UI-Evol: Automatic Knowledge Evolving for Computer Use Agents

Ziyun Zhang [*1] **Xinyi Liu** [*1] **Xiaoyi Zhang** [2] **Jun Wang** [2] **Gang Chen** [2] **Yan Lu** [2]

Abstract

External knowledge has played a crucial role in the recent development of computer use agents. We identify a critical knowledge-execution gap: retrieved knowledge often fails to translate into effective real-world task execution. Our analysis shows even 90% correct knowledge yields only 41% success rate. To bridge this gap, we propose **UI-Evol**, a plug-and-play module for autonomous GUI knowledge evolution. UI-Evol consists of two stages: a *Retrace Stage* that extracts faithful objective action sequences from actual agent-environment interactions, and a *Critique Stage* that refines existing knowledge by comparing these sequences against external references. We conduct comprehensive experiments on the OSWorld benchmark with the state-of-the-art Agent S2. Our results demonstrate that UI-Evol not only significantly boosts success rate but also addresses a previously overlooked issue of high behavioral standard deviation in computer use agents, leading to superior performance on computer use tasks and substantially improved agent reliability.

1. Introduction

Building a computer use agent that can automatically interact with Graphical User Interfaces (GUI) and complete specified tasks with minimal human intervention has always been a major challenge in the field of Autonomous Agents (Hu et al.; ant, 2024; ope, 2024). The diversity and heterogeneity inherent in modern GUI interfaces (Common Crawl, 2025) demand that computer use agents exhibit highly robust and generalizable visual perception capabilities. Moreover, the successful long-trace execution of complex, long-horizon tasks further necessitates agents to demonstrate strong reasoning and sequential decision-making capability. Recent progress in proprietary Large Multimodal Models (LMMs) (OpenAI, 2023; Anthropic, 2024) has substantially advanced the foundational capabilities of computer use agents. This has enabled a surge in research efforts utilizing LMMs as core reasoning engines for automated interface manipulation (Yan et al., 2023; Zheng et al., 2024; Agashe et al., 2024; 2025).

Despite these advancements, it is still challenging for current LMMs to manipulate software and computer system (Xie et al., 2024; Bonatti et al., 2024; Zhou et al.) based solely on their own knowledge learned in training. To mitigate these limitations, some studies (Zhang et al., 2023a; Li et al., 2024; Wu et al.; Zheng et al., 2025) have adopted Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) to augment agents with external task-specific knowledge. By retrieving task-relevant external knowledge, these systems can enhance task planning and execution, alleviating the burden on agents to synthesize execution strategies from scratch for every single scenario (Zhang et al., 2023a; Li et al., 2024; Wu et al.; Zheng et al., 2025; Agashe et al., 2024; 2025). For instance, representative Agent S series (Agashe et al., 2024; 2025) utilize online web search to obtain existing rich and up-to-date knowledge for given task instruction, including task interpretations and step-by-step plans. Experiments have shown that web-based knowledge can significantly improve agent performance.

Nevertheless, our analysis of Agent S2 (Agashe et al., 2025) based on GPT-4o reveals a persistent gap between the availability of correct knowledge and the knowledge can be effectively consumed by agent for task execution. Specifically, in our sampling survey, even when 90% of the knowledge retrieved via Perplexica (per, 2024) is deemed "correct" from the human perspective, the best agent's success rate is only 41%. More details on the sampling survey are provided in Appendix B. Further investigation into this phenomenon reveals that although the external knowledge may appear theoretically correct and appropriate, it often suffers from certain practical drawbacks, including the omission of necessary intermediate steps (which human might consider as natural), assumptions inconsistent with the initial task conditions, and the suggestion of sub-optimal execution paths demanding overly complex manipulations. Figure 1 illustrates

[*]Equal contribution [1]School of Software and Microelectronics, Peking University. Work done during internship in Microsoft Research Asia [2]Microsoft Research Asia. Contract to: Xiaoyi Zhang <xiaoyizhang@microsoft.com>.

Workshop on Computer-use Agents @ Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2026.

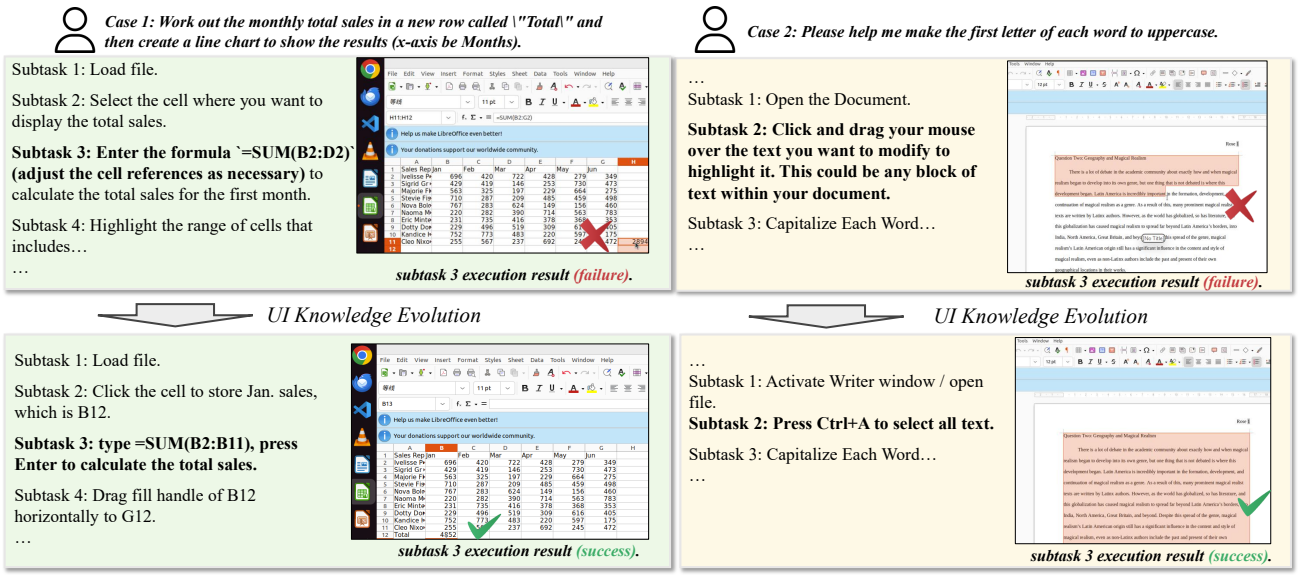

Figure 1. The green box shows web-retrieved task knowledge, while the yellow box shows evolved knowledge from our approach. Web knowledge is generally correct but often lacks practical details (left) or suggests with more complex manipulations (right).

two representative cases. In Subtask 3 of Case 1, the external knowledge merely suggests summarizing the column and adjusting the cell reference, which misdirects the agent and leads to task execution failure. Similarly, in Case 2, the provided external web knowledge advises using a click-and-drag action to select text, whereas the task explicitly requires selecting "all". Such advice proves challenging for a computer use agent to execute accurately through mouse dragging alone. These shortcomings underline a critical gap between externally retrieved web knowledge and the actionable knowledge that agents can effectively consume for practical task completion.

To address the aforementioned problem, we propose UI-Evol, a plug-and-play module that can be seamlessly integrated into existing computer use agent systems by introducing an autonomous GUI knowledge evolution mechanism aimed at improving knowledge through realistic interactions with practical environments. Specifically, given a particular task and corresponding initial task knowledge, a computer use agent first performs task operations within the environment, producing an interaction record of actual task execution behaviors. Subsequently, UI-Evol autonomously refines the existing knowledge based on actual task execution behaviors, thus mitigating the gap between external knowledge and practical environments. Specifically, UI-Evol consists of two stages: **Retrace Stage** and **Critique Stage**. In the Retrace stage, instead of solely relying on the agent's planned actions for introspective reflection, UI-Evol extracts the actual actions executed by the computer use agent based on the screenshots before and after each manipulation. These vision-driven observations are then synthesized into a *objective action sequence*, which is a detailed, structural trajectory description that faithfully captures the agent's behavior in the environment, thereby ensuring an accurate and unbiased representation of the agent's concrete interactions with the computer environment. In the Critique stage, leveraging our earlier observation that retrieved web knowledge is largely reliable, UI-Evol uses the externally retrieved knowledge as a reference anchor and further complement it. A carefully constructed series of reasoning patterns then assesses and critiques the extracted objective action sequence. Specifically, the Critique stage compares the agent-produced action sequences against the reference web knowledge, identifies deviations or anomalies, analyzes underlying causes for these discrepancies, with chain-of-thought reasoning (Wei et al., 2022) to generate a refined version of task-specific knowledge. This newly evolved knowledge is subsequently stored in a dedicated knowledge base, where it serves as improved reference guidance for subsequent agent executions.

We conduct comprehensive experiments on OSWorld (Xie et al., 2024) to evaluate the effectiveness of UI-Evol on state-of-the-art Agent S2. The experimental results demonstrate that knowledge evolved by UI-Evol is better aligned with an agent in the practical environment. Notably, during experimentation, we discovered significant instability (high standard deviation) in the baseline computer use agents even when we fix all the hyperparameters we can set, which has been largely overlooked in prior research. To systematically examine this instability issue and rigorously evaluate robustness of our approach, we developed a highly efficient parallel evaluation framework. Utilizing 30 parallel instances, this framework greatly accelerates the agent evaluation, reducing running time from 10 hours to 2.5 hours,

achieving approximately 4 times accelerate, thus allowing extensive repetitions of experiments to observe and report the significance of our experiment. We repeated each experiment variant three times to precisely measure variability. The extensive experimental analyses confirm that UI-Evol not only boosts overall success rate but also notably reduces behavioral standard deviation, thus significantly enhancing the robustness and stability of computer use agents.

Our contributions can be summarized as follows:

- We identify the gap between externally acquired knowledge and real task execution, and propose UI-Evol, a plug-and-play module that effectively bridges this gap by autonomously evolving GUI task knowledge.
- We are the first to systematically identify and analyze the previously overlooked instability issue in contemporary computer use agents, and develop a highly parallelized environment to facilitate efficient investigation.
- Comprehensive experiments on the OSWorld benchmark show UI-Evol achieves state-of-the-art accuracy and significantly reduces behavioral standard deviation, substantially enhancing agent robustness and stability.

2. Related Work

2.1. Computer Use Agent

Through the early exploration on automatic computer task execution (Zhang et al., 2023c; Fu et al., 2024; Li et al., 2020), recent computer use agents can be broadly categorized into two types: monolithic agents and modular agents. A monolithic agent is typically built using a single post-trained, end-to-end model that independently handles the entire computer use task (Zhang et al., 2023c; Hong et al., 2024; Cheng et al., 2024; Wu et al., 2024; Liu et al., 2025b; Lin et al., 2024; Xu et al., 2024; Qin et al., 2025). Recent open-source pre-trained MLLMs also have native support for computer use via function calls (Bai et al., 2025). In addition to supervised fine-tuning, recent works have explored reinforcement learning approaches to train R1-like models (Zhang et al., 2023b; Xia & Luo, 2025; Liu et al., 2025c; Lu et al., 2025). With the increase in both data size and model size, monolithic agents have achieved stronger performance on benchmarks. However, this comes at the expense of high computational demands and scalability limitations. Modular agents decompose computer use into multiple modules to reduce the burden on each single model. Some approaches train a separate grounding model and use propriety models to generate actions (Gou et al., 2024; Liu et al., 2025a; Yang et al., 2024). Some works further divide action generation into multiple stages such as planning and acting, and incorporate external tools for support. For example, some works design multi-level hierarchical planning procedures (Agashe et al., 2024; Wang & Liu, 2024), while some construct skill libraries to simplify action generation (Zheng et al., 2025; Zhang et al., 2023a; Tan et al., 2024; Wu et al.). There are also methods that retrieve knowledge from the web to serve as a reference during planning (Agashe et al., 2025). However, these additional modules increase the complexity of the overall system and the coupling among them can negatively affect the robustness of the agent.

2.2. Knowledge refinement and self-evolution

Early work has demonstrated that LLM-based agents are capable of analyzing failures and summarizing experiences in interactive environments (Wang et al., 2023; Zhu et al., 2023), thereby enabling self-evolution (Zhou et al., 2024). A complete self-evolution process typically contains an iterative cycle involving the acquisition, refinement, updating, and evaluation of knowledge (Tao et al., 2024). Among them, knowledge refinement refers to filtering or correcting knowledge based on environmental feedback, which assists the LLM to adapt to new information and contexts. In environments with factual feedback, some work utilizes such objective signals to guide the refinement of knowledge (Chen et al., 2023; Zelikman et al., 2024). More commonly, refinement is driven by an additional critique process, including critiques independently generated by LLM itself (Lu et al., 2023; Madaan et al., 2023), or critiques produced during interactions between LLM and external tools such as code interpreter and Wikipedia (Gou et al.; Jiang et al., 2023). Providing appropriate references for the critique process in complex tasks can lead to substantial improvements.

3. Preliminary

Computer use agent with external knowledge. Recent computer use agents typically leverage external knowledge sources to reduce the necessity of generating execution strategies entirely from scratch for each given task. This external knowledge-based approach, which serves as the starting point for our proposed approach, enhances the robustness and success rate of agent execution. To clearly illustrate how external knowledge is generated and integrated into agent execution, we take Agent S2 (Agashe et al., 2025) as a representative example. Specifically, Agent S2 first synthesizes an appropriate query based on the provided task instruction and initial environment state. It then retrieves relevant information via Perplexica (per, 2024), subsequently summarizing the retrieved information into a structured list of sub-tasks. This sub-task list represents the external knowledge format adopted throughout this work. During task execution, the structured sub-task list is utilized as part of the prompting context provided to the agent, offering a soft prior to assist in efficient and accurate task planning.

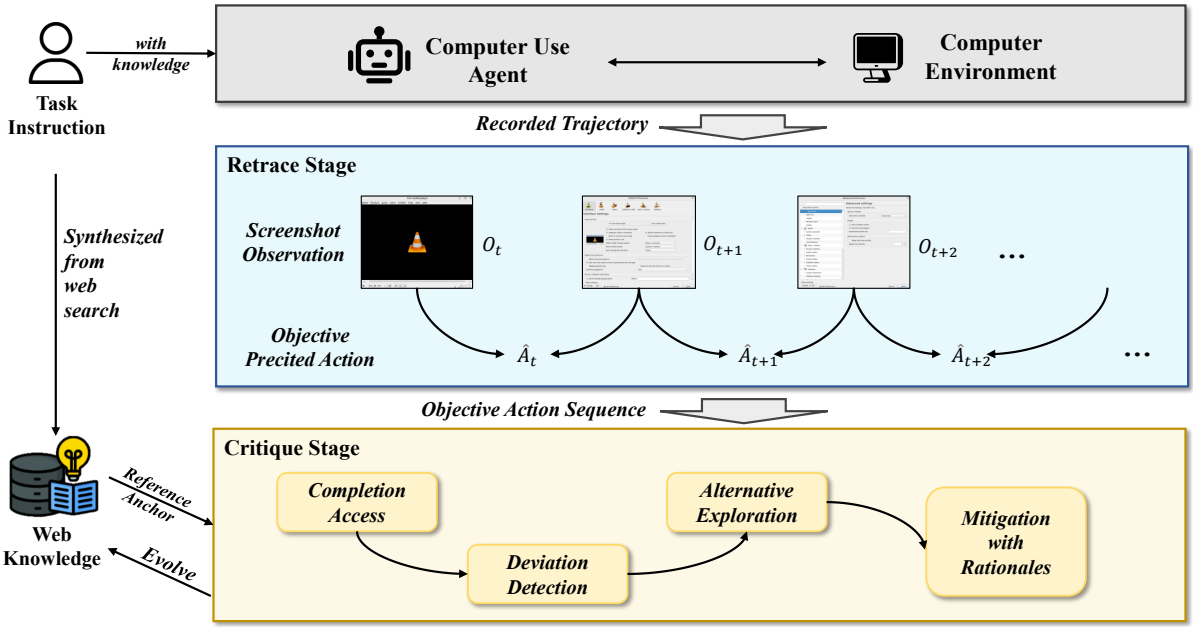

Figure 2. UI-Evol consists of two stages: Retrace replays screenshots to recover objective actions; Critique uses web knowledge to detect deviations, explore alternatives, and output rationale-backed fixes that are fed back into the knowledge base.

4. Method

To mitigate the gap between external knowledge and the actual computer use environment, we propose UI-Evol, a plug-and-play retrospective knowledge evolution module for computer use agents. Though our method can be applied into any knowledge-based agent, here we take Agent S series as the example to describe our framework. Following Agent S series (Agashe et al., 2024; 2025), we interpret the web-based knowledge retrieved by agents as a soft prior task plan over the agent's action policy via prompts. As depicted in Figure 2, to evolve it, after we obtain the web knowledge as described in Section 3, we first execute the task instructions to obtain the recorded trajectory. For the recorded trajectory and web knowledge, UI-Evol consists of two stages: **Retrace** and **Critique**. In the Retrace stage, UI-Evol reconstructs the agent's actual trajectory based on screenshots. In the Critique Stage, carefully designed chain-of-thought reasoning patterns are employed to guide the agent in analyzing the causes of deviation and updating the knowledge accordingly. Without additional human supervision, this process enables the knowledge base to evolve automatically and better support agent execution. We then introduce the details of the two stages separately.

4.1. Retrace Stage

Due to the intrinsic hallucination tendency and limited UI perception capabilities of Large Multimodal Models, existing computer use agents often generate infeasible actions or incorrectly interpret the current computer state. Moreover, the inherent complexity of the computer use environment further exacerbates this issue, leading to executed actions failing to produce the intended effects. As a result, the action sequences originally output by agents, termed *subjective action sequences*, may not accurately represent actual state changes on the user interface. In practice, the subjective action sequence merely represents the intended (rather than actual) behavior of the agent. To address the misalignment between the subjective trajectory and the actual state transitions, we introduce the Retrace Stage. In this stage, we propose the Retrace stage to reconstruct an accurate sequence of executed actions, termed *objective action sequences*, based purely on the observed screenshots captured during execution.

Formally, for each step t in the recorded trajectory, given the observations O_t and O_{t+1} before and after the step, the Retrace Stage first enumerates the screenshot in O_t, and then compares the changes in O_{t+1} relative to O_t. A LMM is used to analyze these changes and objectively predict the action $\hat{A}_t$ that occurred during step t. If the LMM thinks nothing happen between O_t and O_{t+1}, $\hat{A}_t$ is assigned a null value. The union of all objective actions across steps yields the *objective action sequence*.

We represent the objective action sequence as a sequence of textual descriptions of action, which serves as the input for the next stage. Through the Retrace Stage, the resulting objective trajectory is free from noise introduced by invalid actions, thereby enabling a more reliable comparison with external knowledge sources.

Table 1. Performance (%) of original web knowledge versus UI-Evol-refined knowledge across different backbone models on five task groups. SR. denotes the success rate. Std. denotes the standard deviation. Agent S2* is our reproduction result.

Method	Base Model	Min. SR.	Max. SR.	Std. SR.	Avg. SR.	Reported SR.
OpenAI Operator	OpenAI CUA	-	-	-	-	19.7
UI-TARS	UI-TARS-72B-SFT	-	-	-	-	18.7
UI-TARS	UI-TARS-72B-DPO	-	-	-	-	22.7
Aria-UI	GPT-4o	-	-	-	-	15.2
Aguvis-72B	GPT-4o	-	-	-	-	17.0
Agent S2	GPT-4o	-	-	-	-	21.1
Agent S2	Claude-3.7-Sonnet	-	-	-	-	27.0
Agent S2*	GPT-4o	18.3	20.8	±1.00	19.5	19.5
+ UI-Evol	GPT-4o	**21.0**	**22.7**	**±0.71**	**22.0**	**22.0**
Agent S2*	OpenAI-o3	24.3	27.0	±1.09	25.6	25.6
+ UI-Evol	OpenAI-o3	**28.1**	**28.6**	**±0.26**	**28.4**	**28.4**

Table 2. Performance (%) of random vs. completion-based trajectory selection. Rand. Select denotes random selection, and Comp. Select denotes completion-based selection.

Method	Base Model	OS	Daily	Office	Professional	Workflow	Avg. SR.
Ours w/Rand. Select	GPT-4o	47.22	27.83	17.98	35.61	10.43	22.0
Ours w/Comp. Select	GPT-4o	44.45	30.39	20.54	29.96	11.43	22.7

Table 3. Performance (%) of transfering evolved knowledge from OpenAI-o3 to GPT-4o.

Knowledge Base	Base Model	OS	Daily	Office	Professional	Workflow	Avg. SR.
Web Search	GPT-4o	51.39	23.98	14.27	32.43	9.11	19.5
Evolved from 4o Traj.	GPT-4o	47.22	27.83	17.98	35.61	10.43	22.0
Evolved from o3 Traj.	GPT-4o	48.61	26.12	23.09	31.33	8.78	22.4

4.2. Critique Stage

The Critique Stage is designed to refine and systematically enhance the quality of agent knowledge. Our preliminary sampling analysis has demonstrated that the retrieved web-based knowledge generally represents a reliable task-guidance source and thus is leveraged as a reference anchor and further complement it in our framework. By comparing the reference anchor and the objective action sequence reconstructed during the Retrace Stage, the Critique Stage identifies the gaps between the knowledge and the agent's actual behavior. Subsequently, it formulates targeted refinements to directly address and mitigate these discrepancies, thereby systematically evolving the knowledge base toward a closer alignment with the agent's actual execution policies within real-world task environments. This process results in more effective guidance, greater interpretability, and improved performance for future agent runs.

Specifically, the Critique Stage leverages the chain-of-thought reasoning paradigm, composed of a carefully structured sequence of reasoning steps. Given the *objective action sequence* generated in the Retrace Stage, along with previously acquired web-based knowledge and the task instruction itself, a large language model (LLM) conducts a progressive, multi-stage analysis. This analysis comprises three investigation stages, namely (**1**) Completion Assessment, (**2**) Deviation Detection, and (**3**) Alternative Exploration, followed by a final mitigation stage, namely (**4**) Mitigation with Rationales. In detail, each analysis step serves a distinct analytical purpose as outlined below:

- **Completion Assessment:** This initial step involves assessing whether the agent successfully completed the intended task. Specifically, the LLM compares the outcome depicted in the objective action sequence to the task goal defined in the provided instruction, clearly determining if the task was fully executed or partially completed.
- **Deviation Detection:** Subsequently, the LLM conducts a thorough comparative examination between the objective action sequence and the original knowledge-guided action plan. The goal of this step is to explicitly identify deviations, *i.e.*, discrepancies or contradictions between the

intended actions outlined by the external knowledge and the actual actions performed by the agent. Crucially, the LLM also infers plausible explanations and root-causes underlying these mismatches, providing deeper insight into systemic failures in action planning or perception that caused the divergence.

- **Alternative Exploration:** In this stage, analysis shifts toward understanding the agent's strategic behavior more comprehensively. The LLM assesses whether the agent, during task execution, attempted valid alternative action strategies beyond or deviating from the original knowledge-based instructions. Identifying such alternative solutions not only yields insights into the robustness and flexibility of the agent's behavior, but also helps to enrich and diversify the evolved knowledge representation.
- **Mitigation with Rationales:** Based upon observations, insights, and causal explanations extracted from previous steps, the final mitigation stage synthesizes actionable refinements and corrections. The LLM systematically proposes clear and logically-grounded revisions to the original knowledge base, along with corresponding rationales that explicitly justify changes introduced by this critique process. The resultant output maintains the same representational format as the original knowledge, but is systematically refined and enhanced.

Ultimately, upon completing these carefully designed reasoning steps, the LLM generates an updated and refined knowledge representation that explicitly incorporates the identified corrections, clarifications, and supplementary alternative strategies. This renewed knowledge aligns more closely with the agent's actual behavior, addresses previously discovered inconsistencies, and provides richer, instruction-specific guidance. The evolved knowledge is then recorded and stored in the knowledge base to guide next-round agent executions of analogous tasks.

5. Experiments

To demonstrate the effectiveness of our approach, we conducted a series of experiments on OSWorld (Xie et al., 2024). OSWorld is an interactive environment comprising 369 open-ended computer tasks, where the agent can interact with the environment using screenshots as observations. We first introduce our parallelization improvements to OSWorld. We then evaluate the performance of UI-Evol against the baseline, followed by further ablation studies. All experiments are conducted under maximum 15 steps.

5.1. Parallel Evaluation Framework

Because of the inherent stochasticity of large language models, reliably assessing agent performance requires running a large number of repeated trials. However, the original OSWorld benchmark supports only single-machine evaluation and takes about 10 hours to finish even the screenshot-only setting, making large-scale experimentation impractical. To address this limitation, inspired by Windows Agent Arena (Bonatti et al., 2024), we extend OSWorld by developing a parallel evaluation infrastructure on Microsoft Azure.

Specifically, we employ Azure Machine Learning jobs to parallelize benchmark evaluations across multiple compute instances. Unlike OSWorld, which launches multiple VMWare virtual machines on a single local machine, our implementation automatically provisions one virtual machine per compute instance. Evaluation tasks are evenly distributed across these instances to enable parallel execution. During environment setup, our implementation bypasses repeated Docker image builds that are required in Windows Agent Arena. Instead, it downloads the environment and code directly from the cloud storage. The results are also aggregated in the cloud storage when all experiments finish.

Our implementation enables scalable evaluation of OSWorld, allowing the number of instances to scale up to the total number of benchmark tasks. In our experiments, we used 30 instances, which reduces the full-process runtime of the OSWorld benchmark from approximately 10 hours to 2.5 hours under maximum 15 steps.

5.2. Main Results

Settings. For our experiments, we adopt GPT-4o and OpenAI-o3 as the base models for Retrace Stage and Critique Stage separately. We first run computer use agent with web knowledge to obtain the recorded trajectory and the knowledge to be evolved later. Then we run our UI-Evol to evolve the knowledge. Finally we equip the same computer agent with the updated knowledge to evaluate the improvement brought by the knowledge evolution. We select Agent S2 (Agashe et al., 2025) as our baseline, which is the leading computer use agent on OSWorld. Agent S2 uses Perplexica for web search and references the retrieved external knowledge to decompose user instructions into multiple subtasks, which are then executed by a coordinated set of agents. Since Agent S2 retrieves web knowledge at the beginning of each run, we capture and freeze a snapshot of the entire knowledge base before the first experiment while set all hyperparameters as constant value including temperature as 0. This eliminates variability caused by dynamic web.

Stability. During experiments, we observe that even with static precaptured web knowledge and fixed hyperparameters, repeated evaluations may still yield different results. We argue that producing stable and reproducible outcomes is a critical capability for agents, especially for real-world deployment. Therefore, we include stability as one of our evaluation criteria. For each experimental variant, we re-

peat the same run three times and compute the mean and standard deviation of the success rate. The mean reflects the agent's overall performance, while the standard deviation indicates the agent's stability.

Results Using Agent S2 as the baseline, we compare the performance of using only the original web search knowledge with that of using knowledge refined by UI-Evol across different base models. As reported in Table 1, UI-Evol consistently improves the average success rate and reduces standard deviation on both models. This demonstrates that UI-Evol can effectively enhance both agent performance and stability. Notably, when integrating UI-Evol, the standard deviation of OpenAI-o3 drops to as low as 0.26, approximately 4.19 times the reduction observed with GPT-4o. This suggests that models with stronger reasoning capabilities are better at understanding and leveraging external knowledge, and therefore benefit more from UI-Evol.

5.3. Ablation Study

As mentioned in Section 5.2, we repeat each run for three times during evaluation. Due to the stochastic nature of LLMs, the trajectories generated in each run may differ. To study how the selection strategy affects the final results, we compare two approaches: **(1)Random selection**, where a trajectory is randomly chosen from the repeated runs; **(2) Completion-based selection**, where all trajectories are first transformed to textual format through the Retrace Stage, and then an LLM is prompted to select the trajectory from completeness. We detail the prompt in our appendix.

To measure the quality of selection, we introduce the Selection Success Rate (SSR) metric, defined as $\mathrm{SSR} = \frac{N_{\text{succ}}}{N_{\text{solv}}}$, where N_{succ} is the number of cases where the selected trajectory (τ_i) is successful, and N_{solv} is the number of cases in which at least one out of three repeated runs succeeds. Here, τ_i denotes a trajectory generated in an experiment. A higher SSR indicates that the completeness of trajectories.

Under this metric, random selection achieves an SSR of 70%, while completion-based selection reaches 85%. This demonstrates that the LLM can identify trajectories that are more likely to be correct. We use trajectories selected by these two approaches as inputs to UI-Evol, and compare their performance using GPT-4o as the base model. As shown in Table 2, UI-Evol with the trajectories chosen based on completeness performs only marginally better. These results suggest that UI-Evol is robust to the quality of trajectory selection and can effectively leverage knowledge from both better and worse inputs.

5.4. Knowledge Transfer

To evaluate whether knowledge evolved from one model can be effectively reused by another, we use the OSWorld trajectories generated by OpenAI-o3 to derive refined knowledge through UI-Evol. This knowledge is then provided as input to Agent S2 with GPT-4o, as shown in Table 3. Compared to the original GPT-4o baseline which leverages web knowledge, the agent using refined knowledge derived from OpenAI-o3 trajectories performs similarly to that of using GPT-4o's own trajectories. These findings demonstrate that the knowledge refined through UI-Evol can indeed be transferred across different models, capturing task regularities.

5.5. Case Study

Case introduction. In the task of Figure 3, the agent is required to capitalize the first letter of each word in a LibreOffice Writer file. The first step is to select all the words in the document. From web knowledge, the agent is instructed to select entire text by clicking and dragging with the mouse. However, despite strictly following the guidance of the knowledge, the agent only selected part of the text in the first attempt and dragged the selected text to the end of the passage in the second, ultimately failing the task.

Evolution progress. After the trajectory is fed into UI-Evol, the system first enters the Retrace Stage, where each step is distilled into two textual elements: Action and Result. In the first Retrace step, UI-Evol analyzes the pre-execution screenshot and summarizes the initial state, correctly noting that the agent selected only the section from "Question Two: Geography and Magical Realism" to "so important." In the second Retrace step, UI-Evol precisely describes both the outcome of that action and the agent's error—dragging the selected text from the beginning to the end of the passage.

With this objective record in hand, the Critique Stage evaluates the trajectory. During Completion Assessment it observes that the task failed because the drag operation altered the document's structure. In Deviation Detection it traces the failure to an error in the selection step and labels it an Output/Screen Misunderstanding. The Alternative Exploration stage finds no evidence that the agent attempted other methods. In the Mitigation with Rationales stage, it recommends replacing manual dragging with the more reliable Ctrl + A shortcut, noting that this change will appear in step 2 of the final plan.

Finally, drawing on all prior analyses, UI-Evol revises step 2 accordingly, thereby completing the learning pipeline and incorporating the new knowledge into its repertoire.

5.6. Computational Cost and Scalability

As detailed in Section 4, UI-Evol consists of an initial raw-trace collection phase followed by two sequential stages: Retrace and Critique. During the raw-trace collection phase, 30 OSWorld instances process jobs in parallel, requiring approximately two hours to generate the raw traces. The

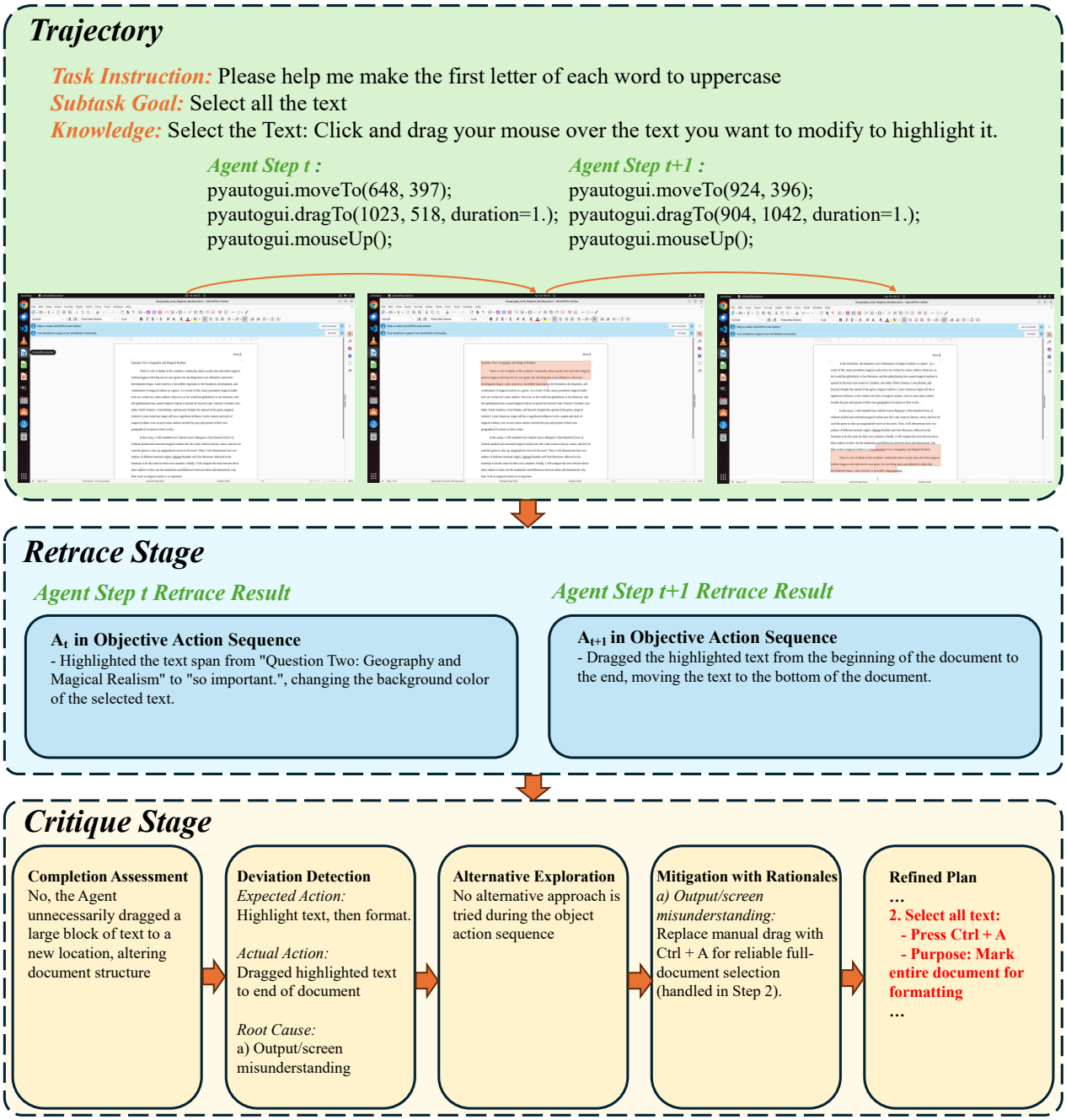

Figure 3. Case study on the "capitalize every word" task from the OSWorld benchmark: Our UI-Evol first retraces the objective action sequence from the screenshots and identifies that the action taken at step t was selecting only a part of the paragraph. In the Critique Stage, it detects that this action deviates from the objective, as the entire document should have been selected rather than a partial selection. Finally, our framework corrects this deviation by proposing a simpler keyboard shortcut, "Ctrl + A", instead of dragging with the mouse.

subsequent Retrace and Critique stages are executed on 12 parallel threads and together process all 369 OSWorld tasks in roughly one hour.

The primary monetary cost stems from OpenAI API usage. In the Retrace stage, each 15-step trace consumes around 85000 input tokens and 400 output tokens and is processed by the GPT-4o model. In the Critique stage, each trace requires around 800 input tokens and 150 output tokens and is handled by the OpenAI-o3 model. On average, each task incurs a cost of $0.22, resulting in a total expenditure of approximately $81.18 for the entire benchmark.

For scalability, UI-Evol's task-level pipeline is highly parallelizable: adding compute instances yields near-linear throughput gains while maintaining per-task latency, ensuring efficient development iteration at scale.

6. Conclusion

In this work, we tackle the knowledge-execution gap in computer use agents. We propose UI-Evol, an autonomous, plug-and-play module that evolves GUI interaction knowledge by evolving external knowledge based on actual agent behaviors. Our experiments on OSWorld show UI-Evol significantly improves task accuracy and, crucially, reduces agent behavioral variance, enhancing stability. This work offers a possible solution towards more reliable and advanced autonomous computer use.

Impact Statement

Computer use agents hold immense potential to significantly boost human productivity on digital devices, yet their misuse poses considerable risks to society. For instance, these agents could be exploited for malicious purposes, such as the automated creation of numerous spam accounts, leading to a deluge of unwanted content and potential security breaches.

References

Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku, 2024. URL https://www.anthropic.com/news/3-5-models-and-computer-use.

Computer-using agent, 2024. URL https://openai.com/index/computer-using-agent/.

Perplexica, 2024. URL https://github.com/ItzCrazyKns/Perplexica. Accessed: 2025-05-03.

Agashe, S., Han, J., Gan, S., Yang, J., Li, A., and Wang, X. E. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

Agashe, S., Wong, K., Tu, V., Yang, J., Li, A., and Wang, X. E. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

Anthropic, A. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 2024.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

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

Chen, X., Lin, M., Schaerli, N., and Zhou, D. Teaching large language models to self-debug. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

Cheng, K., Sun, Q., Chu, Y., Xu, F., YanTao, L., Zhang, J., and Wu, Z. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

Common Crawl. Common crawl - open repository of web crawl data, 2025. URL http://commoncrawl.org.

Fu, J., Zhang, X., Wang, Y., Zeng, W., and Zheng, N. Understanding mobile gui: From pixel-words to screen-sentences. *Neurocomputing*, 601:128200, 2024.

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

Gou, Z., Shao, Z., Gong, Y., Yang, Y., Duan, N., Chen, W., et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Hong, W., Wang, W., Lv, Q., Xu, J., Yu, W., Ji, J., Wang, Y., Wang, Z., Dong, Y., Ding, M., et al. Cogagent: A visual language model for gui agents. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14281–14290. IEEE, 2024.

Hu, X., Xiong, T., Yi, B., Wei, Z., Xiao, R., Chen, Y., Ye, J., Tao, M., Zhou, X., Zhao, Z., et al. Os agents: A survey on mllm-based agents for computer, phone and browser use.

Jiang, S., Wang, Y., and Wang, Y. Selfevolve: A code evolution framework via large language models. *arXiv preprint arXiv:2306.02907*, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Li, Y., He, J., Zhou, X., Zhang, Y., and Baldridge, J. Mapping natural language instructions to mobile ui action sequences. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*, 2020. URL https://www.aclweb.org/anthology/2020.acl-main.729.pdf.

Li, Y., Zhang, C., Yang, W., Fu, B., Cheng, P., Chen, X., Chen, L., and Wei, Y. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*, 2024.

Lin, K. Q., Li, L., Gao, D., Yang, Z., Bai, Z., Lei, W., Wang, L., and Shou, M. Z. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Liu, X., Zhang, X., Zhang, Z., and Lu, Y. Ui-e2i-synth: Advancing gui grounding with large-scale instruction synthesis. *arXiv preprint arXiv:2504.11257*, 2025a.

Liu, Y., Li, P., Wei, Z., Xie, C., Hu, X., Xu, X., Zhang, S., Han, X., Yang, H., and Wu, F. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *arXiv preprint arXiv:2501.04575*, 2025b.

Liu, Y., Li, P., Xie, C., Hu, X., Han, X., Zhang, S., Yang, H., and Wu, F. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*, 2025c.

Lu, J., Zhong, W., Huang, W., Wang, Y., Mi, F., Wang, B., Wang, W., Shang, L., and Liu, Q. Self: Language-driven self-evolution for large language model. *CoRR*, 2023.

Lu, Z., Chai, Y., Guo, Y., Yin, X., Liu, L., Wang, H., Xiong, G., and Li, H. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

OpenAI. Gpt-4 technical report, 2023.

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

Tan, W., Zhang, W., Xu, X., Xia, H., Ding, Z., Li, B., Zhou, B., Yue, J., Jiang, J., Li, Y., et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024.

Tao, Z., Lin, T.-E., Chen, X., Li, H., Wu, Y., Li, Y., Jin, Z., Huang, F., Tao, D., and Zhou, J. A survey on self-evolution of large language models. *CoRR*, 2024.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wang, X. and Liu, B. Oscar: Operating system control via state-aware reasoning and re-planning. *arXiv preprint arXiv:2410.18963*, 2024.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wu, Z., Han, C., Ding, Z., Weng, Z., Liu, Z., Yao, S., Yu, T., and Kong, L. Os-copilot: Towards generalist computer agents with self-improvement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

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

Xia, X. and Luo, R. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.

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

Yan, A., Yang, Z., Zhu, W., Lin, K., Li, L., Wang, J., Yang, J., Zhong, Y., McAuley, J., Gao, J., et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.

Yang, Y., Wang, Y., Li, D., Luo, Z., Chen, B., Huang, C., and Li, J. Aria-ui: Visual grounding for gui instructions. *arXiv preprint arXiv:2412.16256*, 2024.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. D. Star: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proc. the 36th International Conference on Neural Information Processing Systems*, volume 1126, 2024.

Zhang, C., Yang, Z., Liu, J., Han, Y., Chen, X., Huang, Z., Fu, B., and Yu, G. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023a.

Zhang, Z., Xie, W., Zhang, X., and Lu, Y. Reinforced ui instruction grounding: Towards a generic ui task automation api. *arXiv preprint arXiv:2310.04716*, 2023b.

Zhang, Z., Zhang, X., Xie, W., and Lu, Y. Responsible task automation: Empowering large language models as responsible task automators. *arXiv preprint arXiv:2306.01242*, 2023c.

Zheng, B., Gou, B., Kil, J., Sun, H., and Su, Y. Gpt-4v (ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 61349–61385, 2024.

Zheng, B., Fatemi, M. Y., Jin, X., Wang, Z. Z., Gandhi, A., Song, Y., Gu, Y., Srinivasa, J., Liu, G., Neubig, G., et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

Zhou, W., Ou, Y., Ding, S., Li, L., Wu, J., Wang, T., Chen, J., Wang, S., Xu, X., Zhang, N., et al. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*, 2024.

Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., et al. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.

A. Domain Specific Results on OSWorld

Table 4. Performance (%) of different settings across individual applications and overall score on GPT-4o.

Method	Chrome	Gimp	Calc	Impress	Writer	Multiapps	OS	TB	VLC	VSCode	Avg. SR.
Agent S2*	22.31	16.66	7.09	17.10	23.18	9.11	51.39	35.56	18.30	46.38	19.49
+ UI-Evol w/Rand. Select	25.12	21.80	9.22	20.64	30.43	10.43	47.22	44.44	20.27	47.83	22.02
+ UI-Evol w/Comp. Select	27.38	17.95	11.35	18.52	43.47	11.43	44.45	44.44	26.14	40.58	22.74
+ UI-Evol w/knowledge evolved from o3	21.02	19.23	16.31	26.29	30.43	8.78	48.61	44.45	23.76	42.03	22.38

Table 5. Performance (%) of different settings across individual applications and overall score on OpenAI-o3.

Method	Chrome	Gimp	Calc	Impress	Writer	Multiapps	OS	TB	VLC	VSCode	Avg. SR.
Agent S2*	28.21	20.51	19.15	24.19	40.57	9.11	55.55	51.11	31.82	47.83	25.64
+ UI-Evol	29.71	32.05	24.82	25.59	44.92	12.42	55.56	42.22	32.07	46.38	28.28

B. Sampling Survey On Web Knowledge

To evaluate the reliability of knowledge retrieved from web via Perplexica, we conduct a sampling survey. Specifically, we sampled 50 test cases proportionally across various domains from OSWorld, and manually assess the correctness of the knowledge retrieved in each case. The knowledge is considered "correct" if an annotator can successfully complete the given task using only that information without relying on any prior domain knowledge. Under this criterion, 45 of the 50 sampled cases are deemed to contain correct knowledge from the human perspective.

C. Prompts

Prompts for Retrace Stage

```
You are a senior QA assistant.
You receive:
• BEFORE screenshot <image0>
• AFTER  screenshot <image1>
• A snippet of Python automation code.

Your task:

PART A - BEFORE DESCRIPTION
Describe concisely and objectively what is visible in the BEFORE screenshot only.
• <= 80 words, declarative sentences.
• No speculation, no mention of AFTER, no hidden reasoning.

PART B - UI OPERATION LIST
List, in chronological order, every visible UI step (mouse-click, key-stroke, drag,
↪  menu selection...) that converted the BEFORE state into the AFTER state.

OUTPUT FORMAT (STRICT)
[A] BEFORE
<one-to-three short sentences that satisfy PART A>

[B] OPERATIONS
- <action>, <visible consequence>
- ...

RULES FOR PART B (inherited)
1. Bullet list; every line begins with "-".
2. Each bullet MUST pair the action with its visible consequence, e.g.
- Clicked the "Replace All" button in VS Code's Search sidebar, replacing all 12
↪  occurrences of "text" with "test" in the open file
```

```
3. Do not add headings, explanations or blank lines beyond the specified format.
4. If the ONLY difference is the system clock, Part B must contain exactly one bullet:
- No operations performed.
5. If the screenshots cannot be compared, Part B must contain exactly one bullet:
- Unable to determine operations.

Think step-by-step internally but reveal ONLY the two required sections.

FEW-SHOT EXAMPLES

<BEGIN_EXAMPLE>
# Normal change with visible result
BEFORE: VS Code shows 3 occurrences of "foo"
AFTER : All occurrences now read "bar"
CODE  : editor.replace_all("foo", "bar")
OUTPUT:
[A] BEFORE
VS Code editor window is open; the Find/Replace panel indicates 3 matches for the word
↪  "foo".

[B] OPERATIONS
- Pressed Ctrl+H in the VS Code editor, opening the Find/Replace panel
- Typed "foo" into the Find box, highlighting 3 matches in the file
- Typed "bar" into the Replace box
- Clicked the "Replace All" button in the Find/Replace panel, replacing all 3
↪  occurrences of "foo" with "bar" in the document
<END_EXAMPLE>

<BEGIN_EXAMPLE>
# Only the clock changed
BEFORE: Desktop 10:01
AFTER : Desktop 10:02
OUTPUT:
[A] BEFORE
Desktop environment showing wallpaper and system clock reading 10:01.

[B] OPERATIONS
- No operations performed.
<END_EXAMPLE>

<BEGIN_EXAMPLE>
# Incomparable
BEFORE: Corrupted screenshot
AFTER : Corrupted screenshot
OUTPUT:
[A] BEFORE
Screenshot is corrupted; no discernible UI elements are visible.

[B] OPERATIONS
- Unable to determine operations.
<END_EXAMPLE>

The FIRST image (<image0>) shows the screen BEFORE the Agent acted.
The SECOND image (<image1>) shows the screen AFTER the Agent acted.

The Agent executed the following Python code:

```python
{code}
List the UI operations (action + visible result).
```
```

Prompts for Critique Stage

```
INPUT
  Task Instruction: ...
  Action List: ...
  Original Plan: ...

REQUIREMENTS
    • Follow the FIVE SECTION HEADERS below exactly.
    • SECTION E output style:
          1. **<Subtask>**:
             - <Concrete UI / CLI action(s) only>
             - Purpose: <<= 10-word reason>
    • If a field is not applicable, write \None" or \No deviation".
    • If SECTION C judges an Alternative better, the final NEW PLAN must adopt it (or
    ↪  its key advantages).
    • Every Root Cause from SECTION B must have a mitigation explained in SECTION D and
    ↪  be implicitly addressed (not as a standalone step) in SECTION E.
    • Exclude passive \Confirm / Verify / Check / Make sure ..." kinds of steps.
    • Visual inspections are assumed; do not list them.
    • If the Action List shows a dialog / branch / extra option that the Original Plan
    ↪  did not anticipate:
    - Treat it as a Deviation (Root Cause usually f) Invalid assumption).
    - If the Agent picked the wrong option, SECTION D must state the correct option and
    ↪  SECTION E must insert that corrected step.
    - If the Agent picked the right option, still add that step to SECTION E (it is an
    ↪  \added step").
    - Any action shown to be unnecessary in the trajectory must be omitted from SECTION
    ↪  E (this is a \removed step").

SECTION A. Task Completion
  Did the Agent achieve the task goal? (Yes / No)
  Reason.
  Did the Agent execute more than the instruction required? (Yes / No)
  Reason.

SECTION B. Deviation Analysis
  For every mismatch between an Original-Plan assumption and the actual screen/CLI
  ↪  output in Action List, record a Deviation row. Fill in ALL items, even if \No
  ↪  deviation".
  • Deviation Step: <# or \None">
  • Expected Action : ...
  • Actual Action    : ...
  • Root Cause (letters, commas allowed):
        a) Output/screen misunderstanding
        b) Knowledge gap
        c) Command / code / syntax error
        d) Environment or permission issue
        e) Other
        f) Invalid assumption
        g) External transient failure
        h) Step order issue
        i) Missing precondition

SECTION C. Alternative Approaches
  Did the Agent attempt any approach beyond the Original Plan? (Yes / No)
  If Yes:
      • Describe each approach briefly.
      • Which is better (Original / Alternative)? Why?
  If No: \No alternative approach tried."

SECTION D. Mitigation & Rationale
```

```
  For every Root Cause from SECTION B, describe the preventive or corrective idea and
  ↪  mention which forthcoming step embodies it.
  Example:
      c) Syntax error → Add \lint before run" check (handled in Step 2).
      d) Permission → Verify sudo rights before executing installer (Step 5).
      f) Invalid assumption → Choose \Typical" in installer dialog (Step 2).

SECTION E. REFINED PLAN:
  REFINED PLAN:
      1. **<Subtask>**:
         - <Concrete action(s)>
         - Purpose: <Why this step?>
      2. **<Subtask>**:
         - <Concrete action(s)>
         - Purpose: ...
      ...
      up to 15 steps total.
  • No shell prompts (#, $).
  • Safeguards are implicit per SECTION D; do not list them as separate lines.
  • Newly added corrective steps must appear in the proper sequence among Steps 1-15;
  ↪  actions deemed unnecessary must not appear here.
```

Prompts for Completion-based selection

```
INPUT:
1. Task Instruction   : The instruction for the task.
2. Action_List1: The list1 of actions performed by a linux user.
3. Golden_Plan1: The plan1 that the user is trying to achieve to solve a task.
4. Action_List2: The list2 of actions performed by a linux user.
5. Golden_Plan2: The plan2 that the user is trying to achieve to solve a task.
6. Action_List3: The list3 of actions performed by a linux user.
7. Golden_Plan3: The plan3 that the user is trying to achieve to solve a task.

REQUIREMENTS:
You need to find out the best action_list and golden_plan pair that is most likely
↪  completed or closest to completion.
Your output should include the following information:
- <Analysis and Score>: Analysis and Give your score(0 to 10) for each pair, 10 is the
↪  best, 0 is the worst.
- <Best Pair>: A number in [1, 2, 3] indicating the best action_list and golden_plan
↪  pair.
```