BUILDING AUTONOMOUS GUI NAVIGATION VIA AGENTIC-Q ESTIMATION AND STEP-WISE POLICY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in Multimodal Large Language Models (MLLM) have substantially driven the progress of autonomous agents for Graphical User Interface (GUI). Nevertheless, in real-world applications, GUI agents often face non-stationary environments, leading to high computational costs for data curation and policy optimization. In this paper, we introduce a novel MLLM-centered framework for GUI agents, which consists of two components: agentic-Q estimation and step-wise policy optimization. The former one aims to optimize a Q-model that can generate step-wise values to evaluate the contribution of a given action to task completion. The latter one takes step-wise samples from the state-action trajectory as inputs, and optimizes the policy via reinforcement learning with our agentic-Q. Our framework enjoys two favorable advantages: (i) all trajectories are produced by the policy itself, so that data collection costs are manageable; (ii) the policy update is decoupled from the environment, ensuring stable and efficient optimization. Empirical evaluations show that our framework endows Ovis2.5-9B with powerful GUI interaction capabilities, achieving remarkable performances on navigation and grounding benchmarks and even surpassing contenders with larger scales.

1 INTRODUCTION

Multimodal Large Language Models (MLLM) have drawn increasing attention for their powerful perceptual and reasoning capabilities (OpenAI, 2024; Wu et al., 2024b; Lu et al., 2025; Bai et al., 2025; Seed, 2025a; Google, 2025). These capabilities lay the foundation for developing autonomous agents that can interact with Graphical User Interfaces (GUI) to accomplish complex real-world tasks (Zhang et al., 2024; Nguyen et al., 2024; Tang et al., 2025b). Early efforts in developing GUI agents primarily process structured interface descriptions (e.g., HTML), which significantly differs from human-computer interaction, and fails to generalize to novel scenarios without structured data (Gur et al., 2023; Lai et al., 2024). For this reason, recent research has shifted toward *native screen-based GUI agents* that operate directly on raw screen visuals (Anthropic, 2024; OpenAI, 2025).

Despite the growing progress, building advanced screen-based GUI agents is faced with the challenges of *costly data curation*, *sparse step-wise supervision*, and *unstable optimization*. Specifically, first, empowering MLLMs with competent GUI navigation capabilities typically requires substantial amounts of high-quality interaction trajectories (Wang et al., 2024; Wu et al., 2024a; Pahuja et al., 2025). Different from conventional image-text data, these trajectories must capture not only the environmental states but also the detailed reasoning and corresponding actions. Therefore, constructing such high-quality trajectories relies heavily on expert involvement, leading to high costs in data organization and annotation. Second, completing a task in GUI environments typically demands multi-round interactions, resulting in long-horizon trajectories where feedback is only available at the final step. As a result, supervision information tends to be sparse and delayed, providing limited guidance for optimizing intermediate steps within the trajectory (Zhang et al., 2025; Tang et al., 2025a). Third, existing strategies for training GUI agents typically employ the Reinforcement Learning with Verifiable Rewards (RLVR) framework (Luo et al., 2025; Wang et al., 2025a). However, due to the delayed nature of supervision information, the policy optimization must remain tightly coupled with the GUI agent execution environments, which are often non-stationary and thereby, lead to low training efficiency and unstable policy optimization.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To handle these challenges, we introduce a novel framework for native screen-based GUI agents that enables manageable data curation, and supports stable and efficient optimization. Different from the commonly-used RLVR framework, the key is to train an agentic-Q model, and employ it to evaluate each step-wise action. In this way, on the one hand, we alleviate the issue of sparse supervision; on the other hand, we decouple policy optimization from the environment, thereby mitigating the impact of environmental non-stationarity on the training. Our framework incorporates two components:

- **Agentic-Q Estimation.** Our agentic-Q model is employed to evaluate an individual action given the associated state in GUI environments. To this end, we begin by collecting a set of self-generated state–action trajectories, and propagate the final supervision back to each intermediate step along the trajectory. These step-level annotations are then used to train our agentic-Q in a binary classification manner. To ensure the Q-model generalizes well, we incorporate *sliding-window* and *action-focus* strategies during training, which are empirically verified to mitigate potential entropy collapse when the agentic-Q model is subsequently used to guide policy optimization;
- **Step-Wise Policy Optimization.** With the agentic-Q model, we can conveniently leverage step-wise trajectories generated by the policy itself for fine-grained policy optimization. Specifically, at each step, the agentic-Q model evaluates the action taken by the current policy under the corresponding state, which is then employed to refine the policy via reinforcement learning. Algorithmically, we adopt step-wise variants of critic-free methods, i.e., GRPO (Shao et al., 2024), RLOO (Ahmadian et al., 2024), and REINFORCE++ (Hu et al., 2025), which are integrated with *action-level return-level filtering*, and *adaptive group weighting*, to ensure stable optimization.

We highlight that the feasibility of our agentic-Q estimation is grounded in the natural interaction patterns of GUI agents, which differ substantially from other reasoning-decision tasks, e.g., mathematical solving and code generation. To be specific, GUI agents operate in multi-turn interactive settings, where both state transitions and actions are explicitly defined and observable. Each state directly reflects the visual feedback of the environment (i.e., the webpage layout), so that the observation space is structurally organized and semantically informative. Moreover, the action space is finite and task-specific, typically limited to a small set of atomic operations such as *clicking*, *scrolling* and *typing*, which inherently supports tractable agentic-Q estimation. Notably, our framework enjoys two favorable advantages: (i) all state-action trajectories used for training are self-generated by the policy through interactions with the GUI environment, without the need for costly expert annotations ; (ii) since the agentic-Q model offers intermediate supervision at each decision step, policy updates do not rely on delayed trajectory-level feedback. As a result, policy optimization is fully decoupled from non-stationary execution environment, thereby avoiding the instability and inefficiency.

Empirically, to evaluate performance in realistic scenarios, we build a real-time environment where policies can interact with live websites, and employ WebVoyager (He et al., 2024) and Online-Mind2Web (Xue et al., 2025) as benchmarks, both of which comprise tasks constructed from real-world websites. Additionally, we also include ScreenSpot (Cheng et al., 2024) to assess the fundamental grounding capability of GUI agents. We employ Ovis2.5-9B (Lu et al., 2025) as the base model, and use it to generate state–action trajectories through interactions with websites from WebVoyager. Overall, experimental results demonstrate that our framework delivers remarkable performances, achieving state-of-the-art results among models of similar size (e.g., Qwen3-VL-8B and UI-TARS 1.5-7B) and also remaining competitive with significantly larger frontier models (e.g., GPT-4o and Claude 3.7/4 Sonnet CU). Additionally, since the training data only includes websites from WebVoyager, the results on Online-Mind2Web actually reflect out-of-distribution performances, and our models also achieve superior performances, highlighting the powerful generalization capability.

2 PRELIMINARIES

Formulation. The GUI navigation can be formulated as a Markov Decision Process (MDP):

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$$

with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function $\mathcal{P}$ and a reward oracle $\mathcal{R}$. Here, we consider the undiscounted setting with a discount factor $\gamma = 1$. Specifically, at each round i, the agent π_θ parameterized by θ receives a state s_i from the space $\mathcal{S}$, and chooses an action $a_i \in \mathcal{A}$ based on its own thought t_i. After that, the environment updates the state to $s_{i+1} \sim \mathcal{P}(s_{i+1}|s_i, t_i, a_i)$ so that the agent π_θ can select the action a_{i+1} for the next round. This process is repeated until the agent

π_θ considers the task complete or reaches a terminal state. At the end, the agent π_θ receives a reward r from $\mathcal{R}$ that indicates whether the task successfully finishes.

In this framework, (i) the state $s_i \in \mathcal{S}$ includes a task query, user instructions, historical interactions and the screenshot at round i; (ii) the action a_i is chosen from a discrete and finite space $\mathcal{A}$ that spans common web-use operations, such as *left-click* and *scroll*. Detailed descriptions can be found in Appendix C; (iii) the thought t_i is a text produced by π_θ that summarizes the reasoning and planning for action a_i. We define the *step* of the iterative process as a tuple (s_i, t_i, a_i, r_i) at the round i. Then, the whole trajectory with total rounds of T can be represented as:

$$\zeta = \{(s_1, t_1, a_1, r_1), (s_2, t_2, a_2, r_2), \ldots, (s_T, t_T, a_T, r_T)\}.$$

Note that only a terminal reward r_T is obtained (indicating whether the task successfully finishes), and step-wise rewards $r_1, \ldots, r_{T-1}$ are unavailable. To avoid additional assumptions on the unobserved rewards, we set $r_i = 0$ $(i < T)$ for all intermediate steps. Under this setting, the return at step i reduces to $G_i = \sum_{j=i}^{T} \gamma^{j-i} r_j = r_T$ with the choice of $\gamma = 1$. Consequently, $\mathbb{E}[G_i] = P(r_T = 1 \mid s_i, t_i, a_i, \pi_\theta)$ measures the probability for the task success given the first i steps.

Proximal Policy Optimization (PPO). Reinforcement Learning (RL) serves as a fundamental paradigm for improving agent–environment interactions (Ouyang et al., 2022). A widely-used algorithm in RL for large language models is PPO (Schulman et al., 2017), which employs an actor-critic framework to update the policy iteratively. Formally, PPO aims to maximize the following objective:

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{\mathbf{x}\sim q(\cdot), \mathbf{y}\sim\pi_{\theta_{\text{old}}}(\cdot|\mathbf{x})} \left[\frac{1}{|\mathbf{y}|} \sum\nolimits_{j=1}^{|\mathbf{y}|} \min(w_j(\theta)A_j, \text{clip}(w_j(\theta), 1-\epsilon, 1+\epsilon)A_j) \right],$$

where π_θ, $\pi_{\theta_{\text{old}}}$ denote the current and old policies, and $\mathbf{x} \sim q(\cdot)$, $\mathbf{y} \sim \pi_{\theta_{\text{old}}}$ denote the prompt and the corresponding response. The importance sampling ratio $w_j(\theta) = \frac{\pi_\theta(y_j|\mathbf{x}, \mathbf{y}_{<j})}{\pi_{\theta_{\text{old}}}(y_j|\mathbf{x}, \mathbf{y}_{<j})}$ is introduced for unbiased estimation, and the clipping hyper-parameter ϵ is employed for stable optimization. The advantage A_j measures the relative improvement of the expected return for y_j over a baseline that is estimated by a dedicated critic policy. Typically, the critic policy is of comparable scale to π_θ, which brings substantial memory and computational overhead, particularly for LLM scenarios.

Critic-Free Methods. Recently, critic-free methods have emerged as efficient alternatives to PPO by eliminating the need for a separate critic policy. One well-known method is Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which computes the baseline in A_j using the average reward across a group of responses generated for the same prompt. Mathematically, GRPO aims to optimize

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{K} \sum\nolimits_{i=1}^{K} \frac{1}{|\mathbf{y}_i|} \sum\nolimits_{j=1}^{|\mathbf{y}_i|} \min(w_{i,j}(\theta)A_{i,j}, \text{clip}(w_{i,j}(\theta), 1-\epsilon, 1+\epsilon)A_{i,j}) \right]$$

where the expectation operation is taken over the distributions $\mathbf{x} \sim q(\cdot)$, $\{\mathbf{y}_i\}_{i=1}^{K} \sim \pi_{\theta_{\text{old}}}(\cdot|\mathbf{x})$, and K is the group size, and the advantage $A_{i,j}$ is computed by:

$$A_{i,j} = A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_K)}{\text{std}(r_1, \ldots, r_K)} \tag{1}$$

with the reward r_i for the i-th response in the group. Following the similar idea, REINFORCE Leave-One-Out (RLOO) (Ahmadian et al., 2024) computes the advantage by

$$A_{i,j} = A_i = r_i - \frac{1}{K-1} \sum\nolimits_{j\neq i} r_j = \frac{K}{K-1} \left(r_i - \text{mean}(r_1, \ldots, r_K)\right) \tag{2}$$

to ensure unbiased estimation. REINFORCE++ (Hu et al., 2025) proposes to employ global batch statistics for normalization:

$$A_{i,j} = \frac{A'_{i,j} - \text{mean}_{\text{batch}}(A')}{\text{std}_{\text{batch}}(A')}, \text{ with } A'_{i,j} = r_i - \text{mean}(r_1, \ldots, r_K). \tag{3}$$

Notably, although these methods have proven effective in the RLVR framework, it is unsuitable to combine them with our agentic-Q directly. The reasons lie in that: (i) in the RLVR framework, rewards are typically binary, and these methods can naturally suppress extremely easy and hard tasks by

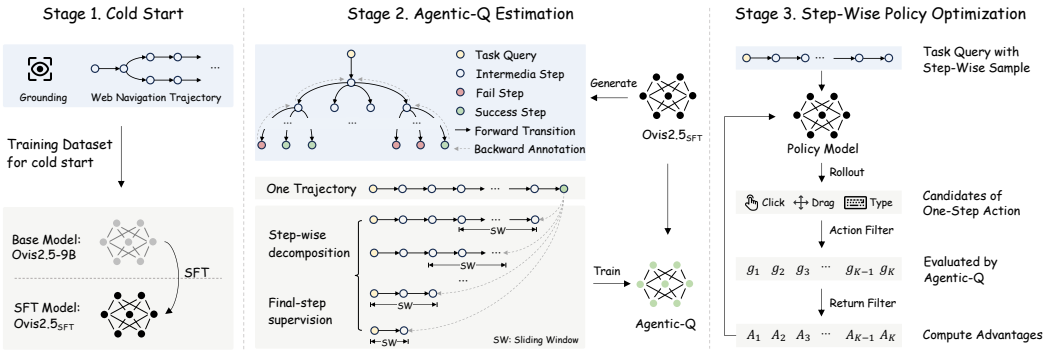


Figure 1: Illustration of our framework with three stages: (i) Ovis2.5$_{\text{SFT}}$ is trained on grounding data and a set of expert web navigation trajectories; (ii) we collect state-action trajectories by Ovis2.5$_{\text{SFT}}$ itself, and train our agentic-Q model in a binary classification manner; (iii) we optimize the policy with self-generated step-wise trajectories under the guidance of our agentic-Q model.

vanished advantages. In our framework, we employ continuous rewards from our agentic-Q for policy optimization. In extreme cases, small numerical differences in rewards can lead to low std, which in turn induces disproportionately large normalized advantages and results in unstable optimization; (ii) these methods optimize the objective by uniformly averaging over different groups, overlooking intrinsic difficulties of each group. Compared to easy tasks, hard ones are often more informative, but uniform weighting may ignore their contributions, leading to suboptimal optimization.

3 METHOD

This section presents our framework for training a GUI agent, with the overall pipeline illustrated in Figure 1. In the subsequent parts, we begin by describing the details of data curation and cold-start training, followed by agentic-Q estimation and step-wise policy optimization.

3.1 DATA CURATION

Detection-Rerunning Strategy. The GUI agent environment is typically non-stationary, posing significant challenges for data collection and performance evaluation. For example, due to load limitations and system configurations, it is common to encounter environmental issues including human verification, inaccessible websites and browser crashes when constructing web-browsing trajectories. In these cases, task-irrelevant noise is inadvertently introduced when collecting training data, which hinders the model from capturing the underlying task-specific patterns. Moreover, failed tasks in these cases cannot be directly attributed to insufficient performance during evaluation. To address these issues, we use a simple yet effective strategy, namely *detection-rerunning*. Specifically, we employ GPT-4o to assess whether a trajectory has been affected by environmental issues. If such disturbances are detected, we rerun the corresponding task and reconstruct the trajectory accordingly. In practice, we find that the simple strategy can mitigate almost all environmental issues.

Data Stratification. To ensure a stable and effective optimization on agent models, we conduct data stratification over the trajectories in training set. Specifically, for a single task, we repeat the web-browsing $n = 8$ times, and rank the task into Level $l \in \{0, 1, \ldots, 8\}$ based on the number of successful trajectories, where Level l indicates that l out of $n = 8$ rollouts successfully handle the task. It is important to note that tasks at Level 0 (highly challenging) and Level 8 (trivially easy) yield the same results (either all failures or all successes) across repeated rollouts, which contribute minimally to enhancing exploration. Therefore, we exclude the two types of tasks from training set.

Progress Supervision Annotation. In state-action trajectories, only the final-step supervision on the last action is available, resulting in a lack of process-level guidance for intermediate actions and posing challenges for training the agentic-Q model. A natural attempt is to annotate intermediate supervision using advanced models. However, this incurs substantial annotation costs and may fail to reflect the underlying contribution of each step to the final result. Therefore, we propose to propagate

the final outcome backward to each intermediate step, treating it as a form of process-level label for training agentic-Q model. Formally, for a trajectory $\zeta = \{(s_i, t_i, a_i, r_i)\}_{i=1}^{T}$, the return $G_i = r_T$ corresponding to the first i steps serves as an unbiased estimator of the probability for task completion. It is therefore natural to approximate G_i in an empirical-risk-minimization manner.

3.2 Cold Start

To initialize the model π_θ with fundamental GUI navigation capabilities, we fine-tune π_θ on grounding data and web navigation trajectories from experts, in which the trajectory data only records the browsing behavior without supervision on task completion. We adopt supervised fine-tuning (SFT) for cold start. For example, given web navigation trajectory data $\zeta = \{(s_i, t_i, a_i)\}_{i=1}^{T}$, the goal of SFT is to minimize:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{\{(s_i,a_i,t_i)\}\sim\mathcal{D}_{\text{SFT}}}\left[\sum\nolimits_{i=1}^{T} \log \pi_\theta(t_i, a_i|s_i)\right],$$

where $\mathcal{D}_{\text{SFT}}$ is the trajectory dataset for SFT. After the cold start, our MLLM-based agent acquires basic GUI navigation capabilities, such as understanding real-world webpages and generating valid actions (e.g., *left-click* and *scroll*). Starting from this initial model $\pi_{\theta_{\text{SFT}}}$, we proceed to train an agentic-Q model, and further enhance GUI navigation abilities via reinforcement learning.

3.3 Agentic-Q Estimation

The agentic-Q estimation begins by collecting a set of self-generated trajectories under $\pi_{\theta_{\text{SFT}}}$, with the process supervision propagated from the final outcome. Then, instead of treating the whole trajectory as a single sample, we decompose it into step-wise segments. Specifically, we employ each step (s_i, t_i, a_i, r_i) in the trajectory $\zeta = \{(s_i, t_i, a_i, r_i)\}_{i=1}^{T}$ as an independent training instance. The goal of our agentic-Q model Q_θ is to predict the return G_i corresponding to the step (s_i, t_i, a_i). Since $G_i = r_T \in \{0, 1\}$ is binary, a natural choice for agentic-Q estimation is to minimize the cross-entropy loss:

$$\mathcal{L}_{\text{CE}} = -\mathbb{E}_{(s_i,t_i,a_i,G_i)\sim\mathcal{D}_{\text{step}}}\left[G_i \log Q_\theta(s_i, t_i, a_i) + (1 - G_i)\log(1 - Q_\theta(s_i, t_i, a_i))\right], \quad (4)$$

where the dataset $\mathcal{D}_{\text{step}}$ contains step-wise samples. According to the favorable property of cross-entropy loss, the agentic-Q with optimal parameters can provably capture the distribution of return.

To ensure the agentic-Q generalizes well and to support stable policy optimization, we incorporate two techniques: *sliding-window* and *action-focus*. Specifically, at each step i, the state s_i encodes all previous interactions. When the interaction history becomes too long, the agentic-Q struggles to focus on the current context, leading to instability in subsequent policy optimization. For this reason, we adopt a sliding-window strategy that restricts the historical interactions in s_i to only the most recent ones. Moreover, since the step-wise sample includes both the thought and the action at each step, we apply an action-focus strategy that masks out the thought part during training. This encourages the agentic-Q to focus on the action, reducing potential disturbance from thought part.

3.4 Step-Wise Policy Optimization (SWPO)

Based on the initial model $\pi_{\theta_{\text{SFT}}}$ and self-generated step-wise trajectories $\mathcal{D}_{\text{step}}$, we proceed to further enhance the GUI navigation capability of our model. To this end, we propose a critic-free step-wise policy optimization method, combined with the agentic-Q model Q_θ. Compared to existing policy optimization methods, such as GRPO (Shao et al., 2024), we incorporate the following modifications:

- Instead of using final-outcomes as supervisions, we employ the predictions of Q_θ for policy optimization. In this way, we can conveniently use step-wise samples, and decouple the GUI execution environment from the policy optimization, thereby improving the efficiency;
- We introduce an action-level return-level filtering strategy. To be specific, given a state s, the agent model π_θ rolls out K candidate actions, and the agentic-Q model predicts the return for each action. If all proposed actions are identical (i.e., lack diversity), we discard the sample s; similarly, if the K returns are highly concentrated (i.e., with low std), we also filter out s;

- We explicitly consider the task difficulty by introducing adaptive group weighting. Inspired by self-normalized inverse propensity scoring (Swaminathan & Joachims, 2015), we use the inverse of average return for a group as the weight, to draw more attention to harder tasks.

The above designs can naturally integrate with existing critic-free policy optimization methods. Taking GRPO as an example, the objective of the step-wise variant of GRPO can be formulated as:

$$\mathcal{J}_{\text{S-GRPO}}(\theta) = \mathbb{E}\left[\frac{u(s)}{K}\sum_{i=1}^{K}\frac{1}{|t_i|+|a_i|}\sum_{j=1}^{|t_i|+|a_i|}\min(w_{i,j}(\theta)A_{i,j}, \text{clip}(w_{i,j}(\theta), 1-\epsilon, 1+\epsilon)A_{i,j})\right]$$

where the state $s \sim q'(\cdot)$ is sampled from the filtered set $q'(\cdot)$ and the expectation is taken over $s \sim q'(\cdot)$, $\{(t_i, a_i)\}_{i=1}^{K} \sim \pi_{\theta_{\text{old}}}(\cdot|s)$. The advantage $A_{i,j}$ and the group weight $u(s)$ are computed by:

$$A_{i,j} = A_i = \frac{g_i - \text{mean}(g_1, \ldots, g_K)}{\text{std}(g_1, \ldots, g_K)} \text{ and } u(s) = \frac{\bar{g}(s)^{-1}}{\sum_{s'} \bar{g}(s')^{-1}},$$

where g_i is the return from Q_θ for the step (s, t_i, a_i) with the i-th reasoning t_i and action a_i, and $\bar{g}(s)$ is the average return for s. Analogously, we can also derive step-wise variants of RLOO (Ahmadian et al., 2024) and REINFORCE++ (Hu et al., 2025), denoted as S-RLOO and S-RF++, respectively.

4 Experiments

In this section, we present empirical studies to validate the effectiveness of our framework. We commence by describing the experimental setup, and then deliver main results with their analyses.

4.1 Experimental Setup

Our framework is built on Ovis2.5-9B (Lu et al., 2025) that comprises a 970M visual encoder (i.e., a native-resolution ViT and a carefully-designed VET) and a 8B LLM (i.e., Qwen3-8B).

Benchmark. We conduct evaluations on a diverse set of benchmarks that cover two categories: (i) For GUI grounding, ScreenSpot (Cheng et al., 2024) provides over 1200 instructions and 600 screenshots, covering both text-based elements and a variety of widgets and icons across mobile, desktop and webpage; (ii) For GUI navigation, WebVoyager (He et al., 2024) compiles tasks from real-world websites (e.g., Amazon and Apple), and employs GPT-4V (OpenAI, 2023) as an automated evaluator to assess the performance of GUI agents. Online-Mind2Web (Xue et al., 2025) offers more diverse tasks of varying difficulty levels (easy, medium, and hard), collected from 136 real-world websites. Both benchmarks are evaluated in real-time environments where GUI agents interact with live websites, measuring authentic navigation capabilities.

Contenders. We compare our models against a set of advanced models with varying sizes, such as GPT-4o (OpenAI, 2024), Claude 3.7/4 Sonnet Compute Use (Anthropic, 2025a;b), Qwen3-VL-8B/32B (Bai et al., 2025), UI-TARS-7B/72B (Qin et al., 2025) and UI-TARS 1.5 7B (Seed, 2025b).

4.2 Experimental Results

We present the results for GUI grounding in Table 1, and for GUI navigation in Tables 2 and 3.

GUI Grounding. Overall, our models demonstrate remarkable performances compared to their powerful contenders across the scenarios of mobile, desktop and web, shown in Table 1. Specifically, the cold-start model Ovis2.5$_{\text{SFT}}$ achieves an average score of 88.60 on ScreenSpot v1 and 92.22 on ScreenSpot v2, outperforming similar-scale baselines (e.g., Qwen3-VL-8B and UI-TARS 1.5-7B), and even surpassing stronger proprietary models (i.e., GPT-4o, Claude 3.7/4 Sonnet). Moreover, we also observe that although step-wise policy optimization is performed on interaction trajectory data, the grounding ability of models does not degrade. In contrast, it even exhibits a slight improvement, such as Ovis2.5$_{\text{S-RF++}}$ achieves 92.45 on ScreenSpot v2, compared to 92.22 by Ovis2.5$_{\text{SFT}}$.

GUI Navigation. Tables 2 and 3 summarize performance comparisons on WebVoyager and Online-Mind2Web, respectively. From the results, we observe that our framework significantly improves

Table 1: Performance (%) comparisons on ScreenSpot v1 and v2, where three models Ovis2.5$_{\text{S-GRPO}}$, Ovis2.5$_{\text{S-RLOO}}$ and Ovis2.5$_{\text{S-RF++}}$ are trained by step-wise variants of GRPO, RLOO and REINFORCE++, respectively. At the last column, we report the average accuracy (Avg) over all scenarios.

Model	ScreenSpot v1							ScreenSpot v2						
	Mobile		Desktop		Web		Avg	Mobile		Desktop		Web		Avg
	Text	I/W	Text	I/W	Text	I/W		Text	I/W	Text	I/W	Text	I/W	
GPT-4o	20.20	24.90	21.10	23.60	12.20	7.80	18.30	–	–	–	–	–	–	20.10
Claude 4 Sonnet CU	97.07	79.48	81.44	64.29	92.61	82.52	84.75	97.59	82.94	84.02	67.14	96.58	82.27	87.11
Claude 3.7 Sonnet CU	88.28	73.36	79.90	62.86	90.87	80.58	80.74	89.31	75.83	82.47	72.86	93.59	80.30	83.57
Qwen3-VL-8B	95.24	79.91	94.85	81.43	92.17	83.50	88.44	97.93	86.73	95.88	81.43	94.02	86.21	91.35
Qwen3-VL-32B	96.34	85.59	96.91	87.86	90.87	90.78	91.67	98.28	89.10	98.97	90.71	94.87	92.61	94.50
UI-TARS-7B	92.31	85.15	90.72	84.29	88.70	85.92	88.21	94.83	88.15	93.81	86.43	93.16	86.21	90.96
UI-TARS-72B	94.14	82.53	95.36	85.00	90.87	83.50	88.92	96.90	86.73	97.42	88.57	91.88	86.21	91.75
UI-TARS 1.5-7B	92.67	77.73	93.81	71.43	86.52	77.18	84.20	96.55	82.94	96.91	76.43	90.17	76.85	87.81
Ovis2.5$_{\text{SFT}}$	95.24	78.60	94.85	85.00	92.61	83.01	88.60	97.24	84.83	97.42	87.86	95.73	86.70	92.22
Ovis2.5$_{\text{S-GRPO}}$	94.51	79.04	94.85	85.00	92.61	83.01	88.52	97.24	85.31	97.94	87.86	96.15	86.21	92.37
Ovis2.5$_{\text{S-RLOO}}$	94.87	79.04	94.85	85.00	92.17	82.52	88.44	97.24	85.31	97.42	88.57	95.73	86.70	92.37
Ovis2.5$_{\text{S-RF++}}$	95.24	79.04	94.85	84.29	92.61	83.50	88.68	97.59	85.31	97.42	87.86	95.73	87.19	92.45

Table 2: Performance (%) comparisons on WebVoyager, where three models Ovis2.5$_{\text{S-GRPO}}$, Ovis2.5$_{\text{S-RLOO}}$ and Ovis2.5$_{\text{S-RF++}}$ are trained by the step-wise variants of GRPO, RLOO and REINFORCE++, respectively. At the last column, we report the average task success rate over all websites, as well as the improvements of our three models over their SFT version Ovis2.5$_{\text{SFT}}$.

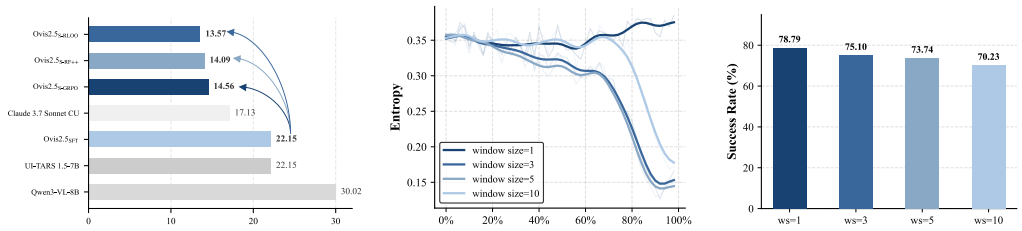
Model	Allrecipes	Amazon	Apple	ArXiv	BBC News	Cambridge Dictionary	Coursera	ESPN	GitHub	Google Map	Hugging-face	Wolfram Alpha	Avg
GPT-4o	55.56	53.66	55.81	60.47	54.76	81.40	64.29	43.18	58.54	56.10	41.86	65.22	57.59
Claude 4 sonnet CU	84.44	75.61	69.77	81.40	85.71	95.35	85.71	86.36	87.80	78.05	69.77	84.78	82.10
Claude 3.7 sonnet CU	75.56	75.61	62.79	86.05	69.05	88.37	78.57	75.00	68.29	85.37	72.09	82.61	76.65
Qwen3-VL-8B	35.56	46.34	34.88	34.88	40.48	67.44	61.90	27.27	48.78	60.98	32.56	50.00	44.94
Qwen3-VL-32B	37.78	26.83	46.51	48.84	54.76	79.07	64.29	54.55	65.85	56.10	18.60	34.78	48.83
UI-TARS-7B	17.78	31.71	16.28	20.93	40.48	58.14	71.43	45.45	31.71	17.07	41.86	39.13	35.99
UI-TARS-72B	15.56	39.02	46.51	65.12	40.48	72.09	59.52	47.73	43.90	26.83	25.58	47.83	44.16
UI-TARS 1.5-7B	26.67	29.27	48.84	53.49	61.90	60.47	66.67	38.64	41.46	58.54	46.51	50.00	48.44
Ovis2.5$_{\text{SFT}}$	66.67	70.73	58.14	79.07	78.57	88.37	69.05	56.82	70.73	60.98	44.19	80.43	68.68
Ovis2.5$_{\text{S-GRPO}}$	82.22	78.05	62.79	76.74	80.95	90.70	95.24	65.91	90.24	87.80	60.47	76.09	78.79$_{(+10.11)}$
Ovis2.5$_{\text{S-RLOO}}$	88.89	85.37	67.44	72.09	76.19	93.02	83.33	70.45	82.93	70.73	62.79	80.43	77.82$_{(+9.14)}$
Ovis2.5$_{\text{S-RF++}}$	77.78	70.73	69.77	83.72	80.95	81.40	92.86	72.73	75.61	80.49	72.09	80.43	78.21$_{(+9.53)}$

GUI navigation performance across both benchmarks, achieving gains of 9.14–10.11 points on WebVoyager and 10.00–11.33 points on Online-Mind2Web over Ovis2.5$_{\text{SFT}}$. Notably, our three models outperform almost all baselines over two benchmarks. Compared to UI-TARS series and Qwen3-VL series, our models demonstrate stronger performances, even surpassing larger-scale counterparts. Compared to proprietary models such as GPT-4o and Claude Sonnet series, our models also exhibit superior overall performance. Furthermore, our framework exhibits high computational efficiency, and significantly improves generalization performances. Specifically, (i) the incorporation of agentic-Q decouples the RL post-training from the external environment, rendering policy optimization efficient; (ii) all state-action trajectories for training are generated by the policy itself, eliminating the need for expensive annotations; and (iii) although the training data consists of websites from WebVoyager, our models achieve notable performance gains on the more diverse Online-Mind2Web. This demonstrates that our framework empowers agents with stronger generalization capability.

Navigation Efficiency. We evaluate the navigation efficiency of GUI agents by measuring the number of steps required for task completion, and present results on WebVoyager in Figure 2(a). The results show that our framework significantly improves navigation efficiency, e.g., reducing the number of steps from 22.15 to 13.57 for Ovis2.5$_{\text{S-RLOO}}$. We attribute this enhancement to the use of step-wise trajectories, which promotes deliberate action at each step and ensures efficient task completion.

Table 3: Performance (%) comparisons on Online-Mind2Web, where three models Ovis2.5$_{\text{S-GRPO}}$, Ovis2.5$_{\text{S-RLOO}}$ and Ovis2.5$_{\text{S-RF++}}$ are trained by the step-wise variants of GRPO, RLOO and REINFORCE++, respectively. At the last row, we report the average task success rate over all difficulty levels, as well as the improvements of our three models over their SFT version Ovis2.5$_{\text{SFT}}$.

	GPT-4o	Claude 4 Sonnet CU	Claude 3.7 Sonnet CU	Qwen3-VL -8B	Qwen3-VL -32B	UI-TARS 1.5 7B	**Ovis2.5$_{\text{SFT}}$**	**Ovis2.5$_{\text{S-GRPO}}$**	**Ovis2.5$_{\text{S-RLOO}}$**	**Ovis2.5$_{\text{S-RF++}}$**
Easy	–	71.60	72.84	34.94	38.55	57.83	66.27	77.11	75.90	78.31
Medium	–	65.47	66.91	25.87	27.97	27.27	51.75	62.94	60.84	65.73
Hard	–	53.52	46.48	9.46	14.86	17.57	39.19	51.35	51.35	44.59
Avg	37.00	62.33	61.00	24.33	27.67	33.33	52.67	64.00$_{(+11.33)}$	62.67$_{(+10.00)}$	64.00$_{(+11.33)}$



(a) Step comparison (b) Entropy comparison (c) Performance comparison

Figure 2: Comparisons under different setting: (a) compares average steps required to complete tasks in WebVoyager; (b) shows the policy entropy of Ovis2.5$_{\text{S-GRPO}}$ under varying sliding window sizes; (c) compares the overall performance of Ovis2.5$_{\text{S-GRPO}}$ across different window sizes.

Sliding Window. In our experiments, we find a strong positive correlation between GUI agent performance and policy entropy: sharp drops in entropy often lead to significant performance degradation. Maintaining stable entropy is therefore critical for effective policy optimization. To address this, we introduce a *sliding window* technique when training and exploiting our agentic-Q, in which returns are computed using only the most recent steps. This approach limits the historical context, mitigating premature policy convergence, preventing early entropy collapse, and reducing the risk of becoming trapped in suboptimal solutions. Additionally, it encourages the agent to adapt more dynamically to changing environments. We empirically validate this approach via ablation studies with window sizes of $\{1, 3, 5, 10\}$, training corresponding agentic-Q and policy models. As shown in Figure 2(b) and (c), larger window sizes lead to less stable entropy and a faster decline in policy entropy, which correlates with noticeable performance drops. Based on these results, we adopt a *window size* of 1 in practice, achieving more stable entropy and improved performance.

Boundary Traversal. During the experiments, we observe an interesting behavior, termed as *Boundary Traversal*: when the task is essentially infeasible within the current website, our models are able to recognize the limitation and proactively switch to alternative solutions to complete the task successfully. We present a real-world example in Figure 3. Specifically, the agent is instructed to find the price of an iPhone 14 Plus on `www.apple.com`, but the product has been delisted due to website updates. According to the trajectory, the agent first attempts to locate the item by thoroughly searching within the Apple website. After 34 unsuccessful steps, it realizes the task cannot be completed locally. At the step 35, the agent pivots to using a search engine, and by the step 39, successfully finds the desired information on Google. This behavior demonstrates a strong task-level generalization ability. Rather than being constrained by the boundaries of a single website, the agent can reason about task feasibility, identify failure conditions, and autonomously switch to alternative strategies. Furthermore, it exhibits adaptive planning and goal-directed reasoning beyond local environments.

Ablation Studies on SWPO. We conduct ablation studies on SWPO to investigate the impact of its each component (i.e., *action-level return-level filtering* and *adaptive group weighting*). Specifically, starting from the baseline GRPO$_{\text{agentic-Q}}$ that is trained by GRPO with our agentic-Q, we construct two variants: one by introducing *action-level return-level filtering*, and the other by further applying *adaptive group weighting*. We report the results on WebVoyager in Table 4. The average performance

Table 4: Ablation studies of SWPO on WebVoyager, where $\text{GRPO}_{\text{agentic-Q}}$ is the baseline trained by GRPO with our agentic-Q, and two variants are trained by sequentially adding *action-level return-level filtering* and *adaptive group weighting* to $\text{GRPO}_{\text{agentic-Q}}$. At the last column, we report the average task success rate over all websites, as well as the improvements over the baseline $\text{GRPO}_{\text{agentic-Q}}$.

Model	Allrecipes	Amazon	Apple	ArXiv	BBC News	Cambridge Dictionary	Coursera	ESPN	GitHub	Google Map	Hugging-face	Wolfram Alpha	Avg
$\text{GRPO}_{\text{agentic-Q}}$	84.44	65.85	55.81	79.07	76.19	81.40	83.33	59.09	73.17	68.29	69.77	82.61	73.35
+ action-return filtering	80.00	75.61	62.79	76.74	78.57	90.70	88.10	61.36	80.49	75.61	72.09	76.09	76.46 (+3.11)
+ adaptive group weighting	82.22	78.05	62.79	76.74	80.95	90.70	95.24	65.91	90.24	87.80	60.47	76.09	78.79 (+5.44)

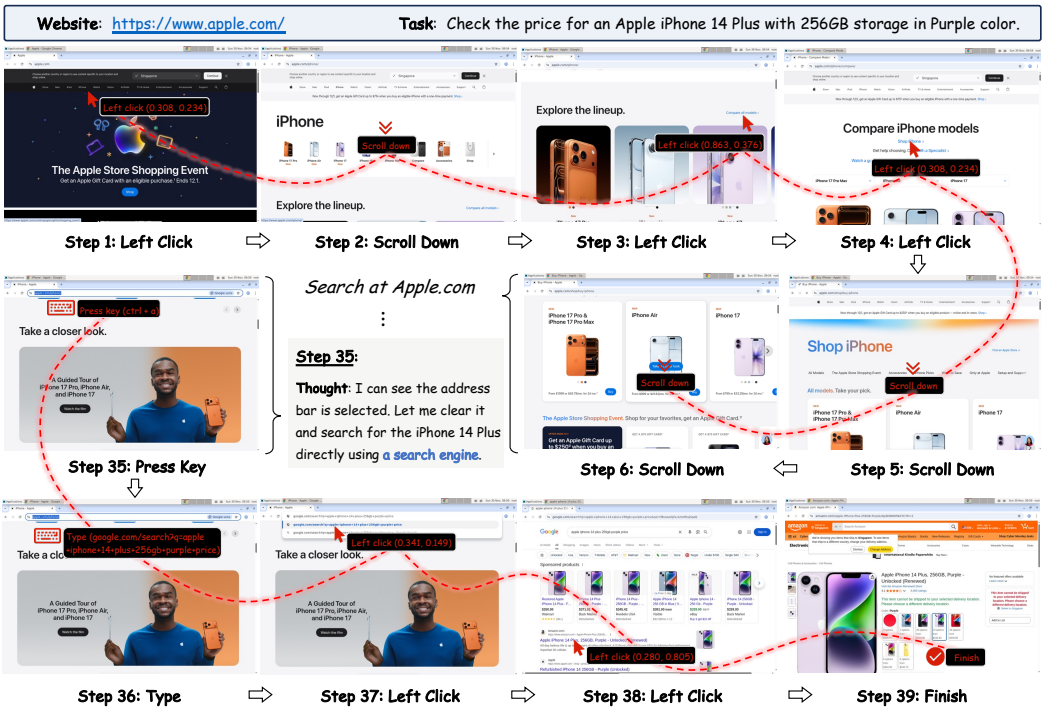


Figure 3: Example of boundary traversal. The agent is prompted to retrieve the price of iPhone 14 Plus at *www.apple.com*. However, due to recent updates, this product has been removed. For the first 34 steps, the agent persistently searches within the site but fails to locate target information. At step 35, it switches strategies and conducts an external search on *Google*, finishing the task by step 39.

across different websites shows a 3.11-point improvement when action-level return filtering is applied to the baseline, and 5.44-point improvement when adaptive group weighting is further introduced. Notably, the gains are more significant on Coursera, GitHub, and Google Maps, highlighting the effectiveness of our two techniques. We also report more ablation results in Section B.

5 Conclusion

In this paper, we investigate native screen-based GUI agents, and propose a novel framework with two key designs: agentic-Q estimation and step-wise policy optimization. First, we train an agentic-Q that can evaluate the contribution of a given action to task completion. Then, we optimize the agent with our agentic-Q over step-wise trajectories. All our training data are generated by the policy itself, so that the data collection costs are manageable. Moreover, the incorporation of agentic-Q decouples the policy update from environments, ensuring stable and efficient optimization. Empirically, we conduct comprehensive experiments to evaluate the performances on GUI grounding and GUI navigation. Experimental results demonstrate that our 9B agent achieves superior performance compared to others of the same scale, and even outperforms more powerful models such as the Claude series.

AI USE STATEMENT

In this work, we used generative AI tools for writing polishing. We take responsibility for the final content of this work, including text and claims produced with the aid of generative AI.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE style optimization for learning from human feedback in LLMs. *ArXiv e-prints*, arXiv:2402.14740, 2024.

Anthropic. Developing a computer use model. https://www.anthropic.com/news/developing-computer-use, 2024.

Anthropic. Claude 3.7 sonnet and Claude code. https://www.anthropic.com/news/claude-3-7-sonnet, 2025a.

Anthropic. Introducing Claude 4. https://www.anthropic.com/news/claude-4, 2025b.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report. *ArXiv e-prints*, arXiv:2511.21631, 2025.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing gui grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 9313–9332, 2024.

Google. A new era of intelligence with Gemini 3. https://blog.google/products/gemini/gemini-3/#learn-anything, 2025.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. In *Proceedings of the 11th International Conference on Learning Representations*, 2023.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 6864–6890, 2024.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization. *ArXiv e-prints*, arXiv:2501.03262, 2025.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024.

Shiyin Lu, Yang Li, Yu Xia, Yuwei Hu, Shanshan Zhao, Yanqing Ma, Zhichao Wei, Yinglun Li, Lunhao Duan, Jianshan Zhao, Yuxuan Han, Haijun Li, Wanying Chen, Junke Tang, Chengkun Hou, Zhixing Du, Tianli Zhou, Wenjie Zhang, Huping Ding, Jiahe Li, Wen Li, Gui Hu, Yiliang Gu, Siran Yang, Jiamang Wang, Hailong Sun, Yibo Wang, Hui Sun, Jinlong Huang, Yuping He, Shengze Shi, Weihong Zhang, Guodong Zheng, Junpeng Jiang, Sensen Gao, Yi-Feng Wu, Sijia Chen, Yuhui Chen, Qing-Guo Chen, Zhao Xu, Weihua Luo, and Kaifu Zhang. Ovis2.5 technical report. *ArXiv e-prints*, arXiv:2508.11737, 2025.

Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. GUI-R1: A generalist R1-style vision-language action model for GUI agents. *ArXiv e-prints*, arXiv:2504.10458, 2025.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Mehrab Tanjim, Nesreen K. Ahmed, Puneet Mathur, Seunghyun Yoon, Lina Yao, Branislav Kveton, Jihyung Kil, Thien Huu Nguyen, Trung Bui, Tianyi Zhou, Ryan A. Rossi, and Franck Dernoncourt. GUI agents: A survey. *ArXiv e-prints*, arXiv:2412.13501, 2024.

OpenAI. GPT-4 technical report. *ArXiv e-prints*, arXiv:2303.08774, 2023.

OpenAI. Hello GPT-4o. `https://openai.com/index/hello-gpt-4o/`, 2024.

OpenAI. Computer-using agent. `https://openai.com/index/computer-using-agent/`, 2025.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35*, pp. 27730–27744, 2022.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Awadallah. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. *ArXiv e-prints*, arXiv:2502.11357, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: Pioneering automated GUI interaction with native agents. *ArXiv e-prints*, arXiv:2501.12326, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv e-prints*, arXiv:1707.06347, 2017.

Seed. Seed1.5-VL vision-language multimodal large model. `https://seed.bytedance.com/en/seed1_5_vl`, 2025a.

Seed. Introducing UI-TARS-1.5: An open-source multimodal agent built upon a powerful vision-language model. `https://seed.bytedance.com/en/ui-tars`, 2025b.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *ArXiv e-prints*, arXiv:2402.03300, 2024.

Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems 28*, 2015.

Fei Tang, Zhangxuan Gu, Zhengxi Lu, Xuyang Liu, Shuheng Shen, Changhua Meng, Wen Wang, Wenqi Zhang, Yongliang Shen, Weiming Lu, Jun Xiao, and Yueting Zhuang. GUI-G^2: Gaussian reward modeling for GUI grounding. *ArXiv e-prints*, arXiv:2507.15846, 2025a.

Fei Tang, Haolei Xu, Hang Zhang, Siqi Chen, Xingyu Wu, Yongliang Shen, Wenqi Zhang, Guiyang Hou, Zeqi Tan, Yuchen Yan, Kaitao Song, Jian Shao, Weiming Lu, Jun Xiao, and Yueting Zhuang. A survey on (M)LLM-Based GUI agents. *ArXiv e-prints*, arXiv:2504.13865, 2025b.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, Chong Liu, Daoguang Zan, Fuxing Leng, Hanbin Wang, Hao Yu, Haobin Chen, Hongyi Guo, Jing Su, Jingjia Huang, Kai Shen, Kaiyu Shi, Lin Yan, Peiyao Zhao, Pengfei Liu, Qinghao Ye, Renjie Zheng, Shulin Xin, Wayne Xin Zhao,

Wen Heng, Wenhao Huang, Wenqian Wang, Xiaobo Qin, Yi Lin, Youbin Wu, Zehui Chen, Zihao Wang, Baoquan Zhong, Xinchun Zhang, Xujing Li, Yuanfan Li, Zhongkai Zhao, Chengquan Jiang, Faming Wu, Haotian Zhou, Jinlin Pang, Li Han, Qi Liu, Qianli Ma, Siyao Liu, Songhua Cai, Wenqi Fu, Xin Liu, Yaohui Wang, Zhi Zhang, Bo Zhou, Guoliang Li, Jiajun Shi, Jiale Yang, Jie Tang, Li Li, Qihua Han, Taoran Lu, Woyu Lin, Xiaokang Tong, Xinyao Li, Yichi Zhang, Yu Miao, Zhengxuan Jiang, Zili Li, Ziyuan Zhao, Chenxin Li, Dehua Ma, Feng Lin, Ge Zhang, Haihua Yang, Hangyu Guo, Hongda Zhu, Jiaheng Liu, Junda Du, Kai Cai, Kuanye Li, Lichen Yuan, Meilan Han, Minchao Wang, Shuyue Guo, Tianhao Cheng, Xiaobo Ma, Xiaojun Xiao, Xiaolong Huang, Xinjie Chen, Yidi Du, Yilin Chen, Yiwen Wang, Zhaojian Li, Zhenzhu Yang, Zhiyuan Zeng, Chaolin Jin, Chen Li, Hao Chen, Haoli Chen, Jian Chen, Qinghao Zhao, and Guang Shi. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *ArXiv e-prints*, arXiv:2509.02544, 2025a.

Ke Wang, Tianyu Xia, Zhangxuan Gu, Yi Zhao, Shuheng Shen, Changhua Meng, Weiqiang Wang, and Ke Xu. E-ANT: A large-scale dataset for efficient automatic gui navigation. *ArXiv e-prints*, arXiv:2406.14250, 2024.

Yibo Wang, Guangda Huzhang, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. SPACE: Noise contrastive estimation stabilizes self-play fine-tuning for large language models. In *Advances in Neural Information Processing Systems 38*, 2025b.

Yibo Wang, Hai-Long Sun, Guangda Huzhang, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. Triplets better than pairs: Towards stable and effective self-play fine-tuning for LLMs. In *Advances in Neural Information Processing Systems 38*, 2025c.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-ATLAS: A foundation action model for generalist gui agents. *ArXiv e-prints*, arXiv:2410.23218, 2024a.

Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, Zhenda Xie, Yu Wu, Kai Hu, Jiawei Wang, Yaofeng Sun, Yukun Li, Yishi Piao, Kang Guan, Aixin Liu, Xin Xie, Yuxiang You, Kai Dong, Xingkai Yu, Haowei Zhang, Liang Zhao, Yisong Wang, and Chong Ruan. DeepSeek-VL2: Mixture-of-experts vision-language models for advanced multimodal understanding. *ArXiv e-prints*, arXiv:2412.10302, 2024b.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *ArXiv e-prints*, arXiv:2504.01382, 2025.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained GUI agents: A survey. *ArXiv e-prints*, arXiv:2411.18279, 2024.

Danyang Zhang, Situo Zhang, Ziyue Yang, Zichen Zhu, Zihan Zhao, Ruisheng Cao, Lu Chen, and Kai Yu. ProgRM: Build better GUI agents with progress rewards. *ArXiv e-prints*, arXiv:2505.18121, 2025.

A TRAJECTORIES ON WEBSITES

In this part, we show trajectories of agents on websites, e.g., Huggingface, Google Map, and Arxiv.

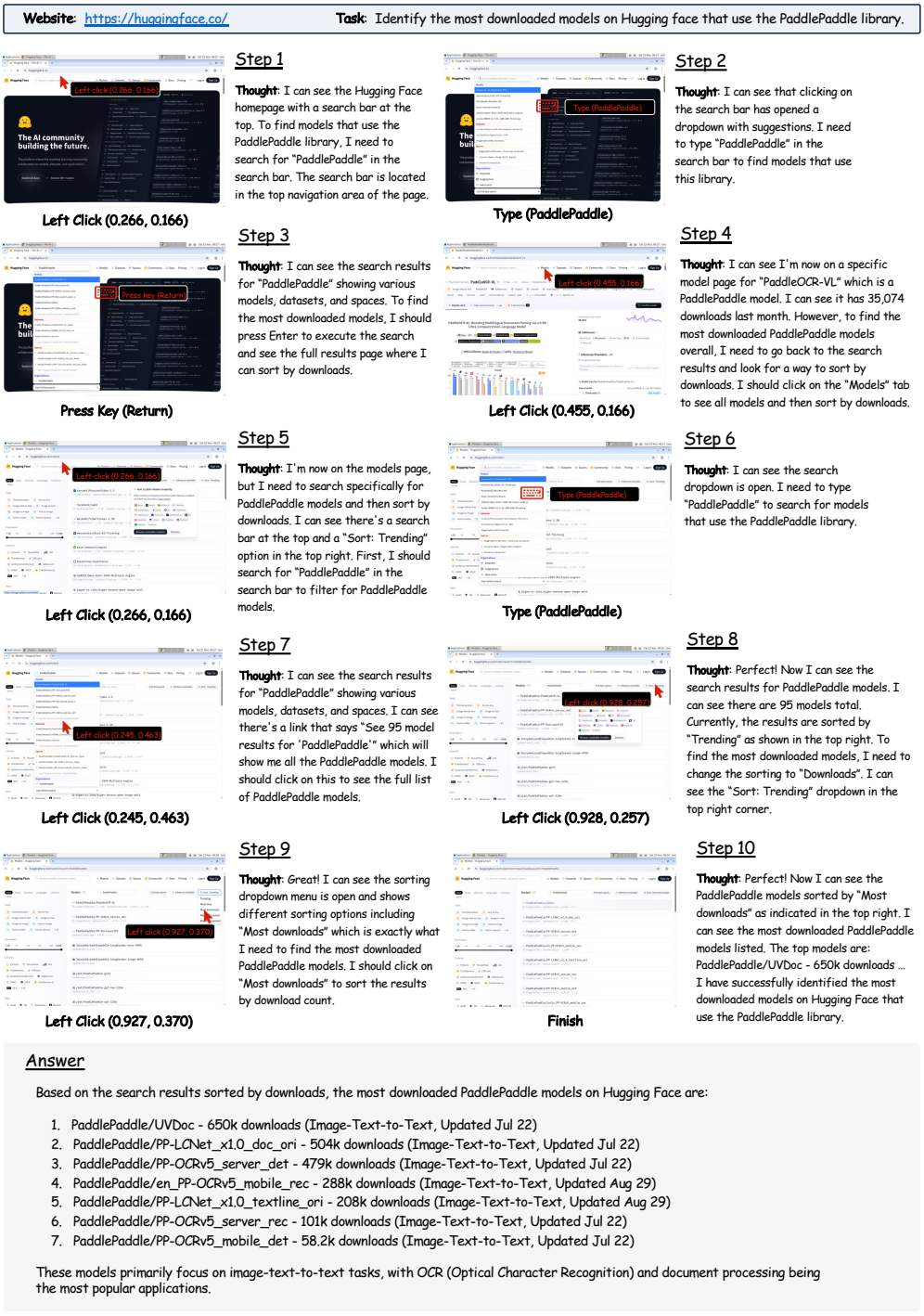
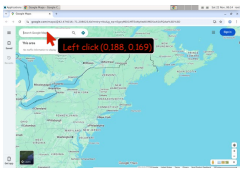


Figure 4: Trajectory for the task on Huggingface.

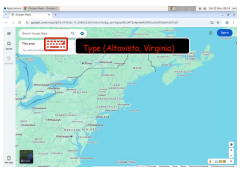
Website: https://www.google.com/maps/ **Task**: Tell me one bus stop that is nearest to the intersection of main street and Amherst street in Altavista.



Left Click (0.188, 0.169)

Step 1

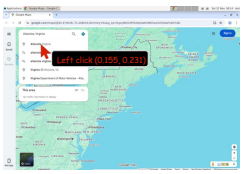
Thought: I can see a Google Maps interface showing a map of the eastern United States. However, I need to search for Altavista, Virginia to find the intersection of Main Street and Amherst Street and locate nearby bus stops. The current map is showing a broader view of the US, so I need to search for Altavista specifically.



Type (Altavista, Virginia)

Step 2

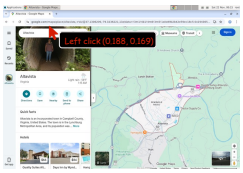
Thought: I can see the Google Maps interface is still showing the same view. I need to click on the search box and type "Altavista, Virginia" to navigate to the specific location where I need to find the intersection of Main Street and Amherst Street and nearby bus stops.



Left Click (0.155, 0.231)

Step 3

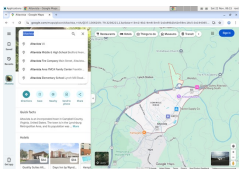
Thought: I can see that I've typed "Altavista, Virginia" in the search box and Google Maps is showing search suggestions. I can see "Altavista Virginia" as the first option in the dropdown. I should click on this option to navigate to Altavista, Virginia.



Left Click (0.188, 0.169)

Step 4

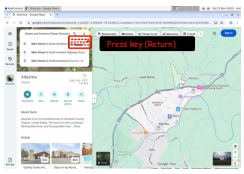
Thought: Great! Now I can see Altavista, Virginia on the map. I can see the town layout with various streets and locations. To find the intersection of Main Street and Amherst Street and nearby bus stops, I need to search more specifically for these streets. I should modify my search to look for "Main Street and Amherst Street Altavista" to get more precise results.



Type (Main Street and Amherst Street Altavista)

Step 5

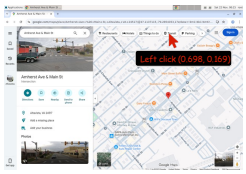
Thought: I can see that I've selected the text in the search box and there are some search suggestions appearing. I notice one suggestion mentions "Altavista Fire Company Main Street" which indicates Main Street exists in Altavista. I should type a more specific search to find the intersection of Main Street and Amherst Street.



Press Key (Return)

Step 6

Thought: I can see the search suggestions are showing various Main Street intersections, but I don't see a specific "Main Street and Amherst Street" intersection in the suggestions. I should press Enter to search for this specific intersection, or I could try clicking on one of the Main Street options to see if it leads me to the area where I can find Amherst Street.



Left Click (0.698, 0.169)

Step 7

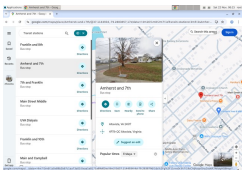
Thought: Perfect! I can see that Google Maps has found the intersection of Amherst Avenue and Main Street in Altavista, VA. I can see the map showing this intersection with a red marker. Now I need to look for nearby bus stops. I should click on the "Transit" button in the top menu to see public transportation options and bus stops in this area.



Left Click (0.298, 0.373)

Step 8

Thought: Excellent! Now I can see the transit stations (bus stops) near the intersection of Amherst Avenue and Main Street in Altavista. Looking at the map and the list on the left, I can see several bus stops. The closest one appears to be "Amherst and 7th" which is very near the intersection. I can also see "Main Street Middle" which is also close. Let me click on "Amherst and 7th" to get more details about this bus stop.



Finish

Step 9

Thought: Perfect! I now have the information I need. I can see the details for the "Amherst and 7th" bus stop, which is located in Altavista, VA 24517. Looking at the map, I can see that this bus stop is very close to the intersection of Amherst Avenue and Main Street. The bus stop is marked with a red pin on the map and appears to be one of the nearest transit options to the requested intersection.

Answer

The nearest bus stop to the intersection of Main Street and Amherst Street in Altavista is the "Amherst and 7th" bus stop, located at 4P78+QC Altavista, Virginia. This bus stop is very close to the intersection and serves as a convenient transit option for people traveling to or from this area.

Figure 5: Trajectory for the task on Google Maps.

Website: https://arxiv.org/ **Task**: Find the ArXiv Blog on the ArXiv website and summarize the content of its latest article.



Scroll Down

Step 1

Thought: I can see the ArXiv website homepage, but I need to find the ArXiv Blog. Looking at the current page, I can see the main navigation and content areas, but I don't see a direct link to the ArXiv Blog. I should look for navigation elements or links that might lead to the blog. Let me check if there's a footer or additional navigation that might contain a link to the blog.



Scroll Down

Step 2

Thought: I can see more content on the ArXiv website after scrolling, including sections for Mathematics, Computer Science, Quantitative Biology, and Quantitative Finance. However, I still don't see a direct link to the ArXiv Blog. I need to continue scrolling to see if there's a footer section that might contain links to the blog, or I should look for other navigation options that might lead to the blog.



Scroll Down

Step 3

Thought: I can see more sections including Statistics, Electrical Engineering and Systems Science, Economics, and an "About arXiv" section at the bottom. I still haven't found a direct link to the ArXiv Blog. Let me continue scrolling to see if there's a footer section that might contain links to the blog, or I should try a different approach like using the search functionality or looking for navigation menus.



Left Click (0.500, 0.101)

Step 4

Thought: I can see the footer section of the ArXiv website now, which contains links like "About", "Help", "Contact", "Subscribe", "Copyright", "Privacy Policy", "Web Accessibility Assistance", and "arXiv Operational Status". I still don't see a direct link to the ArXiv Blog. Let me try a different approach - I'll use the search functionality to look for the ArXiv Blog directly.



Type (arxiv blog)

Step 5

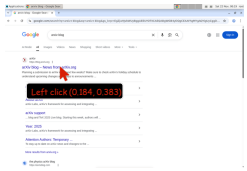
Thought: I can see the address bar is now selected and highlighted. I'll type a search query to look for the ArXiv Blog directly. I'll search for "arxiv blog" to find the blog section.



Left Click (0.200, 0.149)

Step 6

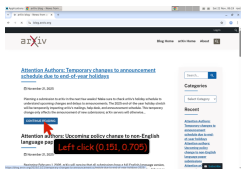
Thought: I can see that Google search suggestions have appeared for "arxiv blog". I can see the first suggestion is "arxiv blog - Google Search" which would be the most direct way to search for the ArXiv Blog. I'll click on this suggestion to perform the search.



Left Click (0.184, 0.383)

Step 7

Thought: Perfect! I can see the Google search results for "arxiv blog" and I found the official ArXiv blog at the top of the results. The first result shows "arXiv blog — News from arXiv.org" with the URL https://blog.arxiv.org/. This is exactly what I'm looking for. I'll click on this link to access the ArXiv Blog.



Left Click (0.151, 0.705)

Step 8

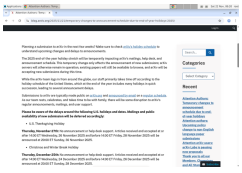
Thought: Excellent! I've successfully found the ArXiv Blog. I can see the blog homepage with the latest articles. The most recent article appears to be "Attention Authors: Temporary changes to announcement schedule due to end-of-year holidays" dated November 21, 2025... To get the full content of this latest article, I should click on "CONTINUE READING" to read the complete article.



Scroll Down

Step 9

Thought: Perfect! I can now see the full content of the latest ArXiv Blog article. The article is titled "Attention Authors: Temporary changes to announcement schedule due to end-of-year holidays" and was published on November 21, 2025 by Kat Boboris... I should scroll down to read the complete article content.



Finish

Step 10

Thought: Excellent! I now have the complete content of the latest ArXiv Blog article. I can see the full details about the temporary changes to the announcement schedule due to end-of-year holidays... I have enough information to provide a comprehensive summary of this latest article.

Answer

The latest ArXiv Blog article informs authors about temporary changes to arXiv's announcement schedule due to the 2025 end-of-year holiday stretch. Key points include:

1. Holiday Impact: The 2025 end-of-year holiday period will temporarily impact arXiv's mailings, help desk, and announcement schedule.
2. Scope of Impact: This temporary change only affects the announcement of new submissions. arXiv servers will remain operational, existing papers will still be available to browse, and arXiv will continue accepting new submissions during this time.
3. Holiday Schedule: arXiv staff primarily follows the United States holiday schedule, which includes multiple holidays in quick succession, leading to announcement delays.
4. Specific Holiday Delays: The article provides detailed information about delays around specific U.S. holidays:
    1. Thanksgiving Holiday: Thursday, November 27th - No announcement or help desk support. Articles received and accepted between November 26-28 will be announced on Sunday, November 30 at 20:00 ET.
    2. Christmas and Winter Break Holiday: Thursday, December 25th - No announcement or help desk support. Articles received and accepted between December 24-26 will be announced on Sunday, December 28 at 20:00 ET.
5. Recommendation: Authors are advised to check arXiv's holiday schedule to understand upcoming changes and delays to announcements.

The article emphasizes that while there will be some disruption to arXiv's regular announcements, mailings, and user support during the holiday period, the core submission and browsing functions will remain available.

Figure 6: Trajectory for the task on Arxiv.

Table 5: Performance (%) comparisons with on WebVoyager, where GRPO is the model trained by GRPO with the trajectory-level reward (i.e., only using the final outcome r_T as the supervision for optimization), and S-GRPO denotes the model trained by SWPO with our agentic-Q. At the last column, we report the average success rate over all websites, as well as the improvements over SFT.

Model	Allrecipes	Amazon	Apple	ArXiv	BBC News	Cambridge Dictionary	Coursera	ESPN	GitHub	Google Map	Hugging-face	Wolfram Alpha	Avg
SFT	66.67	70.73	58.14	79.07	78.57	88.37	69.05	56.82	70.73	60.98	44.19	80.43	68.68
GRPO	82.22	63.41	53.49	83.72	76.19	88.37	83.33	77.27	75.61	70.73	58.14	80.43	74.51$_{(+5.83)}$
S-GRPO	82.22	78.05	62.79	76.74	80.95	90.70	95.24	65.91	90.24	87.80	60.47	76.09	**78.79**$_{(+10.11)}$

Table 6: Performance (%) comparisons with on WebVoyager, where the first column denotes the std threshold for the standard deviation in the return-level filter strategy of SWPO. At the last column, we report the average success rate over all websites.

std	Allrecipes	Amazon	Apple	ArXiv	BBC News	Cambridge Dictionary	Coursera	ESPN	GitHub	Google Map	Hugging-face	Wolfram Alpha	Avg
0	80.00	70.73	55.81	79.07	73.81	90.70	80.95	68.18	73.17	73.17	62.79	71.74	73.35
0.01	86.67	75.61	67.44	81.40	83.33	90.70	85.71	70.45	78.05	80.49	62.79	78.26	78.40
0.03	82.22	78.05	62.79	76.74	80.95	90.70	95.24	65.91	90.24	87.80	60.47	76.09	**78.79**
0.05	80.00	73.17	67.44	83.72	78.57	93.02	95.24	70.45	80.49	80.49	65.12	73.91	78.40
0.1	82.22	70.73	58.14	83.72	83.33	93.02	83.33	63.64	65.85	75.61	65.12	78.26	75.29

B More Details on Experiments

In this section, we provide more information about the experiments, including implementation details, comparison with trajectory-level GRPO, and more ablation studies.

Implementation Details. We collect state-action trajectories on $2k$ tasks with $n = 8$ rollouts per task, yielding $16k$ trajectories in total. After data stratification and step decomposition, we obtain $112k$ step-wise training samples. Agentic-Q is trained with (4), using sliding window size of 1, action-focus masking, and AdamW (learning rate 10^{-6}, batch size 32, 1 epoch, with a warmup ratio of 0.1). During SWPO, we use group size $K = 24$ and std threshold 0.03. The optimization hyper-parameters are set by learning rate 10^{-6}, clip ratio $\epsilon = 0.24$, and batch size 256. All experiments are conducted on H100 GPUs with 80GB memory.

Comparison with Trajectory-Level GRPO. We conduct an ablation study to evaluate the impact of different training strategies on model performance, as summarized in Table 5. SFT represents the base model fine-tuned via supervised fine-tuning. GRPO builds upon SFT by applying GRPO (Shao et al., 2024) using the trajectory-level reward (i.e., only the final outcome r_T) as supervision. S-GRPO also starts from SFT, but leverages an intermediate-step reward model (i.e., agentic-Q) to score each action, enabling step-wise policy optimization. The results in Table 5 indicate that GRPO improves the average success rate from 68.68% (SFT) to 74.51%, demonstrating the benefit of trajectory-level reinforcement learning. More importantly, S-GRPO further enhances performance, achieving an average success rate of 78.79%, outperforming GRPO by 4.28 percentage points and SFT by 10.11 points. This confirms that incorporating intermediate-step reward signals for step-wise optimization can more effectively improve performances on complex sequential GUI agent tasks.

Ablation on STD Thresholds. We further examine the effect of different standard deviation (std) thresholds in return-level filtering strategy, as shown in Table 6. The results demonstrate that the choice of std threshold significantly influences performance. A threshold of 0.03 achieves the best average success rate of 78.79%, outperforming both smaller thresholds (e.g., 0.01 with 78.40%) and larger thresholds (e.g., 0.1 with 75.29%). This indicates that filtering out groups with low reward variance effectively removes uninformative rollouts, while retaining enough diversity to guide the agent toward optimal policies. Based on these findings, we adopt std = 0.03 as the final threshold.

Ablation on Data Stratification and Detection-Rerunning Strategies. We also evaluate the impact of our data curation strategies (i.e., data stratification and detection-rerunning) on performances, as

Table 7: Performance (%) comparisons with on WebVoyager, where *all level* includes data from Level 0 to Level 8 for training, and *EI* employs environmentally corrupted data for training, and *ours* uses data from Level 1 to Level 7 and excludes environmentally corrupted samples. At the last column, we report the average success rate over all websites.

Method	Allrecipes	Amazon	Apple	ArXiv	BBC News	Cambridge Dictionary	Coursera	ESPN	GitHub	Google Map	Hugging--face	Wolfram Alpha	Avg
all level	84.44	73.17	65.12	79.07	76.19	88.37	83.33	65.91	70.73	80.49	53.49	78.26	74.90
EI	77.78	78.05	58.14	74.42	66.67	81.40	85.71	70.45	63.41	78.05	51.16	76.09	71.79
ours	82.22	78.05	62.79	76.74	80.95	90.70	95.24	65.91	90.24	87.80	60.47	76.09	**78.79**

summarized in Table 7. We compared three settings: *all level*, which uses tasks from Level 0 to Level 8 for training; *EI*, which retains environmentally corrupted samples; and *ours*, which uses tasks from Level 1 to Level 7 and excludes environmentally corrupted data. Results show that *ours* achieves the highest average success rate of 78.79%, outperforming both *all level* (74.90%) and *EI* (71.79%). This demonstrates that filtering out both extreme difficulty tasks and environmentally corrupted samples effectively enables the GUI agent to better navigate websites.

C PROMPTS

In this section, we present the prompt (including descriptions of action space) used for our framework.

```
As a robot operating a browser inside a virtual machine, your task is
    to complete the query through a series of actions with the web
   page and strictly follow the rules.

  Core Requirements:
  • Complete ALL requirements specified in the query. If multiple
      requirements exist, ensuring all are satisfied.

  Rules:

  1. In each iteration, you will receive an Observation that includes
       a screenshot of the web page and some texts.

  2. First carefully analyze the visual information to determine the
      location corresponding to the web element that requires
      interaction. Describe the location using relative coordinates
      in the following format:
     <point>(x,y)</point>
     where x and y are floating-point numbers between 0.000 and 1.000
         representing the horizontal and vertical positions,
        respectively.
     Note: If no specific point is required, output "None".

  3. Then choose one Action Type from the following actions:
     (1) HOVER: Hover the mouse over the specified coordinates.
     (2) LEFT_CLICK: Click the left mouse button on the specified
        coordinates.
     (3) RIGHT_CLICK: Click the right mouse button on the specified
        coordinates.
     (4) MIDDLE_CLICK: Click the middle mouse button on the specified
         coordinates.
     (5) DOUBLE_CLICK: Double-click the left mouse button on the
        specified coordinates.
     (6) TRIPLE_CLICK: Triple-click the left mouse button on the
        specified coordinates.
     (7) TYPE: Type the specified text.
     (8) DRAG: Drag the mouse from a start point to an end point.
```

```
        (9) SCROLL: Scroll the page in the specified direction. If the
            coordinates are 'None', the default is to scroll the entire
            window. Otherwise, scroll from the specified coordinates.
        (10) WAIT: Wait for a specified duration or for unfinished page
            processes.
        (11) PRESS_KEY: Press a key or hotkey to complete a specific
            operation. The value supports xdotool's 'key' syntax.
              Examples: "a", "Return", "alt+Tab", "ctrl+s", "Up", "KP_0"
                    (for the numpad 0 key).
        (12) COPY_IMAGE: Get Image Url at the coordinates.
        (13) FINISHED: This action should be selected if the query has
            been resolved or is considered unresolvable.

    4. Finally, determine the action value based on the action type.
        Note that if no content is required, output 'None'.
        (1) HOVER: value is None.
        (2) LEFT_CLICK, (3) RIGHT_CLICK, (4) MIDDLE_CLICK, (5)
            DOUBLE_CLICK, (6) TRIPLE_CLICK, (12) COPY_IMAGE:
           For these click actions, the value is fixed to None. The
               specific coordinates to operate on are provided as the
               Action Element.
        (7) TYPE: value is the text content that needs to be entered.
        (8) DRAG: value is None. Action Element should be two <point>(x,
            y)</point> entries.
        (9) SCROLL: value is the scroll direction, such as left, right,
            up, or down.
        (10) WAIT: value is None.
        (11) PRESS_KEY: value is the key or key-combination to be
            pressed.
        (13) FINISHED: value is the final answer for the given query or
            the reason why it could not be solved.

    5. Output only one action per iteration and strictly in the
        following format:
       • Thought: {brief thoughts (brief summary of information that
            helps answer the query)}
       • Action Element: {the coordinates in <point>(x,y)</point>
            format; for DRAG operations, provide two coordinates;
            otherwise output 'None'}
       • Action Type: {one Action Type you choose to operate}
       • Action Value: {one Action Value you choose in the above format
            }
```