OS-Catalyst: Advancing Computer-Using Agents Efficiency through Adaptive Action Compression

Anonymous ACL submission

Abstract

Driven by advances in Vision-Language Models (VLMs), computer-using agents have recently demonstrated remarkable capabilities in complex reasoning, software control, and the automation of digital workflows. However, the existing step-by-step paradigm requires extensive interaction with the model, and the resulting query latency emerges as a key bottleneck for real-world adoption. To address this limitation, we propose that agents should be able to output a sequence of actions after each observation, enabling efficient execution without constant model queries. In this work, we introduce OS-CATALYST, a method that transforms standard computer-using models into agents with the capability of action sequence prediction. To enable this, we design a data collection pipeline tailored for compressed action trajectories in computer-using environments. Building on this pipeline, we construct a large-scale dataset within the WorkArena benchmark and train computer-using agents for action sequence prediction. Through extensive experiments, we show that OS-CATALYST enables up to 50% faster task completion on office-related benchmarks without sacrificing success rate. Our resources are available at link.

1 Introduction

In recent years, the rapid development of Large Language Models (LLMs) (Anthropic, 2025; OpenAI, 2025a; Bai et al., 2025) has driven LLM-based agents to become a key research focus in both academia and industry (OpenAI, 2025b; Manus, 2025). These agents demonstrate strong capabilities in information processing and knowledge reasoning, with growing potential for handling complex tasks through direct interactions with operating systems and applications. In particular, "computer-using agents" (Cheng et al., 2024; Anthropic, 2025; Qin et al., 2025; Sun et al., 2024b) are designed to simulate human–computer interactions by operating directly on Graphical User Interfaces (GUI), perceiving dynamic layouts, grounding references to interface elements, and planning subsequent actions. Such agents are expected to reduce operational barriers for both daily affairs and professional tasks, thereby advancing human–computer collaboration.

Currently, GUI agents (OpenAI, 2025a; Liu et al., 2025c) primarily interact through the step-by-step paradigm shown in Figure 1 (a). Given an instruction, the model processes inputs such as screenshots or accessibility trees, and iteratively produces reasoning and corresponding actions until completion. Multi-agent frameworks (Agashe et al., 2025a; Jia et al., 2024; Wu et al., 2024) follow a similar paradigm with additional planning stages. Such an interaction paradigm requires agents to execute 10 to 30 steps to complete a single GUI task, with model querying accounting for most of the execution time. Consequently, completing a single GUI task often takes at least several minutes, introducing nontrivial bottlenecks for real-world deployment and commercial adoption.

However, we observe that in many scenarios, especially office-related tasks, this interactive paradigm leaves significant room for compression. In fact, it is often unnecessary to obtain a new observation before every single action. For humans, when completing tasks such as filling out a form, a single round of observation is often sufficient to determine the type and location of several subsequent actions, as shown in Figure 1 (b). Inspired by this observation, we propose OS-CATALYST, which introduces a novel interaction paradigm for GUI agents. OS-CATALYST inte-

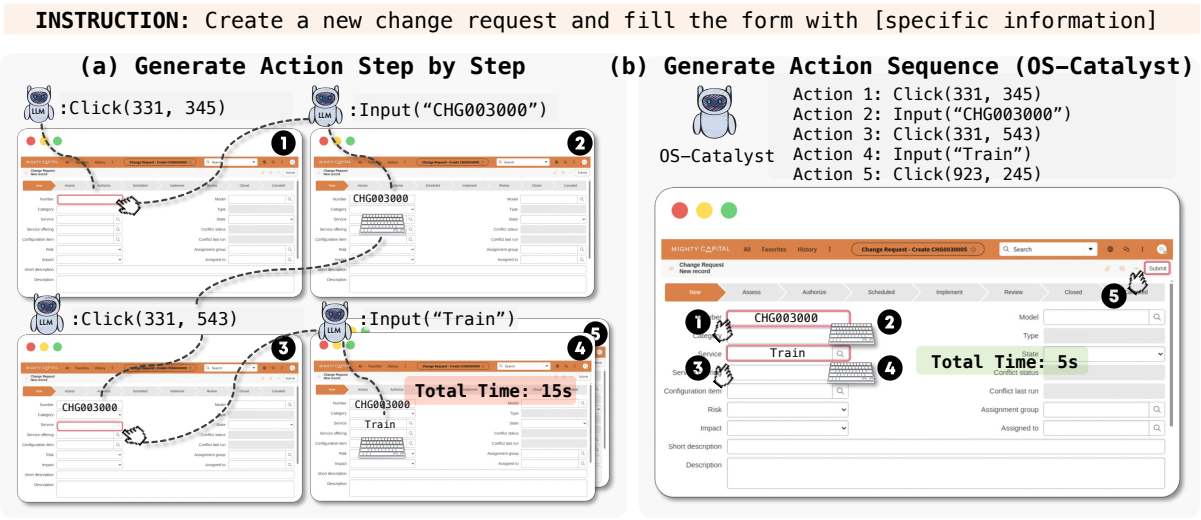


Figure 1: Main idea of OS-CATALYST. Traditional GUI agents generate actions step by step (a), which incurs repeated model–environment interactions and high latency. OS-CATALYST (b) enables the model to predict multiple valid actions in one step, thus compressing trajectories and improving execution efficiency.

grates action-sequence prediction idea, a tailored data compression pipeline, and fine-tuning strategies, enabling models to output coherent multi-step action sequences from a single observation.

We conduct measurements on WorkArena (Drouin et al., 2024), a benchmark designed for GUI-based office tasks. The results reveal that if the model outputs the maximal feasible sequence of actions after each observation, at least 50% of task execution steps can be eliminated. However, current models are unable to correctly infer such multi-step action sequences through prompt-based guidance alone. This suggests that existing GUI agents can hardly reason about longer action plans within a given observation. To address this, we construct a dataset within the WorkArena environment and train models based on UI-TARS (Qin et al., 2025). OS-CATALYST achieves up to a 50% reduction in task execution times compared with the step-by-step paradigm.

- We propose a novel direction to improve the efficiency of GUI agents through **adaptive action compression**, which reduces unnecessary observations between sequential actions.
- We construct a dataset and develop corresponding models within the WorkArena environment, enabling multi-step action prediction from a single observation.
- We achieve up to **50% reduction** in task execution time on office benchmarks, demonstrating the effectiveness of OS-CATALYST.

2 Related Work

Computer-using agents. Unlike early LLM-based agents that parse GUIs into structured text (Deng et al., 2023; Zhou et al., 2024) and navigate through tools (Sun et al., 2024a) or API calls (Wu et al., 2024; Zhang et al., 2024a), VLM-based GUI agents directly perceive raw screenshots and output atomic keyboard/mouse operations—boosting adaptability while introducing new challenges. First, VLMs must perceive detailed information and localize elements in high-resolution screenshots. Beyond supervised pre-training on grounding datasets (Cheng et al., 2024; Chen et al., 2024b; Xu et al., 2024; Gou et al., 2024; Wu et al., 2025c), efforts include training on high-resolution processing (Hong et al., 2024; Yang et al., 2024; Li et al., 2024) or token selection (Ge et al., 2024; Wu et al., 2025b; Zhang et al., 2024b) modules, and designing reasoning strategies for dynamic focusing or test-time scaling (Wu et al., 2025a; Yang et al., 2025; Liu et al., 2025b). Furthermore, computer-using agents require strong multi-turn planning capabilities (Xie et al., 2024b; Sun et al., 2025). Two mainstream approaches exist: one uses elaborately designed agentic workflows for proprietary VLMs (Zhang et al., 2025b; Jiang et al., 2025; Zheng et al., 2024a; Wang et al., 2024; Jia et al., 2024; Agashe et al., 2025b), comprising external modules for hierarchical planning, memory organization, and multi-agent collaboration; the other

conducts supervised fine-tuning and reinforcement learning to endow open-source VLMs with native long-horizon reasoning and error recovery (Wang et al., 2025a; Xia and Luo, 2025; Liu et al., 2025a; Qi et al., 2024).

Efficiency of agent workflows. Agentic workflows rooted in the CoT (Wei et al., 2022) and ReAct-style (Yao et al., 2023) paradigms unlock LLMs' capabilities for complex tasks, while simultaneously significantly increasing tool invocation complexity and context length—ultimately leading to higher costs and degraded performance. To address the issue of reasoning inefficiency, within the multi-agent setup, DAAO (Su et al., 2025) leverages the complementary advantages of heterogeneous models and introduces LLM routing based on query difficulty estimator to implement an adaptive orchestration system. Optima (Chen et al., 2024a) and Puppeteer (Dang et al., 2025) integrate the balance between performance and efficiency into reward functions, continuously enhancing the system's dynamic orchestration and adaptive evolution capabilities through reinforcement learning.

In the specific context of computer-using tasks, OS-Copilot (Wu et al., 2024), Mobile-Agent-E (Wang et al., 2025b), and AppAgentX (Jiang et al., 2025) extract repetitive patterns from historical actions, organize them into shortcuts or tool scripts, and store these in long-term memory for reuse and improved efficiency. Similarly, UFO^2 (Zhang et al., 2025a) incorporates speculative multi-action output; yet GUI environments require system API calls for robust execution as target elements shift unpredictably. Dyna-Think (Yu et al., 2025) demonstrates effective multi-action reasoning under accessibility-tree–based UI representation, leveraging structured semantic and hierarchical information. In contrast, models relying purely on visual observations still struggle to achieve comparable quality, lacking explicit structural cues. OSWorld-Human (Abhyankar et al., 2025) recognizes this limitation and provides manually grouped action annotations as an evaluation benchmark. Building on this insight, we internalize prediction of environmental dynamics into the model through large-scale supervised training.

3 Method

We propose OS-CATALYST, a method that enables GUI agents to improve efficiency by adaptively predicting action sequences. In the following, we detail the components of OS-CATALYST. We first introduce the formulation of action sequences and describe how they are executed in the environment. Next, we present the dataset construction pipeline, which produces both step-level and compressed trajectories. Finally, we present the overall process of training our model.

3.1 Action Sequence Formulation

Most GUI agents today work step by step: the model outputs one action (or a fixed combination of two actions, for example, a click followed by a type), the environment executes it, and then the model is prompted again to predict the next action (Sun et al., 2024b; Gou et al., 2024). This repeats until task completion. While simple, this approach is often inefficient. For example, imagine a form that requires filling multiple fields. A human can look at the page once and immediately know the next several operations, such as *clicking field A → typing the name*, then *clicking field B → typing the gender*, and so on.

Inspired by this, we expect the model to predict multiple consecutive actions in advance as well. The model should adaptively decide the length and content of the sequence based on the task, interface, and progress. We define an action sequence as a short list of consecutive actions predicted at once. The environment executes them one by one in order, but the model is prompted only after the entire sequence completes, reducing model-environment interactions and overall execution time.

Let s_t be the environment state at step t. The model first generates a thought h_t, which describes its plan for the next steps, and then outputs an action sequence

$$A_t = \{a_{t_1}, a_{t_2}, \ldots, a_{t_k}\}, \quad k \le K,$$

where a_i is an atomic action and K is the maximum sequence length. The environment executes A_t sequentially:

$$s_{t+1} = T(s_t, a_{t_1}, a_{t_2}, \ldots, a_{t_k}),$$

where T denotes the process that interprets the model's action output and applies the corresponding action in the environment. After A_t is finished, the model receives s_{t+1} and predicts the next pair (h_{t+1}, A_{t+1}).

3.2 Dataset Construction

We first attempt to use prompts to let the model output multiple actions. However, we found two

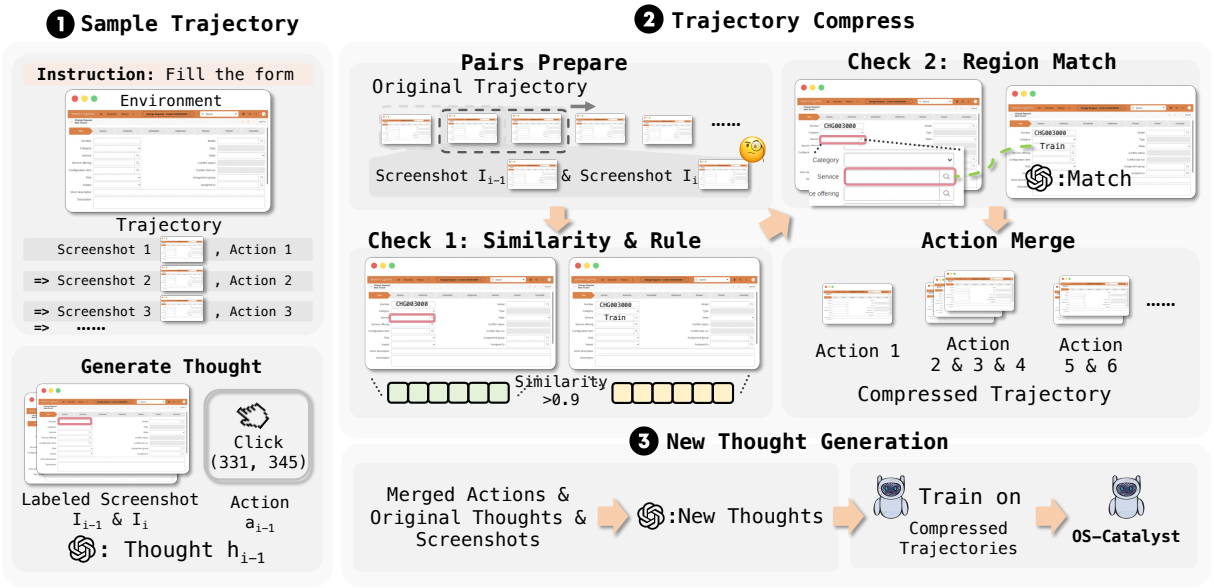


Figure 2: Data construction pipeline of OS-CATALYST. From sampled trajectories, we merge consecutive actions validated by similarity and region checks, regenerate thoughts, and fine-tune the model on the compressed trajectories.

major problems. First, the model had no awareness of producing action sequences and would not attempt multi-step actions within the current interface. Second, even when action sequences were produced, the accuracy was very low. Therefore, we construct a dataset for post-training to enhance the model's multi-step planning and action prediction ability. The complete dataset construction process is shown in Figure 2.

Raw Trajectory Collection. WorkArena (Drouin et al., 2024) is an enterprise software environment built on the ServiceNow[1] platform to evaluate GUI agents on realistic knowledge-work tasks such as form filling, list filtering, information retrieval, service catalog usage, and menu navigation. It provides multimodal observations (HTML, accessibility tree, screenshots) and automatic validation for task completion. In WorkArena, each type of task is defined by a task template. By randomly sampling the conditions and input values within a template, multiple task instances can be constructed. WorkArena provides a `cheat_function` that generates correct Playwright action trajectories for any task configuration, enabling reliable ground-truth trajectories for training.

We collect a total of 420 trajectories across 21 tasks. Each trajectory contains the sequence of environment states (including screenshots) along with the corresponding atomic actions (click, type, select, etc.) and their associated bounding boxes, forming the raw data for subsequent action sequence compression and training. Formally, let a trajectory be $\tau = \{(s_i, a_i)\}_{i=1}^{T}$, where s_i is the environment state (including screenshot I_i and other structural views) at step i, and a_i is the atomic action with its bounding box.

Thought Generation. Previous work has shown that explicitly modeling the reasoning process can improve the inference ability of GUI agents (Qin et al., 2025; Zhang et al., 2025c). Following this, we augment raw trajectories with a *thought* before each action that reflects the agent's consideration of what to do next.

To generate thought h_i for action a_i, we provide GPT-4o (Hurst et al., 2024) with screenshot pair (I_i, I_{i+1}) together with the executed action a_i. In the screenshots, the bounding box of the element involved in a_i is highlighted with a red rectangle to help the model identify the relevant interface element. The model then produces a natural language description that explains the intention of a_i by referring to how it changes the interface from s_i to s_{i+1}. The augmented trajectory is thus $\tilde{\tau} = \{(s_i, h_i, a_i)\}_{i=1}^{T}$.

Thus, each action is aligned with both its execution context and a reasoning thought, providing extra supervision to support task understanding.

[1] https://www.servicenow.com/

Trajectory Compression Pipeline. In GUI tasks, action sequences cannot be arbitrarily compressed, since some actions trigger interface changes that make it impossible to predict the next action without updated observation. To obtain trajectories with action sequences, we design a compression pipeline that transforms raw step-by-step trajectories into compressed ones, as shown in Figure 2. Let a raw trajectory be $\tau = \{(s_i, h_i, a_i)\}_{i=1}^{T}$ from WorkArena. We sequentially check whether adjacent steps can be merged. The prerequisite is that the first action does not affect the element involved in the second one. If the previous action includes navigating to another page or scrolling, the condition is not satisfied.

Pair preparation. From τ we build adjacent candidates consisting of $\mathcal{P} = (s_t, a_t, s_{t+1}, a_{t+1})$.

Check 1: Similarity & Rule. For each pair, we compute the Structural Similarity Index (SSIM) (Wang et al., 2004) between consecutive screenshots I_i and I_{i+1}. Only pairs with similarity greater than a threshold (0.9 in this work) are retained, filtering out major interface transitions. We also add a restriction that if the first action is a `scroll` or any operation that inevitably changes the page layout, the pair is filtered out.

Check 2: Region Match. For the remaining pairs, we perform local verification. The key criterion for compression is whether the first action changes the position or state of the element involved in the second action. To verify this, we use the bounding box b_{i+1} of the second action and crop the corresponding regions from screenshots I_i and I_{i+1}. We then query GPT-4o with these cropped regions and the action descriptions to decide whether the two actions can be safely merged. This step ensures that the element required by the second action remains stable and does not depend on intermediate model feedback.

Action Merge. We greedily merge consecutive actions as long as both checks pass, forming a compressed sequence $A_t = \{a_{t_1}, a_{t_2}, \ldots, a_{t_k}\}, k \le K$, with $K = 5$ as the maximum sequence length. The merged sequence is then stored in the compressed trajectory $\hat{\tau}$.

New Thought Generation. Since original thoughts h_i are tied to atomic actions, we regenerate a new thought $\hat{h}_t$ for each compressed sequence A_t. To do this, we prompt GPT-4o (Hurst et al., 2024) with the start and end screenshots (I_i, I_{i+k}), and the original thoughts. Based on this input, GPT-4o produces a concise description that explains the reasoning behind executing A_t. The final compressed trajectory is represented as $\hat{\tau} = \{(s_t, \hat{h}_t, A_t)\}_{j=1}^{M}$, $A_t = \{a_{t_1}, a_{t_2}, \ldots, a_{t_k}\}$, $|A_t| \le K$, which is then used for training.

Table 1: Statistics of our constructed datasets.

Dataset	#Traj.	Avg. Steps	Avg. Actions
Work-Step	420	19.00	1.00
Work-Seq	420	**7.58** (-33.8%)	**1.51** (+51.0%)

3.3 Dataset Statistics

The original WorkArena benchmark contains 25 task types. To clearly separate the training and test sets, we select 21 task types for training. For each task type, we collect 20 distinct trajectories using different random seeds, resulting in a total of 420 trajectories. Based on these raw trajectories, we construct two datasets that differ in how the actions are represented and organized.

Work-Step is built from the raw trajectories by adding a *thought* to each atomic action, as described in Section 3.2. The dataset keeps the original step-by-step format with reasoning information.

Work-Seq is built from Work-Step using the compression pipeline described in Section 3.2. In this process, consecutive actions are merged into an action sequence when the conditions are satisfied.

Table 1 summarizes the statistics of constructed datasets. Compared to Work-Step , Work-Seq significantly reduces the average step length to 12.6 (-33.8%), due to the increase of the average number of actions per step to 1.51 (+51.0%). Further details of the dataset can be found in Appendix D.

3.4 Training Strategy

To train models to generate coherent action sequences, we adopt a supervised objective that couples *thoughts* with *actions*. Unlike single-step prediction, sequence generation requires the model to decide whether subsequent actions are determinable from the current observation and unaffected by earlier actions. If so, they can be merged as a sequence; otherwise, the model should stop and wait for new observation. By training thoughts and actions together, we encourage the model to use thoughts for multi-step planning, enabling coherent action sequence prediction.

Context-Aware Formulation. Another design choice is to include recent interaction history. Us-

ing only the current screenshot leaves the model unaware of past actions, while conditioning on the full trajectory can exceed the context window and introduce noise. To balance this, we use the last $L = 5$ steps as context, capturing short-term dependencies (e.g., opening a menu before selecting an option). Each training instance conditions on the last L steps,

$$\mathcal{C}_t = \{(s_{t-L}, h_{t-L}, A_{t-L}), \ldots, (s_{t-1}, h_{t-1}, A_{t-1})\},$$

and the current state s_t, to predict both the next thought h_t and action sequence A_t.

Thought–Action Training. We model the joint generation of thought h_t and action sequence A_t as

$$p_\theta(h_t, A_t \mid I_t, \mathcal{C}_t) = \prod_{j=1}^{|h_t|} p_\theta(h_{t,j} \mid I_t, \mathcal{C}_t, h_{t,<j}) \times \prod_{m=1}^{|A_t|} p_\theta(A_{t,m} \mid I_t, \mathcal{C}_t, h_t, A_{t,<m}).$$

Training uses the standard next-token cross-entropy objective:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \Big(\sum_{j=1}^{|h_t|} \log p_\theta(h_{t,j} \mid I_t, \mathcal{C}_t, h_{t,<j}) + \sum_{m=1}^{|A_t|} \log p_\theta(A_{t,m} \mid I_t, \mathcal{C}_t, h_t, A_{t,<m}) \Big).$$

4 Experiment

4.1 Evaluation Benchmark

WorkArena. We evaluate our method on WorkArena (Drouin et al., 2024), a benchmark ideal for testing action-compression due to its focus on automating multi-step business tasks. WorkArena simulates repetitive, structured office workflows such as list operations, form filling, and service catalog tasks, where employees naturally execute predictable action sequences. This makes it well-suited for assessing our method's efficiency in generating multiple actions per turn, as completing tasks requires the model to plan and compress logical sequences into cohesive outputs.

Since the benchmark contains 25 task types and we use 21 types for training, the first test set (**WorkArena(Seen)**) contains tasks from the same 21 types but generated with different random seeds. This results in 84 distinct tasks that occur in the same scenarios as the training set, but differ in their specific requirements. The second test set (**WorkArena(Unseen)**) is built from the remaining 4 task types that are excluded from training. This set includes 16 tasks and serves to evaluate the generalization ability of the models to novel task types.

OSWorld. OSWorld (Xie et al., 2024a) evaluates computer-use agents across heterogeneous software environments (office tools, operating systems, browsers, developer applications), making it well-suited for assessing cross-domain generalization. We use OSWorld to test whether OS-CATALYST transfers beyond WorkArena.

4.2 Model Settings

UI-TARS. We use UI-TARS (Qin et al., 2025) as our model, a GUI agent that takes screenshots as input and produces human-like interactions (mouse clicks, keyboard typing, etc.). Unlike systems relying on prompt engineering or workflow wrappers, UI-TARS is an end-to-end model that unifies perception, grounding, reasoning, and action directly.

4.3 Baseline Construction

We consider four groups of models: (1) The original `UI-TARS-7B-DPO` and `UI-TARS-72B-DPO` with default prompting that predicts actions step by step. (2) The same UI-TARS models prompted to output action sequences on each page without additional training, denoted as **UI-TARS(prompt)**. (3) UI-TARS fine-tuned on the Work-Step dataset with step-by-step trajectories augmented with thoughts, denoted as **UI-TARS(Work-Step)**. (4) UI-TARS fine-tuned on the Work-Seq dataset with compressed action sequences and thoughts, i.e., **UI-TARS(Work-Seq)**.

4.4 Metrics

Task Success. We evaluate success rate using WorkArena's rule-based outcome validator. However, WorkArena tasks are long-tailed, often requiring over 30 steps, making them overly difficult for current GUI models. To better capture model performance under such challenging settings, we implement a Partial Success Rate (PSR) validator that reflects partial progress more fairly. For list tasks, we distribute 1.0 point evenly across input boxes in the filter panel. For form tasks, we allocate 0.2 points for correct submission and distribute the remaining 0.8 across all required items. For service catalog tasks, we allocate 0.3 points for navigation,

Table 2: Evaluation results on WorkArena(Seen). The first three metrics are success rate (SR), partial success rate (PSR), number of action per step (#A/S). We use UI-TARS(Work-Step) as the baseline to compute the relative changes of efficiency metrics for UI-TARS(Work-Seq). For SR, PSR, and efficiency metrics, we highlight the best (bold) and second-best (underline) results.

Model	SR	PSR	#A/S	Step Time (s)	Task Time (s)
7B Models					
UI-TARS-7B-DPO	0.036	0.072	1.00	11.66	580.80
UI-TARS(prompt)	0.024	0.072	1.00	7.95	<u>381.02</u>
UI-TARS(Work-Step)	**0.095**	<u>0.267</u>	<u>1.00</u>	12.00	291.41
UI-TARS(Work-Seq)	<u>0.083</u>	**0.277**	**1.33** (+33.0%)	9.20	**147.90** (-49.2%)
72B Models					
UI-TARS(Work-Step)	**0.079**	<u>0.210</u>	<u>1.00</u>	15.19	<u>389.41</u>
UI-TARS(Work-Seq)	<u>0.060</u>	**0.294**	**1.20** (+20.0%)	13.02	**308.50** (-20.8%)

0.1 for submission, and distribute the remaining 0.6 across requested configurations.

Efficiency Metrics We also report three metrics that measure execution efficiency. A/S (actions per step) denotes the average number of valid actions per model output, excluding steps with no executable action. Step Time (s) measures the average latency per output, while Task Time (s) reflects the total time to finish a task, accumulating both step-level latencies and step count.

5 Main Result and Analysis

5.1 How Does OS-CATALYST Improve Efficiency?

WorkArena(Seen) Results. For both 7B and 72B settings, Work-Seq trained models achieve notable efficiency improvements over Work-Step. Specifically, UI-TARS(Work-Seq) reduces average task time from 291.4s to 147.9s (7B, 50% reduction) and from 389.4s to 308.5s (72B), while maintaining comparable success rates. Compared to the base model and prompt-only variant, which almost never output multiple actions per step (A/S = 1.0), UI-TARS(Work-Seq) achieves 1.33 (7B) and 1.20 (72B) actions per step, reducing model-environment interactions and substantially shortening task duration.

WorkArena(Unseen) Results. On the Unseen set with four untrained task, UI-TARS(Work-Seq) maintains efficiency advantages. For 7B models, it reduces average task time to 135.1s versus 157.4s for UI-TARS(Work-Step) and over 300s for the base model. For 72B models, task time decreases from 405.9s to 360.3s. UI-TARS(Work-Seq) also achieves higher average actions per step (1.28 for 7B and 1.25 for 72B) on unseen tasks, indicating that action sequence prediction generalizes beyond training tasks and yields consistent efficiency gains.

Action Sequence Type. We analyze the distribution of action sequence types produced by UI-TARS(Work-Seq). As shown in Figure 3, most sequences have two actions, with some successful cases containing three or four. The most frequent patterns are `[click, type]` and `[click, click]`, corresponding to common GUI interactions like selecting a field then typing, or navigating menus through consecutive clicks. Other action sequence types also carry concrete meaning, such as `[click, type, click]` for filling in a field followed by confirmation, and `[click, type, click, type]` for completing two consecutive fields in a form. Overall, this distribution suggests that OS-CATALYST enables the model to generate action sequences in a way that reflects common interaction patterns observed in real GUI tasks. Examples of action sequences in Appendix C.1.

5.2 How Does OS-CATALYST Perform on Task Success Rate?

On the Seen set, UI-TARS(Work-Seq) achieves success rates that are close to those of UI-TARS(Work-Step), with 0.083 vs. 0.095 for the 7B models and 0.060 vs. 0.079 for the 72B models. This shows that UI-TARS(Work-Seq) method does not compromise the ability to complete tasks. Additionally, UI-TARS(Work-Seq) consistently yields the highest partial success rates, reaching 0.277 (7B) and 0.294 (72B). Compared with the base UI-TARS models without additional training, both fine-tuned variants achieve higher SR and PSR, suggesting that training improves the model's understanding of the WorkArena environment. On the Unseen

Table 3: Evaluation results on WorkArena(Unseen). We use UI-TARS(Work-Step) as the baseline to compute the relative changes of efficiency metrics for UI-TARS(Work-Seq). For SR, PSR, and efficiency metrics, we highlight the best (bold) and second-best (underline) results. Step Time values are reported without relative changes.

Model	SR	PSR	A/S (non-empty)	Step Time (s)	Task Time (s)
7B Models					
UI-TARS-7B-DPO	**0.063**	0.125	1.00	6.35	309.80
UI-TARS(prompt)	**0.063**	0.100	1.00	10.84	500.48
UI-TARS(Work-Step)	0.000	**0.157**	1.00	9.12	157.38
UI-TARS(Work-Seq)	**0.063**	0.129	**1.28** (+28.0%)	7.41	**135.15** (-14.1%)
72B Models					
UI-TARS(Work-Step)	**0.063**	**0.205**	1.00	20.2	405.9
UI-TARS(Work-Seq)	**0.063**	0.180	**1.25** (+25.0%)	14.7	**360.3** (-11.2%)

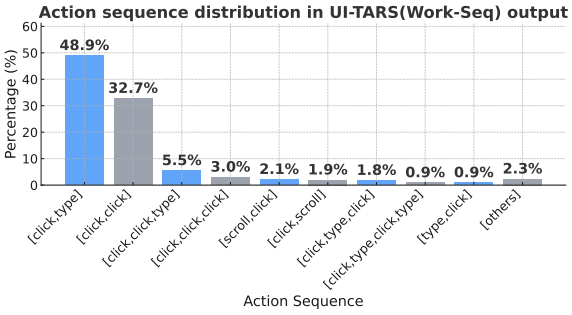


Figure 3: Action-sequence distribution in model output

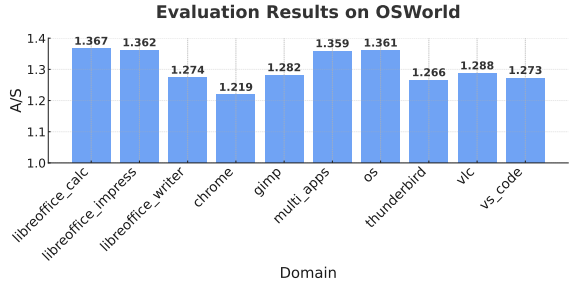


Figure 4: Evaluation Results on OSWorld

set, UI-TARS(Work-Seq) achieves comparable success rates to the baselines, with 0.063 SR for both 7B and 72B models. It also sustains strong partial success rates (0.129 and 0.180), showing that OS-CATALYST generalizes to new task types. Overall, these results show that OS-CATALYST improves efficiency without reducing task success, and its ability to predict action sequences transfers to tasks outside the training set.

5.3 How Does OS-CATALYST Perform on Cross-Domain Settings?

To assess generalization beyond the WorkArena environment, we further evaluate OS-CATALYST on OSWorld. Figure 4 reports the average actions per step (A/S) across task domains. OS-CATALYST consistently maintains non-trivial action-sequence prediction ability across unseen environments, achieving A/S values ranging from 1.219 (`Chrome`) to 1.367 (`LibreOffice_Calc`). Task domains with more frequent page transitions tend to yield lower A/S values (e.g., `chrome`, `thunderbird`), as such patterns limit multi-action opportunities within a single interface state. This aligns with human annotation statistics reported in Table 3 of OSWorld-Human (Abhyankar et al., 2025), where grouped steps in several LibreOffice-related tasks show substantially greater reduction compared to single-step interactions, suggesting higher inherent compression potential in office-style environments.

These results suggest that OS-CATALYST can transfer its adaptive compression capability to heterogeneous GUI environments without retraining, showing potential robustness to domain shifts.

6 Conclusion

We propose OS-CATALYST, a method that improves the efficiency of computer-using agents through adaptive action compression. By allowing models to predict multiple consecutive actions from a single observation, OS-CATALYST reduces redundant model–environment interactions and shortens overall task duration. To enable this capability, we construct two datasets in the WorkArena environment, supporting both step-level interaction and compressed action sequences. Experiments show that OS-CATALYST achieves up to a 50% reduction in task execution time while maintaining comparable task success rates, highlighting the potential of sequence-level action prediction as a new approach for GUI agents. In future work, we hope this approach can generalize to broader application scenarios, further advancing the development of efficient and practical GUI agents.

Limitations

While the environment, benchmark, and method proposed in this work demonstrate the potential to advance the safety research of mobile GUI agents, it is important to acknowledge some limitations:

Verifier Dependency. In our hybrid method, our Formal Verifier relies on obtaining system state traces, which are currently accessible only on open platforms such as Android. This makes the approach less directly applicable to closed environments such as iOS. Nevertheless, we believe that such ideas could be adapted and extended to other platforms according to practical needs.

Environment. We construct as a simulated environment and derive a frozen dataset from it to form . While our experiments demonstrate strong closeness between the live and frozen settings, certain discrepancies inevitably remain, for example, random push notifications under online network conditions. However, we believe these differences do not undermine the general conclusions of our study, and future work can be expected to further reduce such gaps.

Broader Impacts

Computer agents operating in an OS environment may potentially interfere with the normal functioning of a system. In this work, however, all experiments are conducted within controlled virtual environments, which eliminates risks to real devices or user accounts. The instructions and trajectories used in our study are released solely for research purposes, and we encourage interested researchers to conduct experiments using our provided environment or benchmark rather than applying them to their own devices or personal accounts. This precaution is intended to avoid any unintended harm or irreversible consequences to real systems and communities.

References

Reyna Abhyankar, Qi Qi, and Yiying Zhang. 2025. Osworld-human: Benchmarking the efficiency of computer-use agents. *arXiv preprint arXiv:2506.16042*.

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

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

Anthropic. 2025. Claude 3.7 sonnet. https://www.anthropic.com/news/claude-3-7-sonnet. Accessed: 2025-04-25.

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

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2024a. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. *arXiv preprint arXiv:2410.08115*.

Xuetian Chen, Hangcheng Li, Jiaqing Liang, Sihang Jiang, and Deqing Yang. 2024b. Edge: Enhanced grounded gui understanding with enriched multi-granularity synthetic data. *arXiv preprint arXiv:2410.19461*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. 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)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, et al. 2025. Multi-agent collaboration via evolving orchestration. *arXiv preprint arXiv:2505.19591*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pages 28091–28114. Curran Associates, Inc.

Alexandre Drouin, Jorge Gonzalez, Jonathan Krause, Besmira Nushi, Mitchell Wortsman, Ce Zhang, Xiaohua Zhou, Ingrid Zukerman, and Zhiwei Steven Chen. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research. PMLR.

Zhiqi Ge, Juncheng Li, Xinglei Pang, Minghe Gao, Kaihang Pan, Wang Lin, Hao Fei, Wenqiao Zhang, Siliang Tang, and Yueting Zhuang. 2024. Iris: Breaking gui complexity with adaptive focus and self-refining. *arXiv preprint arXiv:2412.10342*.

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

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Chengyou Jia, Minnan Luo, Zhuohang Dang, Qiushi Sun, Fangzhi Xu, Junlin Hu, Tianbao Xie, and Zhiyong Wu. 2024. Agentstore: Scalable integration of heterogeneous agents as specialized generalist computer assistant. *arXiv preprint arXiv:2410.18603*.

Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. 2025. Appagentx: Evolving gui agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*.

Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Prasad Sathya Moorthy, Jeff Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui 2: Mastering universal user interface understanding across platforms. *arXiv preprint arXiv:2410.18967*.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025a. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*.

Yuhang Liu, Zeyu Liu, Shuanghe Zhu, Pengxiang Li, Congkai Xie, Jiasheng Wang, Xueyu Hu, Xiaotian Han, Jianbo Yuan, Xinyao Wang, et al. 2025b. Infigui-g1: Advancing gui grounding with adaptive exploration policy optimization. *arXiv preprint arXiv:2508.05731*.

Zhaoyang Liu, Jingjing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, Shenglong Ye, Qingyun Li, Xuan Dong, Yue Yu, Chenyu Lu, YunXiang Mo, Yao Yan, Zeyue Tian, Xiao Zhang, Yuan Huang, Yiqian Liu, Weijie Su, Gen Luo, Xiangyu Yue, Biqing Qi, Kai Chen, Bowen Zhou, Yu Qiao, Qifeng Chen, and Wenhai Wang. 2025c. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*. Preprint.

Manus. 2025. Manus. https://manus.im/. Accessed: 2025-04-25.

OpenAI. 2025a. Computer-using agent: Introducing a universal interface for ai to interact with the digital world.

OpenAI. 2025b. Introducing operator. https://openai.com/index/introducing-operator/. Accessed: 2025-04-25.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. 2024. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*.

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

Jinwei Su, Yinghui Xia, Qizhen Lan, Xinyuan Song, Yang Jingsong, Lewei He, and Tianyu Shi. 2025. Difficulty-aware agent orchestration in llm-powered workflows. *arXiv preprint arXiv:2509.11079*.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. 2024a. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. 2024b. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*.

Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, et al. 2025. Scienceboard: Evaluating multimodal autonomous agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*.

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

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710.

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

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Hang Wu, Hongkai Chen, Yujun Cai, Chang Liu, Qingwen Ye, Ming-Hsuan Yang, and Yiwei Wang. 2025a. Dimo-gui: Advancing test-time scaling in gui grounding via modality-aware visual reasoning. *arXiv preprint arXiv:2507.00008*.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. 2025b. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *Preprint*, arXiv:2402.07456.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2025c. OS-ATLAS: Foundation action model for generalist GUI agents. In *The Thirteenth International Conference on Learning Representations*.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024a. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024b. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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

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

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2024. Aria-ui: Visual grounding for gui instructions. *arXiv preprint arXiv:2412.16256*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Xiao Yu, Baolin Peng, Ruize Xu, Michel Galley, Hao Cheng, Suman Nath, Jianfeng Gao, and Zhou Yu. 2025. Dyna-think: Synergizing reasoning, acting, and world model simulation in ai agents. *Preprint*, arXiv:2506.00320.

Chaoyun Zhang, He Huang, Chiming Ni, Jian Mu, Si Qin, Shilin He, Lu Wang, Fangkai Yang, Pu Zhao, Chao Du, et al. 2025a. Ufo2: The desktop agentos. *arXiv preprint arXiv:2504.14603*.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024a. Ufo: A ui-focused agent for windows os interaction. *Preprint*, arXiv:2402.07939.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025b. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20.

Jiwen Zhang, Yaqi Yu, Minghui Liao, Wentao Li, Jihao Wu, and Zhongyu Wei. 2024b. Ui-hawk: Unleashing the screen stream understanding for gui agents.

Junlei Zhang, Zichen Ding, Chang Ma, Zijie Chen, Qiushi Sun, Zhenzhong Lan, and Junxian He. 2025c. Breaking the data barrier–building gui agents through task generalization. *arXiv preprint arXiv:2504.10127*.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. 2024. Swift:a scalable lightweight infrastructure for fine-tuning. *Preprint*, arXiv:2408.05517.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024b. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The*

Twelfth International Conference on Learning Representations.

A Large Language Model Usage

In this submission, we leverage LLMs to support and refine the writing process, including grammar and typo correction, and the identification of related work.

B Experimental Details

B.1 Environment Selection

Among the available GUI benchmarks with diverse features, we selected WorkArena for our experiments. This choice was motivated by the fact that WorkArena tasks generally require a relatively higher number of steps too complete. Moreover, the office scenario naturally lends itself to sequential action execution, making it well-suited for observing how models learn to perform multi-step operations. Following WorkArena, the same team introduced WorkArena++, which incorporates complementary tasks along with more fundamental interactions. However, we found WorkArena++ to be excessively challenging—tasks often exceed 100 steps in length, and preliminary tests showed that both GPT-4o and GPT-4o-v achieved near-zero success rates. Consequently, we decided not to adopt WorkArena++ for this study.

Action	Definition	Parameter
`click`	Clicks at coordinates.	`start_box`
`left_double`	Double-clicks at coordinates.	`start_box`
`right_single`	Right-clicks at coordinates.	`start_box`
`drag`	Drags from start to end.	`start_box`, `end_box`
`hotkey`	Presses a keyboard shortcut.	`key`
`type`	Types specified content.	`content`
`scroll`	Scrolls in the given direction.	`start_box`, `direction`
`wait`	Pauses for 5s.	/
`finished`	Marks the task as complete.	/
`call_user`	Requests user intervention.	/

Table 4: Action space with definitions and parameters.

B.2 Action Space

We follow the action space design of UI-TARS, while adapting it to our model and dataset. In particular, the action space of the model includes `click`, `left_double`, `right_single`, `drag`, `hotkey`, `type`, `scroll`, `wait`, `finished`, and `call_user`. The definition and parameter are shown in Table 4.

B.3 Fine-Tuning Setup.

We apply the training strategy in Section 3.4 to fine-tune the base models. For the `UI-TARS-7B-DPO` model, we adopt full SFT for 4 epochs using the ms-swift (Zhao et al., 2024) framework, with a learning

rate of 1×10^{-4}. For the UI-TARS-72B-DPO model, we adopt LoRA-based SFT with rank 8 and train for 4 epochs with the learning rate of 1×10^{-5}, as full SFT is infeasible under our resource constraints. Here we use LLaMA-Factory (Zheng et al., 2024b) framework for lora fine-tuning.

C Case Study

In this section, we present representative cases to illustrate the behavior of UI-TARS(Work-Seq). We include both success and failure examples to show how the model generates action sequences in practice.

C.1 Success Case Examples

Figure 5 and Figure 6 are two success case that demonstrate our model's ability to output consecutive actions. In both examples, the model correctly follows the task instructions and current page state to generate coherent sequences of 3–4 actions.

C.2 Failure Case Examples

We further examine representative failure cases of our model. As shown in Figure 7–9, they can be grouped into three categories: (1) over-compression, where the model outputs an excessively long action sequence beyond what is feasible for the current state; (2) under-compression, where the model fails to merge actions even though multiple steps could safely be combined; (3) incorrect element localization, where the target referenced in the thought is inconsistent with the executed coordinates; These cases illustrate the challenges that remain for robust multi-action planning in GUI environments, and addressing them constitutes an important direction for future work.

While these limitations remain, OS-CATALYST has already led to substantial efficiency improvements over previous methods, reducing overall task time by approximately **50%** and decreasing the average number of interaction steps by **33%**.

D Dataset Details

In this section, we provide additional details of the datasets used for training in OS-CATALYST. As described in Table 1, our data consists of two subsets: **Work-Step** and **Work-Seq**, both constructed within the WorkArena benchmark environment. Each dataset is designed to support the development of GUI agents from both step-level interaction and action-sequence perspectives.

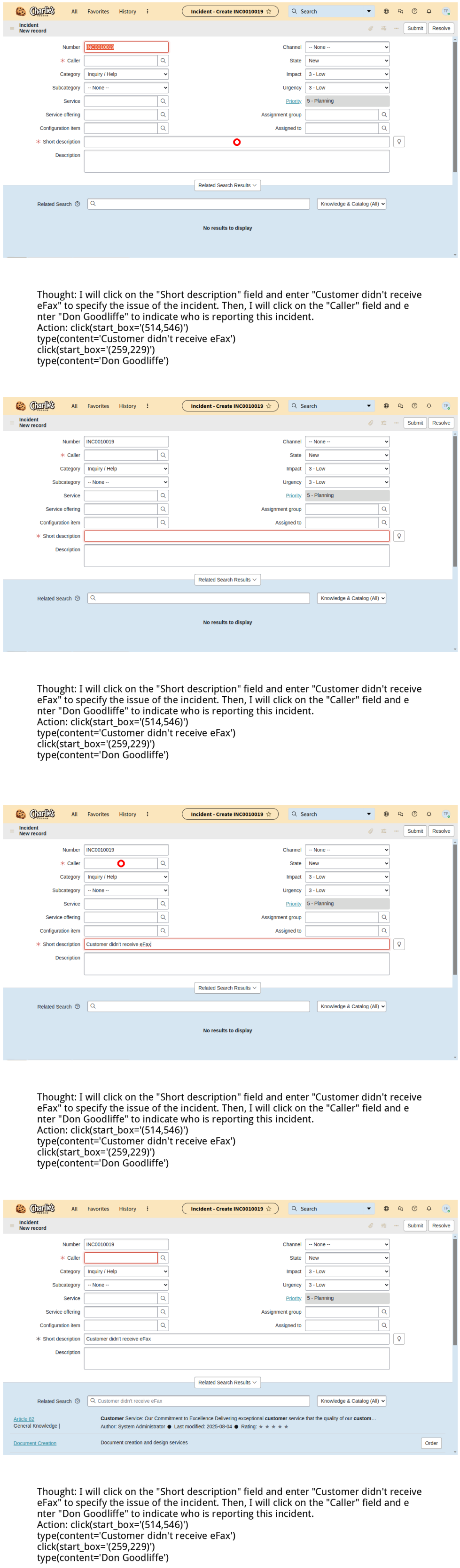


Figure 5: The task is filling up a form. The model outputs a succession of four actions, filling up two items in a row.

Data Structure. Each trajectory includes:

- **Screenshots** of the interface at each interaction step.
- **Ground-truth thoughts** that describe the agent's intention and intermediate reasoning.
- **Actions** (e.g., `click`, `type`, `scroll`) along with their coordinates and content.
- **Bounding boxes** specifying the UI elements involved in each action.

Data Format. The dataset is provided as images and JSON records:

- `*.png`: High-resolution GUI screenshots (1920×1080).
- `*.json`: Structured metadata containing thoughts, action definitions, coordinates, and bounding boxes.

Licensing and Usage. The dataset will be released under the **MIT License** and can be used for non-commercial academic research, including model training, benchmarking, and GUI agent automation studies. It permits redistribution and modification with proper attribution.

E Prompts

E.1 Model Prompts

Original prompt that does not require model to output multiple actions.

```
You are a GUI agent. You are given a task
    and your action history, with
    screenshots. You need to perform the
    next action to complete the task.

## Output Format
‘‘‘
Thought: ...
Action: ...
‘‘‘

## Action Space
{action_space}

## Note
- Use {language} in ‘Thought‘ part.
- Summarize your next action (with its
    target element) in one
sentence in ‘Thought‘ part.

## User Instruction
{instruction}
```

Updated prompt that requires model to output multiple actions.

```
You are a GUI agent. You are given a task
    and your action history, with
    screenshots. You need to perform the
    next action(s) to complete the task.

If multiple actions can be performed
    independentlymeaning one action does
    not interfere with another in terms of
    position or elementsyou should output
    them together in a single ‘Action‘
    block, separated by two newlines
    (‘\n\n‘).

## Output Format
‘‘‘
Thought: ...
Action: ...
‘‘‘

## Action Space
{action_space}

## Note
- Use {language} in ‘Thought‘ part.
- Summarize all upcoming actions (with
    their target elements) in ‘Thought‘
    part.
- In the ‘Action‘ section, include one or
    more actions, each on its own line,
    separated by two newlines.
- Only include multiple actions if they are
    **logically and spatially
    independent**.

## User Instruction
{instruction}
```

E.2 Data Curation Prompts

Prompt for generating thought.

```
You are a GUI agent that specializes in
    reverse-engineering the intent behind
    GUI actions.

You will be given a step from an
    interaction trajectory. Each step
    includes:
- the global instruction to complete,
- the previous actions taken,
- the current action to analyze. (If the
    current action involves a coordinate,
    the coordinates are normalized values:
    absolute coordinates divided by the
    original image width or height, then
    multiplied by 1000),
- the UI screenshot with the red bounding
    box indicating the position of the
    action to help you identify the
    element involved in the action,
- the UI screenshot after the action is
    executed,
```

```
Your job is to identify the element in the
    action and infer the *thought* (i.e.,
    a small plan or rationale) behind the
    current action, and then output it in
    the following format:

Thought: {{<thought>}}

The thought should be a small plan and
    summarize this action in future tense
    (with its target element).
The thought must be consistent with the
    global instruction and current action.
The thought should be a plan in a single
    sentence in first-person perspective,
    and it should not include any code or
    action.
If the current action is none, and the
    relevant element is already set to the
    correct default that satisfies the
    instruction, the thought should state
    that the default option already meets
    the instruction and no further action
    is needed.

--- INPUT ---

Instruction: {instruction}

Previous actions: {previous_actions if
    previous_actions else "None"}

Current action: {current_action}

Current screenshot:
```

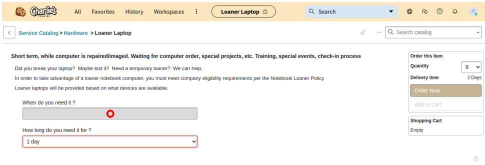
Prompt for judging whether two action can be done in one step

```
You are given two cropped images of GUI
    elements. Each image corresponds to
    the same position in two consecutive
    screenshots from a GUI task execution.

Your task is to determine whether the two
    images represent the same GUI element
    --- that is, the same underlying
    component such as a button, icon, text
    label, or menu item --- even if there
    are slight visual differences caused
    by rendering, state changes (e.g.\
    hover or focus), or animations.

Minor differences in appearance should not
    affect your decision, as long as the
    core identity of the element remains
    the same.

Write your reasoning step by step. Then
    give your final answer as ‘‘yes’’ or
    ‘‘no’’ on the last line. (‘‘yes’’
    means both images show the same GUI
    element.)
```

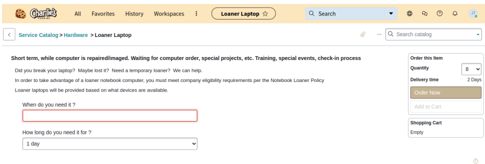
```
The first element:
{image1}
The second element:
{image2}
```

Prompt for merging thoughts of multiple actions

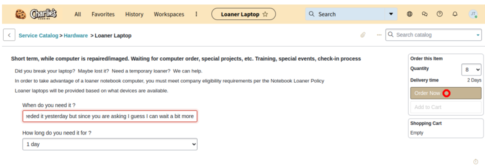
```
In the original GUI task setup, the model
    performs step-by-step inference: it
    generates a thought and action,
    receives an updated screenshot, and
    then proceeds with the next thought
    andaction. The following is a sequence
    of several consecutive thought-action
    steps from that setting and
    corresponding screenshots.

Now, we want the model to output all
    actions in a single step. Your task is
    to merge the multiple thoughts into
    one coherent and concise thought, as
    if the model planned the entire
    sequence of actions without receiving
    any updated screenshots in between.

While doing this, remove any reasoning or
    statements that only exist due to
    intermediate screenshots. The final
    thought should reflect a continuous
    reasoning process that naturally leads
    to the full sequence of actions
    without any interruptions.

## Output Format
You should output the merged thought
    directly in your response, without any
    additional text or formatting. The
    output should be a single string that
    combines all individual thoughts into
    one coherent and unified thought.

## Previous Thoughts
```



Thought: I will enter the text "I needed it yesterday but since you are asking I guess I c
an wait a bit more" into the "When do you need it?" field to specify my intent and ensure
it aligns with the order requirements. Then, I will click on the "Order Now" button to fin
alize and submit the order for 8 "Loaner Laptop" with the specified configuration.
Action: click(start_box='(224,411)')
type(content='I needed it yesterday but since you are asking I guess I can wait a bit more')
click(start_box='(918,326)')

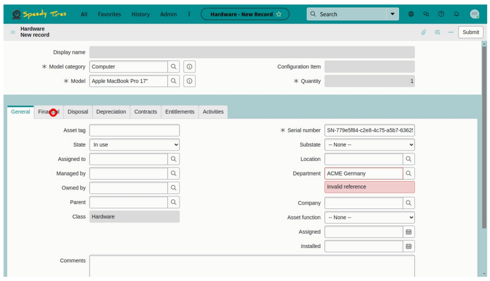


Thought: I will enter the text "I needed it yesterday but since you are asking I guess I c
an wait a bit more" into the "When do you need it?" field to specify my intent and ensure
it aligns with the order requirements. Then, I will click on the "Order Now" button to fin
alize and submit the order for 8 "Loaner Laptop" with the specified configuration.
Action: click(start_box='(224,411)')
type(content='I needed it yesterday but since you are asking I guess I can wait a bit more')
click(start_box='(918,326)')



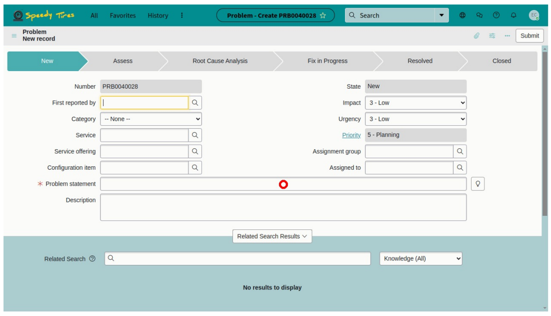
Thought: I will enter the text "I needed it yesterday but since you are asking I guess I c
an wait a bit more" into the "When do you need it?" field to specify my intent and ensure
it aligns with the order requirements. Then, I will click on the "Order Now" button to fin
alize and submit the order for 8 "Loaner Laptop" with the specified configuration.
Action: click(start_box='(224,411)')
type(content='I needed it yesterday but since you are asking I guess I can wait a bit more')
click(start_box='(918,326)')

Figure 6: The task is to order a loaner laptop. The model output a succession of three actions, filling up the last item and clicked the submit button

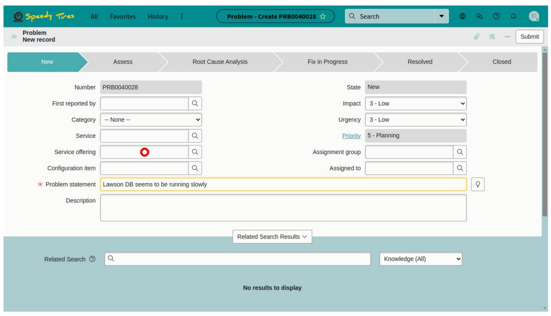


Thought: To complete the task, I will first click on the "Asset function" dropdown to sele
ct the appropriate value. Then, I will proceed to click on the "Financial" tab to enter fi
nancial details related to the new hardware asset. Next, I will click on the "GL account"
field and input "prerequire dinglebird nonchalky misadvis," fulfilling this specific detai
l requirement. Finally, I will switch back to the "General" tab if necessary to continue e
ntering other required information.
Action: click(start_box='(745,644)')
click(start_box='(103,396)')
click(start_box='(766,515)')
type(content='prerequire dinglebird nonchalky misadvis')
click(start_box='(36,396)')

Figure 7: In this form filling task, the model clicks to navigate to the *Financial* subpage. After this transition, it should stop and wait for the environment to return the new page before predicting subsequent actions. However, the model continues to output additional actions prematurely.



Thought: I will click on the "Problem statement" field and enter the value "Lawson DB seem
s to be running slowly" to accurately describe the issue as specified in the instruction.
Action: click(start_box='(514,585)')
type(content='Lawson DB seems to be running slowly')



Thought: I will click on the "Configuration item" field and enter the value "lawson_db" to
fulfill the requirement for this problem record.
Action: click(start_box='(259,479)')
type(content='lawson_db')

Figure 8: In this form filling example, the model fills in the two form fields across two separate steps, whereas the task can actually be completed in a single step with four consecutive actions.

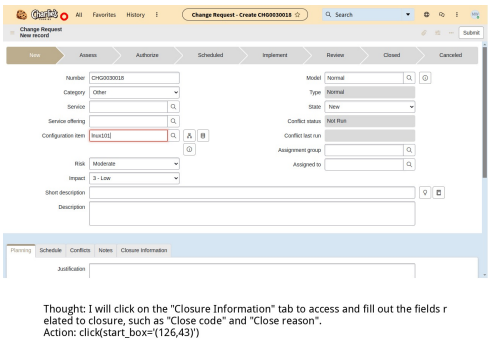


Figure 9: In this example, the correct plan for the model is to click on the *closure information* field. However, the predicted coordinates (marked with a red circle) are far from the correct location.