool} and R_{length}) for trajectory-level optimization.
- Our **ToolCUA** reaches 46.85% accuracy on OSWorld-MCP, achieving state-of-the-art performance among similarly sized models and outperforming its pure-GUI counterpart. Our findings suggest that training in a hybrid GUI-Tool action space enables more generalizable and efficient computer-use automation.

2 ToolCUA

2.1 Definition and Scope

We first formalize the computer-use task as a Markov Decision Process (MDP) where $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. At each time step t, the state $s_t \in \mathcal{S}$ denotes a multimodal observation encompassing both the desktop screenshot and previously invoked tool results. The agent interacts with the environment through a hybrid action space $\mathcal{A} = \mathcal{A}_{GUI} \cup \mathcal{A}_{Tool}$, where $a_t \in \mathcal{A}_{GUI}$ represents atomic GUI interactions such as coordinate-based clicks, and $a_t \in \mathcal{A}_{Tool}$ signifies high-level structured tool invocations. The objective is to learn an optimal policy $\pi_\theta(a_t|s_t)$ that maximizes the expected

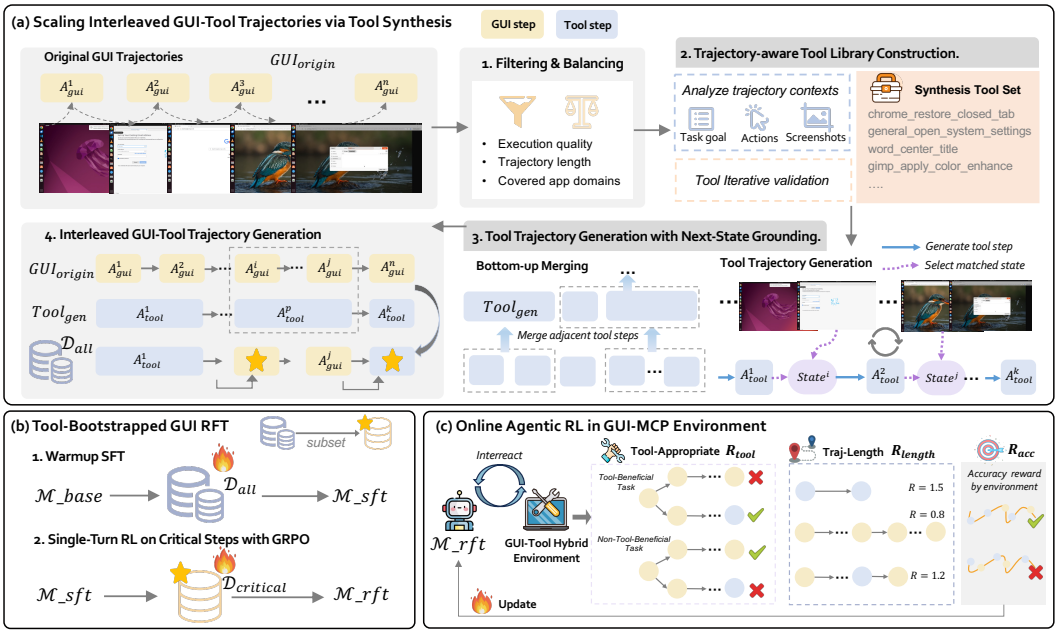


Figure 3: Overview of the ToolCUA data collection and training paradigm. (a) Interleaved GUI-Tool trajectory scaling via offline GUI data and tool synthesis, yielding the dataset $\mathcal{D}_{all}$ containing critical switching steps $\mathcal{D}_{critical}$. (b) Tool-Bootstrapped GUI RFT with warmup SFT on $\mathcal{D}_{all}$ and step-level RL on $\mathcal{D}_{critical}$. (c) Online Agentic RL with Tool-Efficient Path Reward (R_{tool} and R_{length}) for trajectory-level optimization with appropriate tool use and efficient path selection.

cumulative reward over a long-horizon trajectory τ:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t)\right] \tag{1}$$

2.2 Interleaved GUI-Tool Trajectory Scaling Pipeline

To address the scarcity of interleaved GUI-Tool trajectories, we build an offline trajectory scaling pipeline that starts from **existing pure GUI trajectories** and converts them into interleaved GUI-Tool data. As shown in Figure 3(a), the key idea is to use an MLLM (*e.g.*, Kimi-K2.5 or Claude-4.5-Sonnet) to synthesize a **grounded library of simulated tools** from recurrent GUI procedures, and then use these tools to transform GUI-only trajectories into interleaved GUI-Tool trajectories. These simulated tools are symbolic definitions and calls used for execution-free offline supervision; they do not require concrete tool implementation or sandbox execution at this stage. Our pipeline scales data along three dimensions: Tool Functionality across application domains, Tool Granularity from atomic utilities to composite skills, and GUI-Tool Switching Context covering cases where tool use is more or less beneficial. Please refer to Appendix E for prompts we used. We describe the main steps below.

Trajectory Filtering and Balancing. We start from successful raw GUI trajectories and filter them by execution quality, task length, and application coverage. The remaining trajectories are balanced across domains to provide a stable source distribution for tool synthesis.

Trajectory-Aware Synthetic Tool Library Construction. For each GUI trajectory, we use an MLLM to synthesize a candidate library of tools by analyzing the pure GUI path, including the task goal, action sequences, and dense screenshot descriptions. Each synthesized tool abstracts an observed GUI procedure into a callable high-level operation, specified by a functional signature, natural language description, and argument semantics inferred from the trajectory. This makes the tools grounded in concrete trajectory behavior rather than generic API templates or manually predefined functions. To increase **diversity**, we synthesize tools at varying levels of specificity, from single-action wrappers (*e.g.*, `chrome_open_settings`) to multi-step composite functions (*e.g.*,

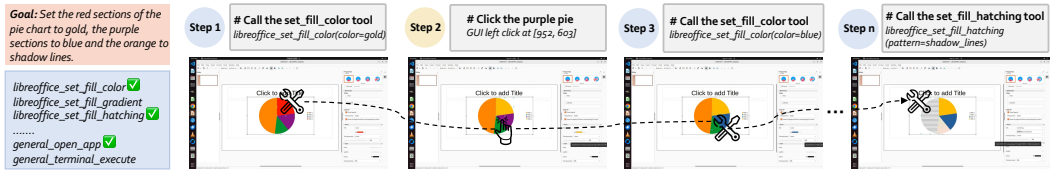


Figure 4: A synthetic interleaved GUI-Tool trajectory generated by our pipeline, demonstrating strategic tool selection and seamless switching between atomic GUI actions and tool calls.

`chrome_open_language_settings`). A rule-based format verification is also applied for tool filtering.

Tool Trajectory Generation with Next-State Grounding. Given the synthesized tool library and the original GUI trajectory, we adopt an MLLM to generate a functionally equivalent tool-only trajectory. For each step, the MLLM selects an appropriate tool from the library, produces a chain-of-thought rationale, and predicts the expected response; the generated call is then validated against the tool schema. Furthermore, we use an MLLM to perform **next-state grounding**, *i.e.*, to anchor each simulated tool call to the real next-state screenshot reached by the corresponding contiguous GUI segment in the original successful trajectory. This verifies that the predicted tool effect is consistent with the observed GUI state while preserving the verified endpoint of the source demonstration. Candidates that fail schema validation or next-state grounding are discarded. We also apply a **bottom-up merging strategy**: adjacent fine-grained steps sharing a common sub-goal are progressively merged into higher-level composite tool calls, yielding multiple variants at different levels of tool granularity.

Interleaved GUI-Tool Trajectory Generation. Given a grounded tool-only trajectory, we randomly sample a subset of tool calls and replace each with its corresponding GUI action sequence from the original trajectory. Notably, the replaced tools are simultaneously removed from the tool library, constructing a partial tool-availability context where the agent must fall back to GUI operations when certain tools are unavailable. By varying the selection of replaced tool calls, we generate diverse interleaved variants from the same trajectory, which are aggregated into $\mathcal{D}_{all}$. A representative interleaved GUI-Tool trajectory is illustrated in Figure 4. Furthermore, each replacement naturally exposes two types of boundary transitions: GUI $\rightarrow$ Tool and Tool $\rightarrow$ GUI (*i.e.*, the yellow star in Figure 3(a)), where the agent switches between GUI and tool calls. We refer to these as **critical switching steps** and collect them into $\mathcal{D}_{critical}$.

2.3 Tool-Bootstrapped GUI RFT

With $\mathcal{D}_{all}$ and $\mathcal{D}_{critical}$, we perform Tool-Bootstrapped GUI RFT to equip the baseline agent $\mathcal{M}_{base}$ with flexible hybrid-action behavior and calibrate local GUI-Tool decisions.

Warmup Supervised Fine-Tuning (SFT). We first perform SFT on $\mathcal{D}_{all}$ using a standard cross-entropy loss $\mathcal{L}_{SFT} = -\sum \log \pi_\theta(a_t|s_t)$. This phase teaches the model multimodal tool-calling knowledge in the CUA domain, including tool selection, argument generation, and the resulting states after tool execution. This SFT warmup yields the model $\mathcal{M}_{sft}$.

Single-Turn RL on Critical Steps. Building upon the model $\mathcal{M}_{sft}$, we implement a single-turn RL phase using the Group Relative Policy Optimization (GRPO) algorithm [33] on $\mathcal{D}_{critical}$. By sampling multiple completions at these critical switching steps, the model receives direct feedback on whether to continue with GUI actions or switch to tool calls when appropriate tools are available. This targeted optimization calibrates the model's discernment at decision boundaries, yielding a coordinated agent $\mathcal{M}_{rft}$ ready for long-horizon online exploration in the GUI-Tool environment.

2.4 Online Agentic RL with Tool-Efficient Path Reward in GUI-Tool Environment

Online RL extends step-level tool-calling knowledge to complete trajectories, enabling the agent to discover which GUI-Tool switching strategies lead to successful outcomes through real environment exploration. However, task success alone cannot distinguish whether tool usage was genuinely appropriate, nor whether the execution path was unnecessarily long. Therefore, we introduce a **Tool-Efficient Path Reward** that explicitly shapes the agent toward tool-appropriate and efficient

trajectories, which consists of a tool appropriateness term R_{tool} and path efficiency term R_{length}:

$$R = R_{fmt} + R_{acc} + \lambda \cdot R_{tool} + \beta \cdot R_{length} \tag{2}$$

where R_{fmt} and R_{acc} are standard format and accuracy rewards, while R_{tool} and R_{length} are activated only when the trajectory succeeds.

Tool Appropriateness Reward Term. In practice, agents may complete a task without tools even when tools would help, or invoke tools unnecessarily on tasks that do not require them. R_{tool} addresses this by introducing a task-level tool-beneficial label $t_b \in \{1, -1\}$ annotated during data construction. Here, $t_b = 1$ indicates that the task favors tool usage and $t_b = -1$ indicates that tool usage is unnecessary. Let c denote the cumulative number of tool calls in a trajectory.

$$R_{tool} = \mathbb{I}_{succ} \cdot \mathbb{I}[(t_b > 0 \wedge c > 0) \vee (t_b < 0 \wedge c = 0)] \tag{3}$$

R_{tool} is assigned when the agent invokes tools on tool-beneficial tasks ($t_b = 1, c > 0$), or when it deliberately abstains from tools on non-tool-beneficial tasks ($t_b = -1, c = 0$). This design augments task success with an explicit tool-appropriateness signal, encouraging the agent to use tools when and only when they are beneficial.

Path Efficiency Reward Term. Even when the agent succeeds and uses tools appropriately, it may still take unnecessarily long paths, for example, by relying on redundant GUI operations when a single tool call could accomplish the same effect. To this end, R_{length} encourages the agent to actively explore and discover more efficient GUI-Tool execution paths through online interaction. Rather than measuring efficiency against a fixed threshold, we evaluate trajectory length relative to the rollout group, where s is the current trajectory's step count, $\bar{s}$ is the group-average step count, and S_{max} is the maximum execution horizon.

$$R_{length} = \mathbb{I}_{succ} \cdot \begin{cases} (1 + \frac{\bar{s} - s}{\bar{s}}) & s < \bar{s} \\ \exp(-\frac{s - \bar{s}}{S_{max} - \bar{s}}) & s \geq \bar{s} \end{cases} \tag{4}$$

For trajectories shorter than the group average, the agent receives a linear bonus proportional to the relative step reduction; otherwise, the reward decays exponentially as the trajectory grows longer. Since useful tool calls often replace multiple atomic GUI operations, this signal naturally incentivizes the agent to switch to tools when they lead to a shorter and more reliable execution path.

With the above reward, we optimize ToolCUA using multi-turn GRPO over online rollouts in a GUI-Tool environment. Inspired by DAPO [61], we apply dynamic filtering and retain only rollout groups containing both successful and failed trajectories, which improves the informativeness of group-relative policy updates while reducing unnecessary computation.

3 Experiments

3.1 Experimental Settings

Implementation Details. Our pipeline aggregates diverse trajectories from open-source datasets [21, 43], as detailed in Appendix C.3. We adopt Qwen3-VL-8B-Instruct [3] as our base model $\mathcal{M}$. In the warmup SFT stage, we train $\mathcal{M}$ for 3 epochs and subsequently conduct single-turn RL with a group size of 32. During the subsequent online agentic RL stage, we set the reward weights to $\lambda = 0.4$ and $\beta = 0.2$, and set $S_{max} = 30$ as the maximum execution horizon. The training configuration for this stage includes a rollout size of 32 per group, a learning rate of 1×10^{-6}, and a training batch size of 32 to obtain **ToolCUA**. We further optimize the tool-calling interface by designing an agent-readable return format that provides concise, semantically dense feedback to reduce token overhead and improve grounding accuracy. For online agentic RL, we use a mixture of original, rewritten, and newly scaled single-domain tasks, while excluding the entire *multi_apps* domain for held-out evaluation. Please refer to Appendix C.4 for the task composition and construction details.

Baselines and Benchmark. We evaluate ToolCUA against two categories: general-purpose foundation models, including Qwen3.5-Plus [32], Claude-4.5-Sonnet [2], and Gemini-3.1-Pro [12]; and specialized CUAs, including UI-Tars-1.5 [31], the EvoCUA series [51], and GUI-Owl-1.5 [49].

For evaluation, we use OSWorld-MCP [14] as our primary benchmark. It evaluates CUAs in a hybrid action space covering standard GUI actions, more than 150 tools, and mainstream desktop

Table 2: Performance on OSWorld-MCP. Results are reported on the feasible tasks only. Tool-Beneficial Tasks favor tool usage, while Non-Tool-Beneficial Tasks do not require tools. Accuracy denotes average success rate; TIR (Tool Invocation Rate) measures whether the agent uses tools when beneficial and avoids them when unnecessary; ACS denotes Average Completion Steps.

Agent Model	Tool-Beneficial Tasks (238)			Non-Tool-Beneficial Tasks (95)			Overall (333)		
	Accuracy	TIR	ACS	Accuracy	TIR	ACS	Accuracy ↑	TIR ↑	ACS ↓
General Model									
Gemini-2.5-Pro	24.79	21.15	26.12	8.77	7.37	39.60	20.22	17.22	29.97
OpenAI o3	26.89	24.51	26.71	4.91	2.46	44.78	20.62	18.22	31.87
Seed1.5-VL	34.03	23.81	18.99	35.79	34.39	24.93	34.53	26.83	20.69
Claude-4-Sonnet	45.24	36.13	18.29	39.30	34.74	21.09	43.54	35.74	19.76
Gemini-3.1-Pro	44.54	37.39	22.46	32.63	26.32	32.82	41.14	34.23	25.40
Claude-4.5-Sonnet	50.00	42.02	17.93	44.21	35.79	21.91	48.35	40.24	19.07
Qwen3-VL-235B-A22B	37.11	28.15	17.04	40.70	29.82	20.25	38.14	28.63	17.95
Qwen3.5-397B-A17B	41.60	0.84	21.74	38.95	38.95	22.17	40.84	11.71	21.86
Specialized CUA Model									
UI-Tars-1.5-7B	10.92	0	37.38	15.79	15.79	36.43	12.31	4.5	37.11
EvoCUA-8B	34.45	3.78	26.88	38.95	38.95	26.49	35.74	13.81	26.77
EvoCUA-32B	37.82	13.03	27.90	47.37	46.32	21.82	40.54	22.52	26.16
GUI-Owl-1.5-8B	44.54	37.82	20.70	42.11	31.58	22.41	43.84	36.04	21.19
GUI-Owl-1.5-32B	47.48	38.66	24.09	49.47	47.37	24.45	48.05	41.14	24.19
Ours (based on Qwen3-VL-8B-Instruct)									
Qwen3-VL-8B-Instruct	27.73	0.00	20.37	29.47	29.47	16.77	28.23	8.41	19.34
ToolCUA-8B	**45.80**	**15.13**	**15.11**	**49.47**	**47.37**	**14.48**	**46.85**	**24.32**	**14.93**
Δ	+18.07	+15.13	-5.26	+20.00	+17.90	-2.29	+18.62	+15.91	-4.41

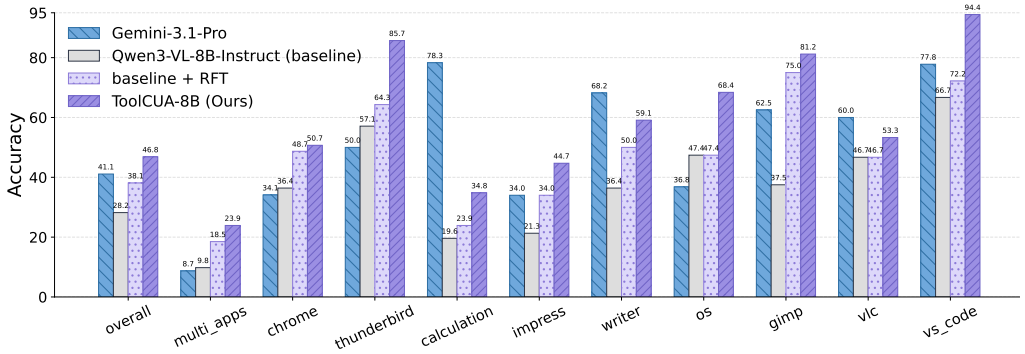


Figure 5: Results across OSWorld-MCP application domains for Gemini-3.1-Pro, Qwen3-VL-8B-Instruct (baseline), the RFT checkpoint ($\mathcal{M}_{rft}$), and ToolCUA-8B.

applications. Following the benchmark setup, we report results on the feasible tasks only. To mitigate environmental stochasticity in the sandbox, we report *average@3* for all primary metrics and set the maximum number of steps per task to 50. We follow the original benchmark metrics (detailed in Appendix C.5), where TIR measures whether the agent uses tools when beneficial and avoids them when unnecessary, and ACS measures average completion steps as an indicator of execution efficiency. Furthermore, we evaluate the cross-task and cross-platform transferability of ToolCUA on unseen Linux *multi_apps* tasks and unseen Windows apps in WindowsAgentArena [6].

3.2 Main Results

Outstanding performance on GUI-Tool Execution Path Selection. Table 2 summarizes the evaluation results on the OSWorld-MCP benchmark, where ToolCUA-8B achieves state-of-the-art performance of 46.85% among 8B-class models. Our model surpasses the previous state-of-the-art GUI-Owl-1.5-8B (43.84%) and outperforms prominent general foundation models, including Gemini-3.1-Pro (41.14%) and Claude-4-Sonnet (43.54%), while trailing the top-tier Claude-4.5-Sonnet by 1.50 percentage points. This approximately 66% relative improvement over the baseline (28.23%) underscores the efficacy of our synthesis-driven scaling pipeline and staged training paradigm.

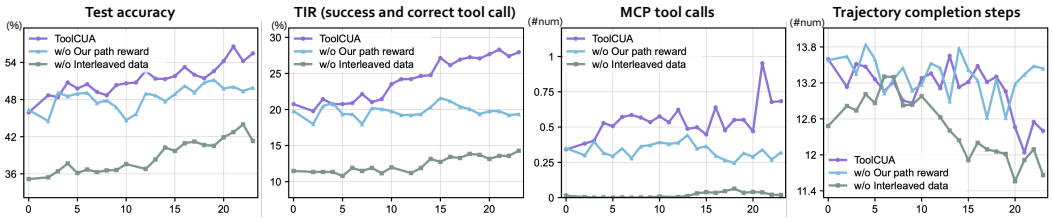


Figure 6: Online Agentic RL training dynamics of ToolCUA and two ablations. *w/o Interleaved data* removes the offline interleaved GUI-Tool data bootstrapping and directly performs online agentic RL from the baseline model. *w/o Our path reward* replaces the tool-efficient path reward with vanilla GRPO during online agentic RL.

Beyond task accuracy, ToolCUA also substantially improves GUI-Tool orchestration and execution efficiency. As shown in Table 1, ToolCUA-8B gains 3.9 percentage points over the pure-GUI setting. Also, compared with the Qwen3-VL-8B-Instruct baseline, ToolCUA increases the overall TIR from 8.41% to 24.32%, while reducing ACS from 19.34 to 14.93. Notably, ToolCUA achieves the lowest ACS (14.93 steps) among all evaluated models, indicating that it not only completes more tasks but also finds more efficient GUI-Tool execution paths with our Tool-Efficient Path Reward.

Cross-task and Cross-platform Generalization. ToolCUA also demonstrates strong generalization beyond the training distribution. Although online agentic RL is conducted only on single-application Linux tasks and excludes the *multi_apps* category, ToolCUA reaches 23.9% on this held-out domain, improving over both the pre-online-RL checkpoint (18.5%) and the baseline (9.8%), as shown in Figure 5. It also achieves consistent gains across specialized domains, increasing performance from 19.6% to 34.8% on *libreoffice_calculation* and from 66.7% to 94.4% on *vs_code*. Beyond cross-task transfer, ToolCUA further generalizes to unseen Windows desktop environments, reaching 33.8% on the full GUI-only WAA evaluation. For a controlled hybrid-action comparison, Table 4 evaluates both GUI-only WAA and WAA with mcp tool support on the same 107-task subset. ToolCUA reaches 37.7% with GUI actions and improves further to 40.5% with GUI-Tool actions, whereas the Qwen3-VL-8B-Instruct baseline decreases from 28.0% to 26.8% after tools are introduced. This shows that ToolCUA transfers not only its GUI capability but also its learned GUI-Tool orchestration to a new platform and tool ecosystem.

3.3 Ablation Analysis

The Importance of the Interleaved GUI-Tool Trajectory Data. As shown by the "w/o Interleaved data" setting in Figure 6, we remove cold-start RFT training with our synthetic interleaved GUI-Tool data and directly apply online agentic RL with the tool-efficient path reward to the Qwen3-VL-8B-Instruct baseline model. Although the model still improves its overall task accuracy during agentic RL, it struggles to acquire reliable tool-calling behavior without the offline interleaved supervision produced by our trajectory scaling pipeline. Specifically, its TIR remains consistently low and only reaches around 15% by the end of training, while the number of tool calls stays close to zero throughout most of the learning process. This suggests that online RL with tool-efficiency rewards alone is insufficient to overcome the GUI-centric bias of base models. Instead, our data scaling pipeline synthesizes simulated tools into static GUI data to provide grounded, interleaved supervision. This instills diverse tool knowledge and switching priors, establishing a superior foundation for subsequent online exploration.

Advantages of Tool-Efficient Path Reward. As illustrated by the "w/o Our path reward" setting in Figure 6, we further train the RFT-initialized model with vanilla multi-turn GRPO, using only the standard R_{acc} and R_{fmt} rewards. Without the tool-appropriate and trajectory-length reward, the accuracy curve becomes less stable than ToolCUA, showing a clear drop around steps 8–11 and an eventual gap of about 7 percentage points after 20 training steps. Moreover, both TIR and tool-calls fluctuate without a consistent upward trend, suggesting that the model does not learn when tool invocation is genuinely beneficial. The trajectory length also lacks a stable downward trend and rebounds after temporary decreases, indicating that vanilla agentic RL does not reliably discover more efficient execution paths. These results validate that the Tool-Efficient Path Reward is essential for turning online exploration into tool-appropriate and efficiency-aware GUI-Tool path selection.

Table 3: Comparison of pure GUI training and training with hybrid GUI-Tool action spaces. Accuracy is reported in OSWorld-MCP.

Action	Agent Model	Accuracy (%)
GUI	Qwen3-VL-8B-Instruct	29.03
	+ SFT	34.93
	+ SFT + Agentic RL	42.05
GUI+Tool	+ RFT	38.13
	ToolCUA-8B	**46.85**

Table 4: Cross-platform evaluation on the same 107-task subset of WindowsAgentArena [6] under the GUI-only setting and the setting with mcp tool support. Results are reported as average@3. Tool calls are averaged over successful tasks in the latter setting.

Agent Model	WAA	WAA + mcp tools	Tool Calls
Qwen3-VL-8B-Inst.	28.0	26.8	0.003
Qwen3-VL-32B-Inst.	34.4	33.1	0.05
ToolCUA-8B	**37.7**	**40.5**	1.74

Hybrid GUI-Tool Training Is More Effective than Pure-GUI Training. We compare hybrid GUI-Tool training with a pure GUI training pipeline in Table 3. For the pure GUI setting, we first cold-start the baseline using a GUI-only dataset of comparable scale to ToolCUA's warmup data, and then conduct agentic RL entirely within the pure GUI action space. Although GUI-only training improves the baseline from 29.03% to 34.93% after SFT and further to 42.05% after agentic RL, both stages remain below their GUI-Tool counterparts. In comparison, RFT training with our synthetic interleaved GUI-Tool trajectories already reaches 38.13%, and the full ToolCUA further improves to 46.85% after online agentic RL. These results indicate that the hybrid GUI-Tool action space provides a more effective training environment than pure GUI actions, enabling the agent to learn not only visual grounding but also when structured tool calls can replace redundant low-level operations.

3.4 Case Study

As illustrated in Table 9, the agent is tasked with creating two pivot tables in a new sheet named Sheet2 to summarize revenue by product and sales channel. While standard GUI-only approaches involve a laborious and error-prone sequence of data range selection and menu navigation, ToolCUA leverages high-level tools for superior efficiency. Specifically, it first retrieves sheet information such as column values and data fields via *libreoffice_calc.env_info(Sheet1)* and then identifies the corresponding data range needed for each pivot table. After that, the agent invokes *libreoffice_calc.create_pivot_table(source_sheet, table_name, ...)* with the corresponding parameters to generate the tables directly, bypassing brittle step-by-step GUI interactions. Furthermore, ToolCUA demonstrates the ability to navigate optimal execution paths within hybrid GUI-Tool action spaces. As shown in Table 10, the agent successively invokes the efficient *osworld_mcp_code.add_folder* tool twice to add the required directories to the workspace. It then correctly identifies the need for a GUI action and clicks the "I trust the authors" dialog to grant folder permissions, thereby successfully completing the end-to-end workflow. See the complete case study in Appendix D.

4 Related Work

Multimodal Agents for Computer Use. Building generalist multimodal agents in digital environments has long been a foundational yet challenging pursuit [2, 3, 27, 31, 43, 46, 48, 59]. Current CUAs generally follow two paradigms: multi-agent systems [1, 11, 19, 35, 54, 56], which decompose tasks across specialized modules, and end-to-end agentic models [21, 31, 43, 49, 51], which integrate planning and grounding within a unified policy. Most of these agents, however, still rely primarily on primitive GUI actions such as clicking, typing, and scrolling, making long-horizon tasks vulnerable to cascading errors and inefficient execution. Reinforcement learning has therefore been introduced to improve GUI agents, from mobile navigation settings [24, 25] to broader agentic GUI environments [16, 23, 45, 50, 55]. These studies demonstrate the promise of environment-driven

optimization, but their action spaces are largely restricted to GUI operations and often provide limited supervision for trajectory-level orchestration. Consequently, they do not directly address the hybrid action spaces that combine raw GUI interaction with structured tool invocation in realistic computer-use environments [28, 58]. ToolCUA builds on this direction by using a staged training paradigm to move beyond GUI-only control and optimize complete GUI-Tool execution paths.

Hybrid GUI-Tool Actions for CUAs. A broad line of research equips LLMs with tools [29, 30], enabling progress in autonomous coding [9, 10, 41], deep research [38, 60], and multimodal visual search [13, 22, 36, 39, 66]. In CUA scenarios, structured tools can reduce repetitive low-level GUI operations and improve efficiency when reliable interfaces are available. Recent studies [14, 53] have begun to explore this direction by introducing various mcp tools for hybrid-action evaluation, while several GUI-Tool systems [11, 35, 40, 52, 54, 57] connect GUI agents with APIs, SDKs, external executors, or multi-agent tool routers. Nevertheless, these efforts still fail to solve two critical challenges. First, high-quality interleaved GUI-Tool trajectories remain scarce because existing pipelines often rely on costly tool construction, environment engineering, or closed-source data collection. Second, existing methods provide limited guidance for trajectory-level selection. ToolCUA addresses these limitations by synthesizing interleaved GUI-Tool trajectories and adopting a staged training paradigm. More recent work also explores hybrid computer-use settings. RecreationWorld [5] provides scalable and verifiable environments for GUI exploration and software development, WeaveBench [18] benchmarks long-horizon workflows combining GUI and CLI/code operations, and HybridCUA [7] studies learning to orchestrate GUI and CLI actions through supervised fine-tuning and reinforcement learning.

5 Conclusion

In this work, we presented **ToolCUA**, an end-to-end computer-use agent that learns to orchestrate GUI actions and tool calls through optimal GUI-Tool path selection. We show that simply exposing agents to both GUI actions and tool calls is insufficient, as current models often overuse tools or remain overly GUI-centric, leading to inefficient and brittle execution trajectories. To address this challenge, ToolCUA first scales interleaved GUI-Tool trajectories from existing GUI data without manually constructing tools, then applies Tool-Bootstrapped GUI RFT to acquire tool-calling knowledge and calibrate critical switching decisions. Finally, Online Agentic RL with a Tool-Efficient Path Reward optimizes ToolCUA in a GUI-Tool environment, encouraging appropriate tool use and shorter execution paths. Experiments on OSWorld-MCP show that ToolCUA achieves 46.85% accuracy, a relative improvement of approximately 66% over the baseline, and consistently improves over pure GUI action settings. Together with the cross-task, cross-platform, and hybrid Windows transfer results and the ablation study, these findings suggest that training with a hybrid GUI-Tool action space provides a high-fidelity training paradigm for robust real-world digital agents.

Acknowledgements

This work was funded by Alibaba Token Hub, Alibaba Group, which provided API access, GPU resources, and substantial collaboration and research guidance. We also thank the researchers at Fudan University for their valuable guidance throughout this work.

References

[1] Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

[2] Anthropic. Claude opus 4.5, 2026. URL https://www.anthropic.com/news/claude-opus-4-5. Accessed: 2026-04-20.

[3] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report, 2025. *URL https://arxiv. org/abs/2511.21631*, 2025.

[4] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[5] Shuai Bai, Jiayong Deng, Sicheng Fan, Yikun Fu, Chang Gao, Xuhao Hu, Mianqiu Huang, Yizhen Jiang, Yuheng Jing, Dehui Kong, Keliang Li, Ning Li, Wanli Li, Dayiheng Liu, Dunjie Lu, Changwei Luo, Que Shen, Zheyuan Wang, Zijian Wang, Jie Wu, Gao Wu, Zhihui Xie, Rui Xie, Haiyang Xu, An Yang, Jiakang Yuan, Yanming Zhang, Jiajun Zhang, Xi Zhang, Zhenru Zhang, Zhuo Zhen, Mingkang Zhu, and Bowen Zhou. RecreationWorld: Scalable and verifiable environments for hybrid computer-use agents. *arXiv preprint arXiv:2609.22000*, 2026. URL https://arxiv.org/abs/2609.22000.

[6] 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.

[7] Tongbo Chen, Junbo Niu, Zhengxi Lu, Niu Lian, Fei Tang, Yuchen Yan, Yike Hong, Yong Du, Yizhou Liu, Bofan Chen, and Yongliang Shen. HybridCUA: Learning to orchestrate GUI and CLI for computer-use agents. *arXiv preprint arXiv:2609.38008*, 2026. URL https://arxiv.org/abs/2609.38008.

[8] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 24185–24198, 2024.

[9] Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025.

[10] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

[11] Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*, 2025.

[12] Google DeepMind. Gemini: The most capable and general model we've built, 2026. URL https://deepmind.google/models/gemini/pro/. Accessed: 2026-04-28.

[13] Jack Hong, Chenxiao Zhao, ChengLin Zhu, Weiheng Lu, Guohai Xu, and Xing Yu. Deepeyesv2: Toward agentic multimodal model. *arXiv preprint arXiv:2511.05271*, 2025.

[14] Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. *arXiv preprint arXiv:2510.24563*, 2025.

[15] Xiangru Jian, Shravan Nayak, Kevin Qinghong Lin, Aarash Feizi, Kaixin Li, Patrice Bechard, Spandana Gella, and Sai Rajeswar. Cua-suite: Massive human-annotated video demonstrations for computer-use agents. *arXiv preprint arXiv:2603.24440*, 2026.

[16] Pengxiang Li, Zechen Hu, Zirui Shang, Jingrong Wu, Yang Liu, Hui Liu, Zhi Gao, Chenrui Shi, Bofei Zhang, Zihao Zhang, et al. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*, 2025.

[17] Shilong Li, Xingyuan Bu, Wenjie Wang, Jiaheng Liu, Jun Dong, Haoyang He, Hao Lu, Haozhe Zhang, Chenchen Jing, Zhen Li, et al. Mm-browsecomp: A comprehensive benchmark for multimodal browsing agents. *arXiv preprint arXiv:2508.13186*, 2025.

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

[19] Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, et al. Pc-agent: A hierarchical multi-agent collaboration framework for complex task automation on pc. *arXiv preprint arXiv:2502.14282*, 2025.

[20] Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, et al. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*, 2024.

[21] Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, et al. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*, 2025.

[22] Ziyu Liu, Zeyi Sun, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. Visual-rft: Visual reinforcement fine-tuning. *arXiv preprint arXiv:2503.01785*, 2025.

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

[24] Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 1(2):3, 2025.

[25] Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

[26] Jian Mu, Chaoyun Zhang, Chiming Ni, Lu Wang, Bo Qiao, Kartik Mathur, Qianhui Wu, Yuhang Xie, Xiaojun Ma, Mengyu Zhou, et al. Gui-360: A comprehensive dataset and benchmark for computer-using agents. *To appear*, 2025.

[27] OpenAI. Introducing operator, 2026. URL https://openai.com/index/introducing-operator/. Accessed: 2026-04-20.

[28] OpenClaw. Openclaw, 2026. URL https://github.com/openclaw/openclaw. Accessed: 2026-04-20.

[29] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis, 2023. *URL https://arxiv. org/abs/2305.15334*, 2023.

[30] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[31] 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.

[32] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

[34] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

[35] Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, et al. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*, 2025.

[36] Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, et al. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. *arXiv preprint arXiv:2506.23918*, 2025.

[37] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

[38] Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

[39] Chaoyang Wang, Kaituo Feng, Dongyang Chen, Zhongyu Wang, Zhixun Li, Sicheng Gao, Meng Meng, Xu Zhou, Manyuan Zhang, Yuzhang Shang, et al. Adatooler-v: Adaptive tool-use for images and videos. *arXiv preprint arXiv:2512.16918*, 2025.

[40] 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.

[41] Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently. *arXiv preprint arXiv:2504.14870*, 2025.

[42] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024.

[43] 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.

[44] Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026.

[45] Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. Rlanything: Forge environment, policy, and reward model in completely dynamic rl system. *arXiv preprint arXiv:2602.02488*, 2026.

[46] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025.

[47] Qianshan Wei, Yishan Yang, Siyi Wang, Jinglin Chen, Binyu Wang, Jiaming Wang, Shuang Chen, Zechen Li, Yang Shi, Yuqi Tang, et al. Agentic-mme: What agentic capability really brings to multimodal intelligence? *arXiv preprint arXiv:2604.03016*, 2026.

[48] 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.

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

[50] Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. Mobilerl: Online agentic reinforcement learning for mobile gui agents. *arXiv preprint arXiv:2509.18119*, 2025.

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

[52] 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.

[53] Yunhe Yan, Shihe Wang, Jiajun Du, Yexuan Yang, Yuxuan Shan, Qichen Qiu, Xianqing Jia, Xinge Wang, Xin Yuan, Xu Han, et al. Mcpworld: A unified benchmarking testbed for api, gui, and hybrid computer use agents. *arXiv preprint arXiv:2506.07672*, 2025.

[54] Bowen Yang, Kaiming Jin, Zhenyu Wu, Zhaoyang Liu, Qiushi Sun, Zehao Li, JingJing Xie, Zhoumianze Liu, Fangzhi Xu, Kanzhi Cheng, et al. Os-symphony: A holistic framework for robust and generalist computer-using agent. *arXiv preprint arXiv:2601.07779*, 2026.

[55] Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, et al. Zerogui: Automating online gui learning at zero human cost. *arXiv preprint arXiv:2505.23762*, 2025.

[56] Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, et al. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*, 2025.

[57] 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.

[58] Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Toward trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

[59] 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: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

[60] Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

[61] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[62] Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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

[64] Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence. *arXiv preprint arXiv:2503.11069*, 2025.

[65] Chaoyun Zhang, Liqun Li, He Huang, Chiming Ni, Bo Qiao, Si Qin, Yu Kang, Minghua Ma, Qingwei Lin, Saravan Rajmohan, et al. Ufo^3: Weaving the digital agent galaxy. *arXiv preprint arXiv:2511.11332*, 2025.

[66] Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. Deepeyes: Incentivizing" thinking with images" via reinforcement learning. *arXiv preprint arXiv:2505.14362*, 2025.

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 clearly state the paper's contributions and scope.

2. **Limitations**

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

   Answer: [Yes]

   Justification: The limitations are discussed in the Appendix.

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: [Yes]

   Justification: The set of assumptions and a complete proof are provided.

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 provided the detailed reproduction information needed like datasets and training settings in Appendix C

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: [No]

   Justification: The core code and data are restricted by the company policy, we will try our best to open-source the resources.

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: We have provided the training and test details necessary to understand the results in Section 3 and Appendix C.

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: We have provided in Appendix C.5

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: We provide the compute resources details in Section 3.1.

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: The research conforms with the NeurIPS Code of Ethics in every respect.

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We provide the social impacts in Appendix B

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: [N/A]

    Justification: The paper exploring CUA automation on public datasets poses no such risks.

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: We have properly credited the used assets. All the datasets introduced in this paper are available for free to researchers for non-commercial use.

13. **New assets**

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

    Answer: [Yes]

    Justification: The assets will be documented alongside the open access to the code and data.

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: The paper does not involve crowdsourcing nor research with human subjects.

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 paper does not involve crowdsourcing nor research with human subjects.

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: The paper uses LLM for data synthesis described in Section 2.2.

Appendix

A Limitations and Future Work

Although ToolCUA demonstrates the effectiveness of learning from synthesized interleaved GUI-Tool trajectories, our synthesis-driven pipeline is still constrained by the state frames and domain coverage of the original GUI-only trajectories. As a result, the diversity and quality of the synthesized hybrid trajectories are coupled with the breadth, fidelity, and task distribution of the source demonstrations. Our tool-scaling process also depends on the capability of the general model used for synthesis; in our internal trials, replacing stronger proprietary models with Qwen3.5-Plus led to noticeably lower generation efficiency and trajectory quality. Moreover, the synthesized tools are not tied to a specific concrete implementation, which makes them scalable and potentially more generalizable, but also means that real-world execution still depends on the maturity of available tools and the way tool feedback is organized for the CUA. Finally, due to the scarcity of open-source GUI-Tool coordination benchmarks for computer-use agents, our main performance evaluation is primarily conducted on OSWorld-MCP, leaving broader benchmark coverage as an important limitation.

Future work should further explore hybrid GUI-Tool action spaces across broader platforms, including desktop, mobile, and web environments, where the balance between atomic GUI operations and high-level tools may vary substantially across interface structures and task types. Another promising direction is to reduce the dependence of agentic RL on heavy sandbox infrastructure by building lighter, more diverse, and more robust environments with hybrid GUI-Tool actions. We are also interested in asynchronous RL frameworks that decouple training and inference-time rollout, which may improve the scalability and stability of long-horizon policy optimization for computer-use agents.

B Broader Impact and Ethics Statement

ToolCUA aims to improve real-world digital automation by enabling computer-use agents to coordinate GUI actions and tool calls more efficiently, which may benefit productivity, accessibility, and repetitive workflow assistance. At the same time, more capable CUAs introduce potential risks, including unauthorized operation, accidental modification of user data, privacy leakage from desktop observations, and misuse for automating harmful or deceptive online activities. Our work focuses on benchmarked sandbox environments and does not grant the agent uncontrolled access to personal accounts, sensitive files, or external services. For real-world deployment, we believe such systems should require explicit user consent, transparent action logging, permission boundaries, and human confirmation for high-impact operations.

C Implementation Details

C.1 Preliminary Study: Optimal Path Confusion in Hybrid Action Spaces

To examine whether current CUAs can identify the optimal GUI-Tool path under tool-conditioned contexts, we conduct a diagnostic comparison between pure GUI execution and hybrid GUI-Tool execution across multiple agents, as reported in Table 1. All models in the hybrid setting are given access to the same tool interface and tool documentation through the system prompt. Accuracy is reported over the full feasible task set. ACS reports the average number of trajectory steps over the full feasible task set, reflecting execution efficiency. In Table 1, Tool-Calls reports the average number of tool calls among successful trajectories in the Tool-Beneficial subset; this conditional statistic directly measures tool-use intensity when tools are useful and the task is completed successfully.

Experimental Settings. For the results reported in Table 1, we follow two evaluation protocols corresponding to the two action-space settings. For the GUI-only setting, we directly use the official OSWorld-released verified trajectories [48] for EvoCUA-32B and the Claude baselines, set the maximum horizon to 50 steps, and retain only the feasible tasks. For Qwen3VL-8B-Instruct and Qwen3VL-235B-A22B-Thinking, we conduct the evaluation ourselves, aligned with the official implementations. For clarity, Qwen3-VL-8B-Instruct is abbreviated as Qwen3VL-8B, and Qwen3VL-235B-A22B-Thinking is abbreviated as Qwen3VL-235B throughout Table 1. For the "GUI+Tool" setting, we evaluate the same models directly on OSWorld-MCP, ensuring consistency with the

official prompts and message construction protocol. We also use the same maximum horizon of 50 steps and report results only on the feasible task subset. See more metric details in Appendix C.5.

The results in Table 1 reveal a counter-intuitive finding: simply giving a strong model access to both GUI actions and tools does not reliably improve performance. Instead, the hybrid action space often *confuses* the agent, causing it to deviate from the most effective execution path. More concretely, Table 1 exposes two representative failure modes that align with the intuition illustrated in Figure 2: once both action spaces are available, the agent stands at a "forked road" and often fails to choose the right branch.

Failure Mode I: tool underuse. Some models remain overly GUI-centric even when a short tool call is available. Qwen3VL-8B is the clearest example: even among successful Tool-Beneficial trajectories, it invokes only 0.01 tools on average, indicating that it almost never leaves the GUI branch. Yet this conservative behavior does not preserve performance: its accuracy drops from 29.0% to 28.2%, while ACS also slightly increases from 19.2 to 19.3. This suggests that the model is unable to recognize the decision boundary where switching to a structured tool would shorten the path and reduce accumulated GUI errors.

Failure Mode II: tool overuse. At the other extreme, stronger or larger models may invoke tools aggressively, but frequent tool usage alone does not translate into better task completion. For example, Claude-4-Sonnet invokes 4.30 tools on average among successful Tool-Beneficial trajectories, yet its overall accuracy drops from 47.7% to 43.5% after tools are added. Qwen3VL-235B, EvoCUA-32B, and Claude-4.5-Sonnet exhibit the same failure from the outcome side: access to tools shortens some trajectories but reduces their overall accuracy by 3.0, 12.1, and 13.5 percentage points, respectively. These cases indicate that the agent is not learning "use tools more," but rather needs to learn *when* tool usage is appropriate. Over-switching can prematurely abandon necessary GUI grounding, call tools before the required context is established, or commit the trajectory to a brittle tool-heavy path that is shorter but less reliable.

Taken together, Table 1 demonstrates that the core challenge of hybrid GUI-Tool agents is optimal GUI-Tool path selection rather than raw action-space expansion. The issue is not whether the model can recognize that a tool exists, but whether it can determine if the current state calls for continued GUI grounding, an immediate tool invocation, or a later switch after additional GUI setup. In other words, hybrid execution introduces a trajectory-level decision problem: a locally plausible action may still lead to a globally inferior path. This is exactly the confusion abstracted by Figure 2, where the agent must choose between the GUI path and the tool path without a reliable internal policy for deciding which route is actually better. In contrast, ToolCUA is the only model in Table 1 that benefits from the hybrid action space: its accuracy improves from 42.9% to 46.8%, while ACS drops from 19.4 to 14.9 with 2.11 tool calls on average among successful Tool-Beneficial trajectories. This result suggests that effective hybrid execution does not require excessive tool usage, but a selective switching policy that calls tools only at beneficial points along the trajectory. Therefore, these preliminary results motivate our training paradigm: ToolCUA is designed to internalize optimal GUI-Tool path selection from interleaved supervision and trajectory-level reinforcement, rather than relying on zero-shot prompting to resolve the fork on its own.

C.2 Detailed Analysis of Adaptive Tool Usage

The Tool-Calls column in Table 1 focuses on successful Tool-Beneficial trajectories. To explain this conditional statistic and distinguish selective invocation from indiscriminate tool use, Table 5 further separates tool calls by task type and outcome. ToolCUA concentrates its calls on successful Tool-Beneficial trajectories, averaging 2.11 calls, while using only 0.12 calls on failed Tool-Beneficial trajectories and almost none on Non-Tool-Beneficial tasks. By contrast, Claude-4-Sonnet continues to invoke tools frequently in failed and Non-Tool-Beneficial trajectories. This pattern shows that ToolCUA's tool use is associated with successful, appropriate execution rather than uniformly increased calling frequency.

Averaged over all 333 feasible tasks, ToolCUA makes 0.74 tool calls per task; this pooled value is reduced by the 95 Non-Tool-Beneficial tasks, where abstention is the desired behavior. The conditional breakdown therefore provides a more direct view of orchestration quality. For an empirical efficiency reference, we pool all evaluated models, retain the shortest successful tool-invoking trajectory for each covered task, and average over Tool-Beneficial tasks. This reference

Table 5: Average tool calls separated by task type and outcome on OSWorld-MCP.

Model	Tool-Beneficial Set (238)		Non-Tool-Beneficial Set (95)		ACS ↓
	Successful	Failed	Successful	Failed	
Kimi-K2.5	1.90	1.33	0.90	0.80	19.8
Claude-4-Sonnet	4.30	5.40	2.90	3.80	19.8
Qwen3VL-8B	0.01	0.00	0.00	0.00	19.3
ToolCUA-8B	**2.11**	0.12	0.02	0.00	**14.9**

covers the union of 249 tasks and averages 2.65 tool calls; it should not be interpreted as a ground-truth optimum. ToolCUA's 2.11 calls on successful Tool-Beneficial trajectories are close to this empirical reference, while its near-zero usage elsewhere indicates that it neither remains GUI-only nor invokes tools indiscriminately.

C.3 Data Statistics of the Scaling Pipeline

Our Offline Interleaved GUI-Tool Trajectory Scaling Pipeline constructs a robust foundation by aggregating diverse pure GUI interaction trajectories from OpenCUA [43] and ScaleCUA [21]. To further augment the data, we leverage diverse powerful MLLMs to execute sandbox rollouts across internal tasks, resulting in 1,200 curated, high-quality trajectories post-filtering. As detailed in Table 7, this combined source corpus comprises 10,000 trajectories and 192,000 raw GUI steps, with OpenCUA serving as the primary source (8,500 trajectories). From this base, our pipeline synthesizes 10,000 interleaved GUI-Tool trajectories, yielding 180k high-quality steps for the warmup Supervised Fine-Tuning (SFT) stage. Furthermore, we extract a specialized subset of 5k critical switching steps ($\mathcal{D}_{critical}$) to facilitate offline single-turn Reinforcement Learning (RL).

The synthesis process generates a diverse and hierarchically structured tool inventory. As outlined in Table 8, the synthetic dataset contains 4,350 unique tools spanning multiple abstraction levels. It is primarily anchored by fine-grained and mid-grained operations, complemented by coarse-grained skills. To emulate realistic decision-making complexity, each trajectory provides an average tool pool of 19.75 candidate tools, from which the agent executes an average of 7.89 tools. Overall, these metrics show that our synthesized trajectories cover diverse tool types, large candidate pools, and substantial executed tool calls.

Human Verification of Next-State Grounding. Overall, 82.5% of synthesized tool steps pass the automatic next-state grounding filter. We further audit both accepted and rejected examples manually. For each sampled tool step, an annotator examines the tool description, schema, call arguments, and corresponding next-state screenshot, assumes normal execution according to the tool specification, and judges whether the predicted effect is aligned with the recorded GUI state. As shown in Table 6, 93.0% of accepted steps are judged aligned, whereas only 28.0% of rejected steps are judged aligned. This contrast indicates that the filter retains state-consistent synthetic steps while removing most low-consistency candidates.

Table 6: Human audit of the next-state grounding filter. State alignment denotes the percentage of sampled tool steps whose predicted effects are judged consistent with the recorded next-state screenshot.

Tool-step set	Sampled	State alignment ↑
Accepted by next-state grounding	100	93.0%
Rejected by next-state grounding	50	28.0%

Figure 7 visualizes the synthesized tools in a projected space, where each point corresponds to one tool node and marker shapes denote granularity tiers. Major application categories such as `LibreOffice`, `Chrome`, and `VSCode` occupy broad and distinguishable regions, indicating that the synthesized inventory does not collapse into a narrow set of routines. Within each region, the coexistence of fine-grained, mid-grained, and coarse-grained tools shows that the learned tool space

is not restricted to a single abstraction level. The visualization therefore supports the two intended properties of our bottom-up pipeline: diverse tool functionality and clear multi-granular organization.

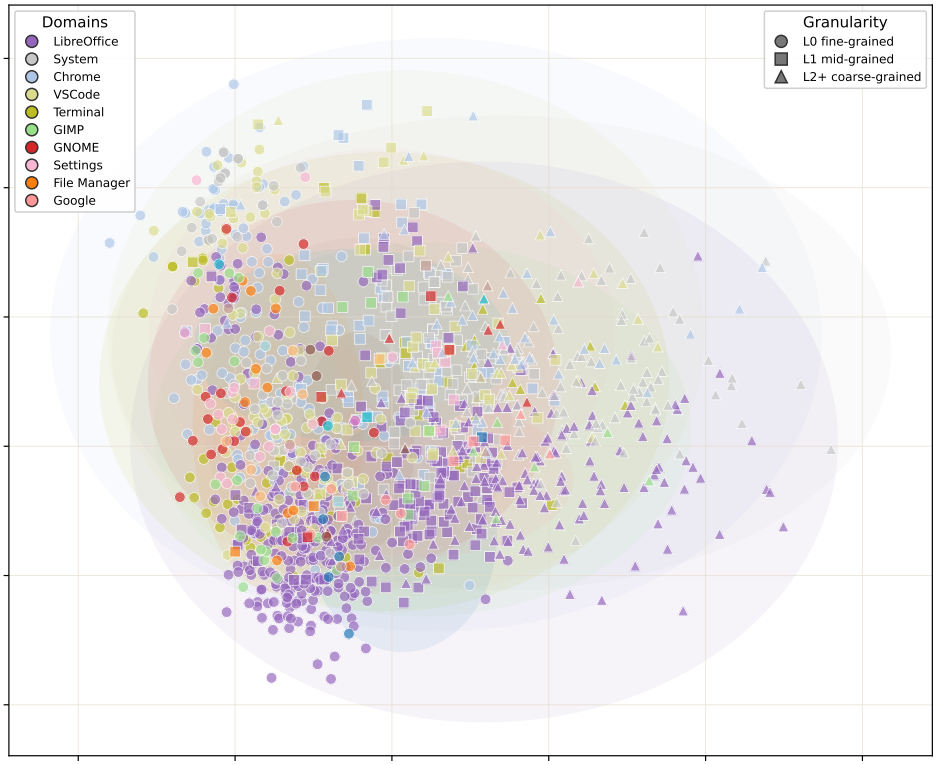


Figure 7: Visualization of the synthesized tools in a projected action space, where each point corresponds to one tool node, colors denote the application taxonomy, and marker shapes denote granularity tiers.

Table 7: Data distribution of pure GUI trajectories used by our synthesis pipeline.

Data Source	**Trajs**	**Steps**
OpenCUA [43]	8500	110k
ScaleCUA [21]	300	20k
Sandbox collected	1200	62k
Source total	**10k**	**192k**
critical steps ($\mathcal{D}_{critical}$)	-	5k

Table 8: Statistics of our interleaved GUI-Tool data.

Statistics	**Count**
Unique Tools	4,350
fine-grained	2,000
mid-grained	1,900
coarse-grained	450
avg. tool-pool size per traj.	19.75
avg. executed tools per traj.	7.89

C.4 Training Details

Tool-Bootstrapped GUI RFT. In our staged training paradigm, we adopt Qwen3-VL-8B-Instruct [3] as our base model M. The warmup SFT is conducted for 3 epochs using full-parameter fine-tuning of both the vision tower and the LLM backbone on a cluster of 8×8 GPUs, resulting in the intermediate model $\mathcal{M}_{sft}$. For the offline single-turn RL phase, we use a rollout group size of 32, a learning rate of 1×10^{-6}, and a training batch size of 128 to produce $\mathcal{M}_{rft}$.

Online Agentic RL Training. During the subsequent online agentic RL stage, we set hyperparameters $\lambda = 0.4$, $\beta = 0.2$, and $S_{max} = 30$ to define the maximum execution horizon. The training configuration for this stage includes a rollout size of 32 per group, a learning rate of 1×10^{-6}, and a

training batch size of 32 to obtain **ToolCUA**. **Tool-Beneficial Labeling.** For the Tool Appropriateness Reward Term, we assign a task-level label $t_b \in \{1, -1\}$ through rollout-based comparison followed by human verification. A frontier MLLM performs rollouts for each task with and without mcp tools using pass@32. We label a task as tool-beneficial when the with-tool success rate clearly exceeds the GUI-only success rate, and as non-tool-beneficial in the opposite case. The OSWorld-MCP annotations are used only as a cross-reference rather than as supervision for the reward. Human annotators then verify whether the available tools genuinely simplify, accelerate, or improve the reliability of each task. In a sampled audit, rollout-based labels agree with human judgments on 91.7% of 60 tool-beneficial tasks and 80.0% of 40 non-tool-beneficial tasks. Cases without a clear success-rate margin are treated as ambiguous and excluded from the online RL pool; these account for fewer than 5% of the manually checked tasks. Since R_{tool} is a soft shaping term with weight $\lambda = 0.4$ and is activated only for successful trajectories, residual label noise does not override the task-success signal. **Online RL Task Pool.** The online RL pool contains 339 single-domain tasks and excludes the entire *multi_apps* domain, which is reserved for held-out evaluation. Of these tasks, 141 are drawn directly from OSWorld [48], 124 are rewritten and augmented variants, and 74 are newly scaled tasks constructed following RLAnything [45]. The rewritten tasks vary the original phrasing, user intent, and constraints, while the scaled tasks further alter task queries and difficulty and use newly constructed evaluators. We retain source-task identifiers throughout construction and perform task-ID-level deduplication when organizing the training split.

CUA Agentic RL in a GUI-Tool Environment. To support large-scale online exploration, we implement a decoupled training-inference infrastructure using the verl [34] framework, where policy optimization occurs on a GPU cluster while rollouts are executed on distributed ECS servers. The environment sandbox is built upon OSWorld [48] QEMU images, incorporating mcp tool designs from OSWorld-MCP [14] and AutoGLM [20]. We further optimize the tool-calling interface by designing an agent-readable return format that provides concise, semantically dense feedback to reduce token overhead and improve grounding accuracy. The online RL phase uses around 250 independent Docker instances for environment rollouts, together with 8×8 GPUs for policy training and 4×8 GPUs for dedicated inference serving.

C.5 Benchmark Evaluation

We evaluate ToolCUA primarily on OSWorld-MCP [14], which extends OSWorld with tool actions and therefore directly measures hybrid GUI-Tool execution. Following the benchmark protocol, we report three metrics: task accuracy, Tool Invocation Rate (TIR), and Average Completion Steps (ACS). To reduce sandbox stochasticity, we report average@3 results for OSWorld-MCP and set the maximum number of execution steps to 50 for each task.

Task Accuracy. Task accuracy is the primary success metric and measures whether an agent completes the target instruction according to the benchmark evaluator. In OSWorld-MCP, this metric reflects not only visual grounding and GUI interaction ability, but also whether the agent can use the provided tools to reach the correct final state.

Tool Invocation Rate (TIR). OSWorld-MCP further separates tasks into Tool-Beneficial Tasks and Non-Tool-Beneficial Tasks, allowing us to measure whether an agent invokes tools in appropriate contexts rather than merely calling tools more often. Let N_t be the total number of Tool-Beneficial Tasks, and n_t the number of such tasks in which the agent invoked a tool and successfully completed the task during evaluation. Let N_g be the total number of Non-Tool-Beneficial Tasks, and n_g the number of such tasks in which the agent did not invoke a tool and successfully completed the task. We define TIR as:

$$\text{TIR} = (n_t + n_g)/(N_t + N_g) \tag{5}$$

TIR therefore captures whether the agent aligns its tool usage with task-level tool utility, including both using tools when they are beneficial and avoiding them when GUI actions are more appropriate.

Average Completion Steps (ACS). ACS measures the average number of environment interaction steps used by an agent across tasks. For N tasks, if the number of execution steps for task i is S_i, the

Average Completion Steps is computed as:

$$\text{ACS} = \sum_{i=1}^{N} S_i / N \tag{6}$$

ACS reflects execution efficiency: agents that identify shorter tool-conditioned paths and avoid redundant GUI operations generally require fewer steps to complete the same task.

To assess cross-platform transfer, we evaluate ToolCUA in both the GUI-only WindowsAgentArena (WAA) setting and a setting with mcp tool support. We port 117 mcp tools covering five domains—`file_explorer`, `libreoffice_calc`, `libreoffice_writer`, `vlc`, and `vs_code`—which correspond to 107 evaluation tasks. Applications whose tool support could not be stabilized, including `microsoft_paint`, `msedge`, and `notepad`, are excluded from this matched subset. In the setting with mcp tool support, the agent is jointly provided with the standard GUI action space and the available ported mcp tool schemas, following the same message construction and tool-calling interface used in our Linux evaluation. For a controlled comparison, results for the GUI-only setting and the setting with mcp tool support in Table 4 are recomputed on the same 107 tasks. We set `max_steps`=50 and report average@3 accuracy. Tool calls are averaged over successful trajectories in the setting with mcp tool support.

C.6 Ablation Details

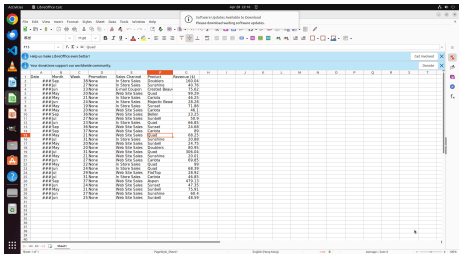
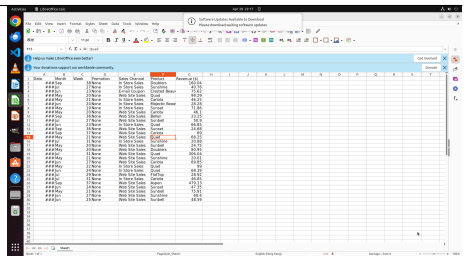
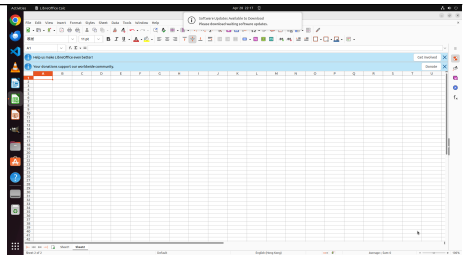
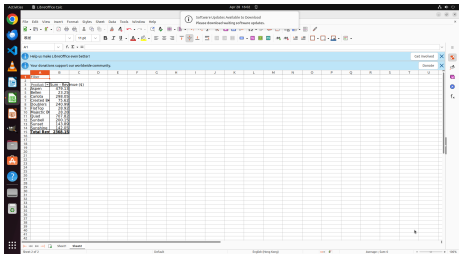
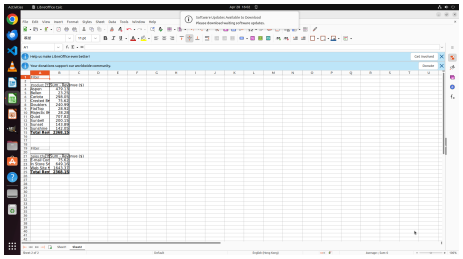
We conduct three ablation studies to analyze the contribution of each training component and the effect of the hybrid GUI-Tool action space. The first two studies, shown in Figure 6, examine the effectiveness of our staged training design. In the first setting, denoted as "w/o Interleaved data", we remove the offline interleaved GUI-Tool data bootstrapping stage and directly apply online agentic RL with the proposed Tool-Efficient Path Reward to the Qwen3-VL-8B-Instruct baseline. This setting tests whether online exploration alone can acquire tool-calling knowledge and GUI-Tool switching ability from sparse trajectory-level feedback. In the second setting, the model is first initialized with Tool-Bootstrapped GUI RFT, and then optimized with a vanilla GRPO-based agentic RL objective that only uses the standard accuracy reward R_{acc} and format reward R_{fmt}. This variant removes the Tool Appropriateness Reward Term and Path Efficiency Reward Term, isolating the effect of the Tool-Efficient Path Reward after the model has already acquired basic tool-calling knowledge and local switching capability.

The third ablation, reported in Table 3, compares training in the hybrid GUI-Tool action space with training in a pure GUI action space. For the pure-GUI setting, we fine-tune the Qwen3-VL-8B-Instruct baseline using a GUI-only dataset of comparable scale to the warmup SFT data, and then perform online agentic RL within the pure GUI action space. Since tool actions are unavailable in this environment, the reward consists only of $R_{acc} + R_{fmt} + R_{length}$, where the Path Efficiency Reward Term still encourages efficient execution but cannot reward tool-appropriate decisions. The final evaluation is also conducted under the pure GUI action space, allowing us to directly compare whether hybrid GUI-Tool training provides a higher-fidelity optimization setting than GUI-only training.

For all online agentic RL ablations, we follow the evaluation protocol in Section C.5 and report average@3 results. The maximum rollout horizon during training is set to `max_steps`=30. Due to the computational cost of sandbox-based online RL under our decoupled training-inference infrastructure, each run is trained for roughly 25 optimization steps. With a unified training batch size of 32 and dynamic filtering enabled, each ablation uses approximately 1,200 effective training samples. Each run consumes approximately 8×8 GPUs together with distributed ECS sandbox workers for about six days.

D More Case Studies

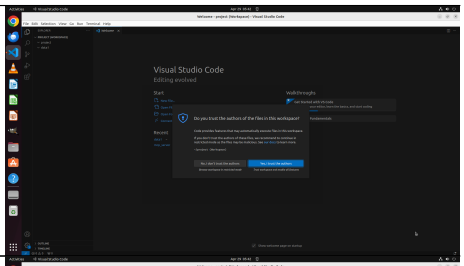
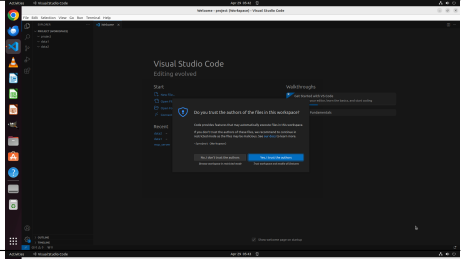
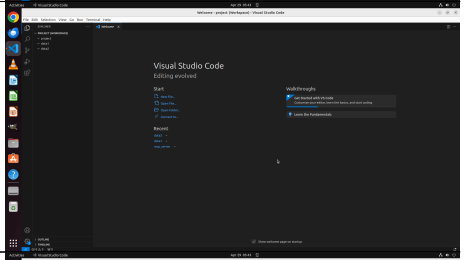
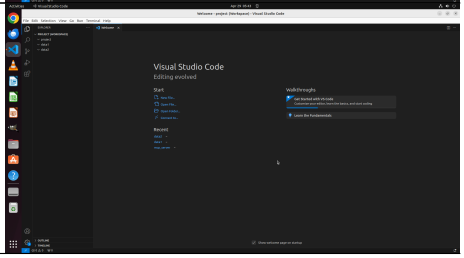
Table 9: Case Study: LibreOffice Calc (ID: 535364ea-05bd-46ea-9937-9f55c68507e8).

Agent Prediction	Tool-Calling Response	Screenshot
Step 2: Action: Get the workbook information to identify the file path and sheet details. <tool_call> {\"name\": \"osworld_mcp_libreoffice_calc.get_workbook_info\", \"arguments\": {}} </tool_call>	<tool_response> {"type": "text", "text": "{\"success\": true, \"result\": {\"file_path\": \"file:///home/user/SummerSales.xlsx\", \"file_title\": \"SummerSales.xlsx\", \"sheets\": [{\"name\": \"Sheet1\", \"index\": 0, \"visible\": true, \"row_count\": 30, \"column_count\": 7}], \"active_sheet\": \"Sheet1\", \"sheet_count\": 1}, \"error_message\": null}"} </tool_response>	
Step 3: Action: Get content information of Sheet1 to understand the data structure including headers and data ranges. <tool_call> {\"name\": \"osworld_mcp_libreoffice_calc.env_info\", \"arguments\": {\"sheet_name\": \"Sheet1\"}} </tool_call>	<tool_response> {"type": "text", "text": "{\"success\": true, \"result\": {\"name\": \"Sheet1\", \"data\": \" | A| B| C | D | E | F | G |\\n--|-----...}, \"error_message\": null}"} </tool_response>	
Step 4: Action: Create a new sheet named \"Sheet2\" to serve as the destination for the pivot tables. <tool_call> {\"name\": \"osworld_mcp_libreoffice_calc.switch_active_sheet\", \"arguments\": {\"sheet_name\": \"Sheet2\"}} </tool_call>	<tool_response> {"type": "text", "text": "{\"success\": true, \"result\": \"Success\", \"error_message\": null}"} </tool_response>	
Step 5: Action: Create the first pivot table named 'PivotTable1' in Sheet2 to show total revenue for each product. <tool_call> {"name": "osworld_mcp_libreoffice_calc.create_pivot_table", "arguments": {"source_sheet": "Sheet1", "table_name": "PivotTable1", "row_fields": ["F"], "value_fields": ["G"], "aggregation_function": "sum", "target_cell": "A1"}} </tool_call>	<tool_response> {"success": true, "result": "Success", "error_message": null} </tool_response>	
Step 6: Action: Create the second pivot table named 'PivotTable2' in Sheet2 to show total revenue for each sales channel. <tool_call> {"name": "osworld_mcp_libreoffice_calc.create_pivot_table", "arguments": {"source_sheet": "Sheet1", "table_name": "PivotTable2", "row_fields": ["E"], "value_fields": ["G"], "aggregation_function": "sum", "target_cell": "A19"}} </tool_call>	<tool_response> {"success": true, "result": "Success", "error_message": null} </tool_response>	

Analysis: This case examines a LibreOffice Calc task that requires two pivot tables in a new sheet named Sheet2, showing total revenue by product and by sales channel. Tool calls allow the agent to

inspect workbook metadata and sheet contents before directly invoking `create_pivot_table` with the fields obtained from the environment information.

Table 10: Case Study: VS Code (ID: 6ed0a554-cbee-4b44-84ea-fd6c042f4fe1).

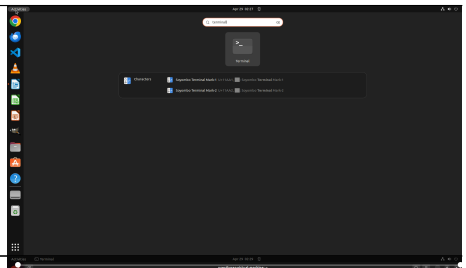
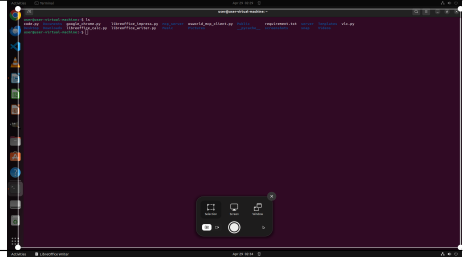
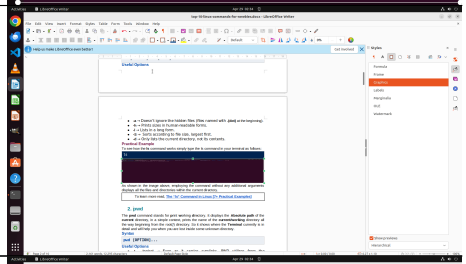
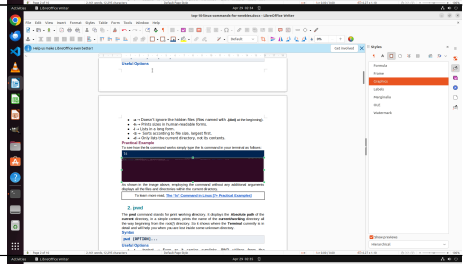
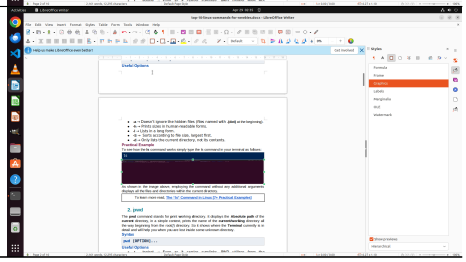
Agent Prediction	Tool-Calling Response	Screenshot
Step 1: Action: Add the folder '/home/user/data1' to the current workspace in VSCode. <tool_call> {\"name\": \"osworld_mcp_code.add_folder\", \"arguments\": {\"folder\": \"/home/user/data1\"}} </tool_call>"	<tool_response> {"type": "text", "text": "{\"success\": true, \"result\": \"Successfully added folder\", \"error_message\": null}"} </tool_response>	
Step 2: Action: Add the folder '/home/user/data2' to the current workspace in VSCode. <tool_call> {\"name\": \"osworld_mcp_code.add_folder\", \"arguments\": {\"folder\": \"/home/user/data2\"}} </tool_call>"	<tool_response> {"type": "text", "text": "{\"success\": true, \"result\": \"Successfully added folder\", \"error_message\": null}"} </tool_response>	
Step 3: Action: Click on the "Yes, I trust the authors" button in the trust dialog popup. <tool_call> {\"name\": \"computer_use\", \"arguments\": {\"action\": \"click\", \"coordinate\": [596, 627]}} </tool_call>	## Not available	
Step 4: Action: Terminate the task as successful since the goal has been accomplished. <tool_call> {\"name\": \"computer_use\", \"arguments\": {\"action\": \"terminate\", \"status\": \"success\"}} </tool_call>	## Not available	

Analysis:

This case demonstrates a VS Code workspace-management task that adds `/home/user/data1` and `/home/user/data2` to the current workspace. During execution, the agent directly invokes the `add_folder` tool to add each folder to the current VS Code window. This step highlights the efficiency and determinism of tool calls for structured workspace operations. Subsequently, although the core objective had already been accomplished via the tool calls, VS Code presented a "Do you trust the authors?" confirmation dialog. Rather than stalling at this point, the agent switches to GUI interaction, clicks the "Yes, I trust the authors" button, and completes the remaining interface actions, ensuring that the workspace reaches a usable state.

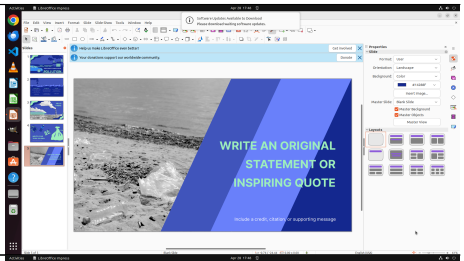
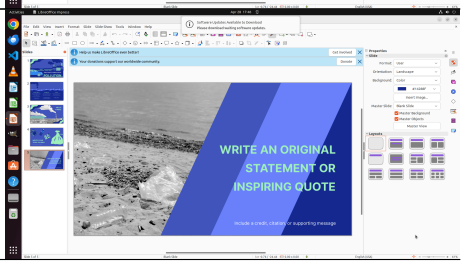
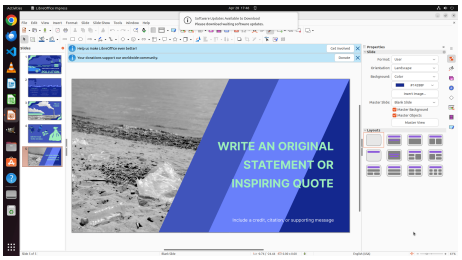
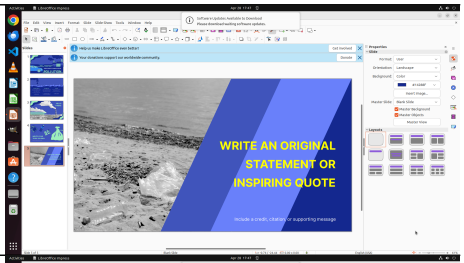
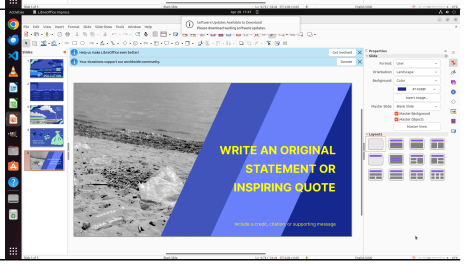
This case demonstrates that the system does not rely on a single interaction paradigm. Tool calls are well-suited for directly completing well-defined editing operations, while GUI actions can handle scenarios that tools cannot fully cover, such as pop-up dialogs, permission confirmations, and interface state finalization. By combining the two, the agent can both complete the primary task efficiently and handle post-execution interactions in a real desktop environment.

Table 11: Case Study: Multi-App Task (ID: 02ce9a50-7af2-47ed-8596-af0c230501f8).

Agent Prediction	Tool-Calling Response	Screenshot
Step 2: Action: typed 'terminal ' into the search box. <tool_call> {\"name\": \"computer_use\", \" arguments\": {\"action\": \" type\", \"text\": \"terminal\", \"clear\": 0, \"enter\": 0}} </tool_call>	## Not available	
Step 8: Action: dragged mouse from the left to the right and released. <tool_call> {\"name\": \"computer_use\", \" arguments\": {\"action\": \" mouse_move\", \"coordinate\": [24, 36]}} </tool_call>	## Not available	
Step 18: Action: Double-clicked on ls.png. <tool_call> {\"name\": \"computer_use\", \" arguments\": {\"action\": \" double_click\", \"coordinate\": [314, 251]}} </tool_call>	## Not available	
Step 19: Action: Save the document. <tool_call> {\"name\": \" osworld_mcp_libreoffice_writer. save\", \"arguments\": {}} </tool_call>	<tool_response> {"type": "text", "text": "{\" success\": true, \"result\": \" Success\", \"error_message\": null}"}, </tool_response>	
Step 20: Action: Terminate the task as successful. <tool_call> {\"name\": \"computer_use\", \" arguments\": {\"action\": \" terminate\", \"status\": \" success\"}} </tool_call>	## Not available	

Analysis: "I am currently utilizing LibreOffice Writer to compose a Linux tutorial, and I intend to display the outcomes generated by executing the İsc̈ommand in /home/user. Kindly execute this command and save the screenshot of the terminal as 'ls.png' on the Desktop." This case aims to generate visual evidence for a Linux tutorial: the agent needs to run ls in /home/user and save a terminal screenshot as ls.png on the Desktop. The key steps include opening the terminal, executing the command, capturing and saving the screenshot, checking the generated image, saving the LibreOffice document, and marking the task as complete. Each step contributes to turning the user's instruction into an actual file-based result.

Table 12: Case Study: LibreOffice Impress (ID: 57667013-ea97-417c-9dce-2713091e6e2a).

Agent Prediction	Tool-Calling Response	Screenshot
Step 1: Action: Navigate to slide 5 by clicking on its thumbnail in the slide navigation panel. <tool_call> {"name": " osworld_mcp_libreoffice_impress .go_to_slide", "arguments": {" slide_index": 5}} </tool_call>	<tool_response> {"success": true, "result": " Successfully navigated to slide 5", "error_message": null} </tool_response>	
Step2: Action: Get the slide count to understand the total number of slides in the presentation. <tool_call> {"name": " osworld_mcp_libreoffice_impress .get_slide_count", "arguments": {}} </tool_call>	<tool_response> {"success": true, "result": {" value": 5}, "error_message": null} </tool_response>	
Step 3: Action: Get the content of all slides to identify and understand the textboxes that need their font color changed to yellow. <tool_call> {"name": " osworld_mcp_libreoffice_impress .env_info", "arguments": {}} </tool_call>	<tool_response> {"success": true, "result": " Slide 1:\n- Box 0: \n- Box 1: \n- Box 2: \n- Box 3: \n- Box 4: \n- Box 5: \n- Box 7: \n- Box 8: \n- Box 9: \n- Box 10: \n- Box 12: \n\nSlide 2:\n- Box 0: \n- Box 1: \n\nSlide 3:\n- Box 0: \n- Box 1: HELLO\ nTHERE!\n- Box 2: \n- Box 4: I 'm Rain. Welcome to my education presentation.\ nFollow me at @reallygreatsite to learn more.\n- Box 5: \n- Box 6: \n\nSlide 4:\n- Box 0: \n- Box 1: \n- Box 2: \n- Box 3: \n\nSlide 5:\n- Box 0: \n- Box 3: WRITE AN ORIGINAL\ nSTATEMENT OR\nINSPIRING QUOTE \n- Box 4: Include a credit, citation, or supporting message\n\nCurrent Slide Index : 5", "error_message": null} </tool_response>	
Step 4: Action: Set the font color of the title text in slide 5 to yellow. <tool_call> {"name": " osworld_mcp_libreoffice_impress .set_text_color", "arguments": {"slide_index": 5, "box_index": 3, "color": "yellow"}} </tool_call>	<tool_response> {"success": true, "result": " Successfully set text color to yellow for textbox 3 on slide 5", "error_message": null} </tool_response>	
Step 5: Action: Terminate the task as successful since the goal has been accomplished. <tool_call> {"name": "computer_use", " arguments": {"action": " terminate", "status": "success "}} </tool_call>	## Not available	

Analysis: The instruction is to navigate to slide 5 and set the font color of all text boxes to yellow. This is a typical case in which tools improve execution efficiency. ToolCUA first calls `go_to_slide`, then uses `get_slide_count` and `env_info` to identify the target text box, and finally invokes `set_text_color(box_index=3)` to complete the task.

E Prompts Used in Data Pipeline

This section lists the prompts used in our Offline Interleaved GUI-Tool Trajectory Scaling Pipeline. The sequence is as follows: SCREENSHOT DESCRIPTION PROMPT summarizes raw UI states; TOOL GENERATION PROMPT proposes the tool library; FIX TOOL PROMPT repairs invalid tool definitions; JOINT GENERATION PROMPT generates step-by-step tool trajectories; PREDICT SCREENSHOT PROMPT and DESCRIBE AND LOCATE PROMPT support state prediction and screenshot grounding; and MERGE TREE PLANNING PROMPT together with BOTTOM UP MERGE PROMPT builds the bottom-up hierarchy by merging fine-grained steps into coarser tools.

Prompts Used in Data Pipeline

SCREENSHOT DESCRIPTION PROMPT

```
Describe this desktop screenshot in English.
Include:
1. Which application, window, or web page is visible
2. The main interactive UI elements that are visible, such as buttons, menus, tabs, tables, forms, inputs, dialogs,
 or side panels
3. The primary content currently shown on screen

Write 2-4 sentences.
Do not invent details that are not visible in the screenshot.
Return plain text only, not JSON.
```

JOINT GENERATION PROMPT

```
## Role
You are a trajectory generator, not a real agent.
You must simulate how a smart and efficient agent would complete exactly the current step of a computer-using task.


## Current trajectory diversity strategy
{granularity_instruction}

## Important constraints
- Do not generate multiple steps at once.
1. The screenshot description already tells you the current screen state, so you do not need a separate "view page
" or "inspect page" tool.
2. Do not call wait-like tools more than once in a row.
3. Do not call wait-like tools unless there is a clear error condition.
4. Every step must make progress toward the task goal.
5. Assume the network, application, and desktop environment are functioning normally. Do not wait for loading
unless the state explicitly requires it.
6. Choose only a tool action that can plausibly move the current state to a later real state in the recorded
trajectory.
7. Do not call terminate unless the task has already reached the final recorded state.

## Task goal
{goal}

## Trajectory history
{history}

## Current screenshot description
{screenshot_description}

## Current virtual world state
{world_state}

## Available tools for this task
{tools}

Generate the complete output for the current step, including:
1. Observation
2. Thought
3. Action
4. Tool Call
5. Tool Response

## Rules for generating the Step
- The observation should reflect the current screenshot description, especial attention to important region.
- Thougt must demonstrate the agent's real thinking progress advancing the task given current step.
- Action must be a natural-language description of the intended tool call action, it should be a simple, short and
 concise sentence. NOT any explict GUI desctiption like 'click', 'drag', 'key' and so no.
- Tool call should be a valid JSON foramt of the intended tool call.

## Rules for generating the Tool Response
- The response fields must strictly follow the selected tool's returns schema.
- Do not include fields that are not defined by the tool's returns schema.
- The value of success must be consistent with the current world_state.
- Concrete values in result, such as URLs, file paths, window names, or extracted content, must come from the
trajectory history or the current screenshot description.
- Do not invent a tool usage whose effect would not be visibly groundable to a later recorded screenshot.
```

```
## Output requirements
- Generate exactly one step.
- The root object must contain the following fields and no wrapper layer.
- Output as exactly the JSON object in the following format:

## Output format
{{
  "observation": "Observation grounded in the current screenshot description",
  "thought": "Reasoning about the next Tool calling action. It must clearly advance the task.",
  "action": "Simple and concise natural-language description of the intended Tool calling action",
  "tool_call": {{
    "tool_name": "One of the available tools",
    "tool_parameters": {{
      "parameter_name": "parameter value that matches the tool schema"
    }}
  }},
  "tool_response": {{
    "success": true,
    "result": "...",
    "error_message": null
  }}
}}
```

TOOL GENERATION PROMPT

```
    You need to design a set of semantic-level tools for the following computer-using task.
The tools must be useful for this task and also be grounded to the state transitions visible in the recorded
trajectory.

## Task goal
{goal}

## Current diversity round
{round_name}

## Tool granularity strategy
{granularity_instruction}

## Recorded trajectory frame descriptions
{trajectory_context}

## Required tool schema reference
{meta_schema}

## Tool design principles
- Think about the specific applications or software likely used in this task and design tools accordingly.
- Generate no more than 15 tools an no less than 5 tools.
- Category must be one of: navigation / interaction / extraction / filesystem / system / terminate.
- Every non-terminate tool must include ‘granularity‘ with value ‘fine‘ or ‘coarse‘.
- The ‘terminate‘ tool may omit ‘granularity‘, but if present it must be ‘coarse‘.
- Each tool’s returns object must contain exactly these fields:
  - success (boolean)
  - result (object/string/null)
  - error_message (string/null)
- Do not generate wait / sleep / delay / get_ui_tree style tools.
- Each tool should represent one complete user intention, not one primitive UI gesture.
- A user intent such as "open the export dialog" or "apply the date filter" should be a single tool even if the
actual UI would require multiple clicks.
- The tools must express user intent at the desktop workflow level, not low-level UI mechanics.

  Avoid low-level tools such as:
  - click_element(element_description)
  - input_text(field, text)
  - scroll_down(pixels)
  - tap_coordinates(x, y)

  Prefer semantic tools such as:
  - open_application(application_name)
  - navigate_to_section(section_name)
  - search_records(query)
  - set_filter(filter_name, value)
  - export_report(destination_path)
  - confirm_submission()

## Multi-granularity requirement
- The tool library must contain both finer-grained tools and coarser-grained tools.
- Fine-grained tools should capture a focused semantic intent that usually corresponds to one local sub-goal, such
 as opening a specific panel, applying one filter, selecting one record, or confirming one dialog.
- Coarse-grained tools should capture a broader semantic intent that may cover several adjacent UI operations,
such as preparing an export, completing a checkout stage, or finishing a search-and-download subroutine.
- Keep every tool grounded to visible state transitions in the recorded trajectory even when it is coarse-grained.
- Avoid making every tool equally granular. The mix should reflect the current round strategy.
- Among non-terminate tools, include at least {min_fine_grained_tools} fine-grained tools and at least {
min_coarse_grained_tools} coarse-grained tools.

## Grounding rule
- Prioritize tools that explain real state transitions visible in the recorded trajectory frames.
- The main tool set should be sufficient for the observed workflow in this trajectory.
```

```
- You may add a small number of app-relevant helper tools, but do not invent a broad toolbox that cannot be
grounded to the recorded states.

## Naming rule
- Use lowercase_with_underscores.
- If a tool is clearly tied to a specific desktop application, its name must start with that application name, for
 example:
  - ‘libreoffice_export_pdf‘
  - ‘vscode_open_user_settings‘
  - ‘chrome_download_file‘
Do not use the specific website name as the application name, where ’chrome’ is the suitbale application name.
- If no concrete application can be determined, the name must start with ‘general_‘.
- Avoid vague names whose purpose is unclear from the name alone.
- Bad examples:
  - ‘open_file‘
  - ‘set_option‘
  - ‘click_button‘
  - ‘edit_content‘

## Terminate rule
- There must be exactly one tool named ‘terminate‘.
- The ‘terminate‘ tool must use category ‘terminate‘.
- No non-‘terminate‘ tool may use category ‘terminate‘.
- No other tool may express task-ending semantics such as finish task, complete task, end task, stop task, or
report task completion.


## Existing tools that may be reused
{existing_tools}

## Output format
{{
  "tools": [...]
}}
Return valid JSON only.
"""


FIX_TOOL_PROMPT = """
The following tool definition is invalid:
{tool}

## Validation errors
{error_msg}

## Required schema
{meta_schema}

Please repair the tool definition so that it matches the schema.
Return exactly one JSON object in the following format:
{{
  "name": "...",
  "description": "...",
  "granularity": "fine or coarse",
  "parameters": {{}},
  "returns": {{
    "success": {{"type": "boolean"}},
    "result": {{"type": ["string", "object", "null"]}},
    "error_message": {{"type": ["string", "null"]}}
  }},
  "category": "...",
  "side_effects": []
}}
```

FIX TOOL PROMPT

```
The following tool definition is invalid:
{tool}

## Validation errors
{error_msg}

## Required schema
{meta_schema}

Please repair the tool definition so that it matches the schema.
Return exactly one JSON object in the following format:
{{
  "name": "...",
  "description": "...",
  "granularity": "fine or coarse",
  "parameters": {{}},
  "returns": {{
    "success": {{"type": "boolean"}},
    "result": {{"type": ["string", "object", "null"]}},
    "error_message": {{"type": ["string", "null"]}}
  }},
  "category": "...",
  "side_effects": []
}}
```

PREDICT SCREENSHOT PROMPT

```
## Role
You are a desktop UI state predictor.

## Task goal
{goal}

## Action that was just performed
- Tool: {tool_name}
- Parameters: {tool_parameters}
- Result: {tool_response}

## UI state before the action
{previous_screenshot_description}

## Current world state
{world_state}

## Instruction
Predict what the desktop interface most likely shows after that action.
Be concrete. Mention the current page, window, dialog, visible UI elements, and any visible data or status changes.

Write 3-5 sentences in English.
Return plain text only, not JSON.
```

DESCRIBE AND LOCATE PROMPT

```
## Action that was just performed
Tool: {tool_name}
Parameters: {tool_parameters}
Result: {tool_response}

## Tasks

### Task 1: Describe the current screenshot
Describe the current desktop screenshot before the action in 3-5 English sentences.

### Task 2: Locate the best matching result screenshot
You are given {num_candidates} candidate screenshots, indexed from 1.
Select the candidate that best represents the UI state after the action has completed.

## Output format
{{
  "current_description": "English description of the current screenshot",
  "matched_index": 1,
  "confidence": "high/medium/low/none",
  "reason": "What visible evidence shows that this screenshot matches the post-action state"
}}

If none of the candidates match, set matched_index to null.
```

MERGE TREE PLANNING PROMPT

```
You are planning an optimal bottom-up merge tree for a fine-grained desktop tool trajectory.

## Task goal
{goal}

## Fine-grained leaf steps
{leaf_summary}

## Constraints
- Leaf steps are indexed from 0 to {max_leaf_index}.
- Preserve the original left-to-right leaf order exactly.
- Every internal merge node must merge between 2 and {max_branching_factor} contiguous children.
- Children may be either direct leaf indices or nested internal nodes.
- Every internal node must represent a coherent higher-level sub-goal.
- Prefer a tree that maximizes semantic cohesion inside each merge and clear abstraction jumps between levels.
- Avoid unnecessary merges that do not create a meaningful higher-level action.
- The final root must cover all leaves from 0 to {max_leaf_index}.
- The tree height must not exceed {max_coarse_levels} coarse levels above the fine leaves.
- Return English only.

## Output format
{{
  "tree": {{
    "summary": "Short summary of the root-level task subroutine",
    "children": [
      0,
      {{
        "summary": "Merged sub-goal summary",
        "children": [1, 2]
      }},
      {{
        "summary": "Another merged sub-goal summary",
        "children": [
          {{
            "summary": "Nested merged sub-goal",
```

```
          "children": [3, 4]
        }},
        5
      ]
    }}
  ]
 }}
}}

Return valid JSON only.
```

BOTTOM UP MERGE PROMPT

```
You are performing bottom-up multi-granularity tool synthesis for a desktop task trajectory.

## Task goal
{goal}

## Target granularity level
{target_level}

## Instruction
You are given a contiguous chunk of already-grounded finer-grained tool steps.
Synthesize one broader tool that compresses the whole chunk into a single higher-level semantic action.
The merged tool must still be grounded to the same final UI state reached by the chunk.

## Rules
- The merged tool must summarize the whole chunk, not just the final fine-grained action.
- The merged tool must be semantically broader than each constituent fine-grained tool.
- Do not invent effects beyond the final state already reached by the chunk.
- The merged tool must use ‘granularity = "coarse"‘.
- The merged tool call name in ‘merged_step.tool_call.tool_name‘ must exactly match ‘tool_definition.name‘.
- The merged step’s screenshot index and result state will be assigned by the caller, so do not include them.
- Keep the tool response grounded to the final fine-grained step in the chunk.
- Do not use terminate semantics.
- Return English only.

## Chunk summary
{chunk_summary}

## Output format
{{
  "tool_definition": {{
    "name": "lowercase_with_underscores",
    "description": "One-sentence broader semantic intent for the whole chunk",
    "granularity": "coarse",
    "parameters": {{}},
    "returns": {{
      "success": {{"type": "boolean"}},
      "result": {{"type": ["string", "object", "null"]}},
      "error_message": {{"type": ["string", "null"]}}
    }},
    "category": "navigation|interaction|extraction|filesystem|system",
    "side_effects": []
  }},
  "merged_step": {{
    "observation": "Observation of the UI state before the merged action starts",
    "thought": "Reasoning about why this broader tool covers the whole chunk",
    "action": "Short natural-language summary of the merged higher-level action",
    "tool_call": {{
      "tool_name": "same as tool_definition.name",
      "tool_parameters": {{
        "parameter_name": "parameter value"
      }}
    }},
    "tool_response": {{
      "success": true,
      "result": {{}},
      "error_message": null
    }}
  }}
}}

Return valid JSON only.
```

F Messages for ToolCUA

Our ToolCUA utilizes the following message construction for training and inference. The system prompt is composed with predefined GUI actions, optional tool-calling actions, and an "important_reminder” section.

ToolCUA Messages

System Prompt

```
IMPORTANT_GUI_ONLY = """<IMPORTANT>
Reminder:
- The ‘computer_use‘ function provides **GUI actions** to interact with the computer directly via mouse and
keyboard.
- After each GUI action, you will receive a new screenshot reflecting the current state of the screen.
- Always consult the latest screenshot before deciding your next action.
</IMPORTANT>"""

IMPORTANT_WITH_MCP = """<IMPORTANT>
Reminder:
- Use ‘computer_use‘ to interact with the computer via mouse and keyboard.
    - ‘computer_use‘ GUI actions usually only provide a simple success result such as ‘Success‘.
    - After each action, you will receive a new screenshot of the current state of the computer.
- If there are other functions, they are mcp tool actions, used to interact with the mcp server in computer.
    - Their results are returned as screenshot and a textual raw JSON string with fields ‘success‘, ‘result‘, and ‘
    error_message‘.
    - Some mcp tool actions may NOT cause any visible change in the screenshot, so rely on the JSON tool result
    when appropriate.
    - Do NOT use ‘read_text_file‘ to read PDF or other non-plaintext files; use GUI actions instead.
    - Do NOT repeat the same mcp tool call if it keeps failing or produces no useful progress -try a different
    approach or terminate.
    - Do NOT repeatedly call ‘env_info‘ tools to retrieve file information.
</IMPORTANT>"""


OSWorld_MCP_SYS_PROMPT = (
"# Tools\n\n"
"You may call one or more functions to assist with the user query.\n\n"
"You are provided with function signatures within <tools></tools> XML tags:\n"
"<tools>\n"
"{{"
    ’"type": "function", ’
    ’"function": {{’
        ’"name_for_human": "computer_use", ’
        ’"name": "computer_use", ’
        ’"description": (’
            ’"Use a mouse and keyboard to interact with a computer, and take screenshots.\\n’
            ’* This is an interface to a desktop GUI. You do not have access to a terminal or applications menu. You
             must click on desktop icons to start applications.\\n’
            ’* Some applications may take time to start or process actions, so you may need to wait and take
            successive screenshots to see the results of your actions. E.g. if you click on Firefox and a window
            doesn\’t open, try wait and taking another screenshot.\\n’
            ’* The screen\’s resolution is 1000x1000.\\n’
            ’* Whenever you intend to move the cursor to click on an element like an icon, you should consult a
            screenshot to determine the coordinates of the element before moving the cursor.\\n’
            ’* If you tried clicking on a program or link but it failed to load even after waiting, try adjusting
            your cursor position so that the tip of the cursor visually falls on the element that you want to click
            .\\n’
            ’* Make sure to click any buttons, links, icons, etc with the cursor tip in the center of the element.
            Don\’t click boxes on their edges unless asked."’
        ’), ’
        ’"parameters": {{’
            ’"properties": {{’
                ’"action": {{’
                    ’"description": (’
                        ’"\\n’
                        ’* ‘key‘: Performs key down presses on the arguments passed in order, then performs key
                        releases in reverse order.\\n’
                        ’* ‘type‘: Type a string of text on the keyboard.\\n’
                        ’* ‘mouse_move‘: Move the cursor to a specified (x, y) pixel coordinate on the screen.\\n’
                        ’* ‘left_click‘: Click the left mouse button at a specified (x, y) pixel coordinate on the
                        screen.\\n’
                        ’* ‘left_click_drag‘: Click and drag the cursor to a specified (x, y) pixel coordinate on the
                         screen.\\n’
                        ’* ‘right_click‘: Click the right mouse button at a specified (x, y) pixel coordinate on the
                        screen.\\n’
                        ’* ‘middle_click‘: Click the middle mouse button at a specified (x, y) pixel coordinate on
                        the screen.\\n’
                        ’* ‘double_click‘: Double-click the left mouse button at a specified (x, y) pixel coordinate
                        on the screen.\\n’
                        ’* ‘triple_click‘: Triple-click the left mouse button at a specified (x, y) pixel coordinate
                        on the screen (simulated as double-click since it\’s the closest action).\\n’
                        ’* ‘scroll‘: Performs a scroll of the mouse scroll wheel.\\n’
                        ’* ‘hscroll‘: Performs a horizontal scroll (mapped to regular scroll).\\n’
                        ’* ‘wait‘: Wait specified seconds for the change to happen.\\n’
                        ’* ‘terminate‘: Terminate the current task and report its completion status.\\n’
                        ’* ‘answer‘: Answer a question.\\n’
                        ’       "’
                    ’), ’
                    ’"enum": ["key", "type", "mouse_move", "left_click", "left_click_drag", "right_click", "
                    middle_click", "double_click", "scroll", "wait", "terminate"], ’
                    ’"type": "string"’
                ’}}, ’
                ’"keys": {{"description": "Required only by ‘action=key‘.", "type": "array"}}, ’
                ’"text": {{"description": "Required only by ‘action=type‘.", "type": "string"}}, ’
```

```
            '"coordinate": {{"description": "The x,y coordinates for mouse actions.", "type": "array"}}, '
            '"pixels": {{"description": "The amount of scrolling.", "type": "number"}}, '
            '"time": {{"description": "The seconds to wait.", "type": "number"}}, '
            '"status": {{"description": "The status of the task.", "type": "string", "enum": ["success", "
            failure"]}}'
        '}}, '
        '"required": ["action"], '
        '"type": "object"'
    '}}, '
    '"args_format": "Format the arguments as a JSON object."'
'}}'
'}}\n'
"{tool_list}\n"
"</tools>\n\n"
"For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML
tags:\n"
"<tool_call>\n"
'{{"name": <function-name>, "arguments": <args-json-object>}}\n'
"</tool_call>\n\n"
"{important_reminder}\n\n"
"# Response format\n\n"
"Response format for every step:\n"
"1) Action: a short imperative describing what to do in the UI, or specifying which tool to invoke\n"
"2) A single <tool_call>...</tool_call> block containing only the JSON: "
'{{"name": <function-name>, "arguments": <args-json-object>}}.\n\n'
"Rules:\n"
"- Output exactly in the order: Action, <tool_call>.\n"
"- Be brief: one sentence for Action.\n"
"- Do not output anything else outside those parts.\n"
"- If finishing, use action=terminate in the tool call."
)
```

Message Construction

```
# We construct a multimodal message list as follows:
# 1) A system message containing the tool-calling specification. The system prompt is composed of three parts:
#    (a) the predefined GUI action schema for ‘computer_use‘ (always present);
#    (b) optional mcp tool definitions appended from obs["tool_list"] (only when action_space == "mcp" and tools
exist);
#    (c) an important-reminder section -either
#    IMPORTANT_WITH_MCP or IMPORTANT_GUI_ONLY -(only when action_space == "mcp", omitted entirely when action_space
 == "pyautogui").
# 2) For historical context, we keep: steps before the history window [0, t0) as a text-only action history
embedded in instruction_prompt ("Step 1: ..., Step 2: ..., ..."); and at most the most recent history_n (= 5)
screenshots (image history), each paired with its tool-calling result in a <tool_response> block.
# 3) For each retained past step i in [t0, T): append a user message containing -only for the first retained step
i == t0 -instruction_prompt prepended as a text part (task instruction + text-only pre-window action history),
followed by a <tool_response> block consisting of the textual tool-calling result tool_calling_results[i] and
screenshot[i] as an image_url part; then append an assistant message with the model's response at step i (Action +
 <tool_call>).
# 4) For the current step T: if T == 0 and history_n > 0 (very first step, no tool has been called yet), append a
user message with instruction_prompt and the bare current screenshot as an image_url part (no <tool_response>
wrapper); otherwise (T > 0, or T == 0 with history_n == 0), append a user message containing a <tool_response>
block with cur_tool_calling_result and the current screenshot as an image_url part, additionally prepending
instruction_prompt when history_n == 0 since no prior turn carried it.

# Key variables:
# - T: current step index (0-based), i.e., T = len(actions)
# - history_n: maximum number of past steps retained as image history, fixed at 5
# - t0: start index of the history window, t0 = max(0, T - history_n)
# - screenshots[i]: the processed screenshot at step i, uploaded as an OSS URL
# - tool_calling_results[i]: the textual tool-calling result received at step i, i.e.,
#   obs["exe_result"] after executing step i's action; defaults to "Success" if empty
# - responses[i]: the model's response text at step i (Action + <tool_call>)
# - cur_screenshot: OSS URL of the current screenshot at step T
# - cur_tool_calling_result: tool_calling_results[T], the result received before step T

messages = [
    {"role": "system", "content": [{"type": "text", "text": system_prompt}]}
]

t0 = max(0, T - history_n)

previous_actions_str = "None" if t0 == 0 else "\n".join(
    [f"Step {i+1}: {actions[i]}" for i in range(t0)]
)

instruction_prompt = f"""Please generate the next move according to the UI screenshot, instruction and previous
actions.

Instruction: {instruction}

Previous actions:
{previous_actions_str}"""

for i in range(t0, T):
    user_content = []
    if i == t0:
        user_content.append({"type": "text", "text": instruction_prompt})
    user_content += [
```

```
        {"type": "text",    "text": "<tool_response>\n"},
        {"type": "text",    "text": tool_calling_results[i]},
        {"type": "image_url", "image_url": {"url": screenshots[i]}},
        {"type": "text",    "text": "\n</tool_response>"},
    ]
    messages.append({"role": "user", "content": user_content})
    messages.append({"role": "assistant", "content": [{"type": "text", "text": responses[i]}]})

if T == 0 and history_n > 0:
    messages.append({
        "role": "user",
        "content": [
            {"type": "text",    "text": instruction_prompt},
            {"type": "image_url", "image_url": {"url": cur_screenshot}},
        ]
    })
else:
    current_user_content = []
    if history_n == 0:
        current_user_content.append({"type": "text", "text": instruction_prompt})
    current_user_content += [
        {"type": "text",    "text": "<tool_response>\n"},
        {"type": "text",    "text": cur_tool_calling_result},
        {"type": "image_url", "image_url": {"url": cur_screenshot}},
        {"type": "text",    "text": "\n</tool_response>"},
    ]
    messages.append({"role": "user", "content": current_user_content})
```