Adapting Web Agents with Synthetic Supervision

Anonymous ACL submission

Abstract

Web agents struggle to adapt to new websites due to the scarcity of environment specific tasks and demonstrations. Recent works have explored synthetic data generation to address this challenge, however, they suffer from data quality issues where synthesized tasks contain hallucinations that cannot be executed, and collected trajectories are noisy with redundant or misaligned actions. In this paper, we propose `SynthAgent`, a fully synthetic supervision framework that aims at improving synthetic data quality via dual refinement of both tasks and trajectories. Our approach begins by synthesizing diverse tasks through categorized exploration of web elements, ensuring efficient coverage of the target environment. During trajectory collection, tasks are refined only when conflicts with observations are detected, which mitigates hallucinations while preserving task consistency. After collection, we conduct trajectory refinement with global context to mitigate potential noise or misalignments. Finally, we fine-tune open-source web agents on the refined synthetic data to adapt them to the target environment. Experimental results demonstrate that `SynthAgent` outperforms existing synthetic data methods, validating the importance of high-quality synthetic supervision.

1 Introduction

Large language models (LLMs) with multimodal capabilities have enabled a new wave of web agents capable of autonomously completing complex tasks on the internet (Hong et al., 2024; He et al., 2024; Agashe et al., 2025; Li et al., 2025; Yang et al., 2025b). These agents take user instructions and then interact with websites to accomplish tasks, showing promising results on standardized benchmarks (Zhou et al., 2024; Yao et al., 2022; Wei et al., 2025a; Deng et al., 2023). However, a persistent challenge is that web agents struggle to adapt to new websites not seen during training (Zhou et al., 2024; Pahuja et al., 2025; He et al., 2024), because new environments often lack sufficient task demonstrations. And issuing environment specific tasks and collecting trajectories demonstrated by human on every new website can be prohibitively expensive. Existing training datasets are either limited to a few domains or lack diversity (Xu et al., 2024; Deng et al., 2023; Xie et al., 2024; Chen et al., 2024a), thus when an agent is deployed on a new website, it could frequently encounter unfamiliar states or tasks for which it has no experience. This raises the challenge of how to effectively adapt web agents to new environments with minimal human involvement.

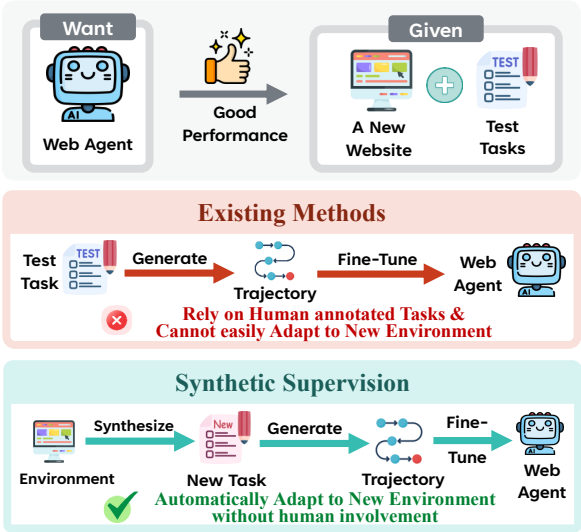

Figure 1: The web agent is expected to adapt and operate on a new environment. Some existing works choose to train on tasks from test set due to the scarcity of environment specific data. In contrast, this paper offers an approach for fully synthetic supervision to adapt agents to such new environment without human involvement.

A straightforward way to improve an agent's performance on a new website environment is to collect more environment specific training data. However, traditional agentic data collection relies on human experts or manually scripted tasks (Zhou et al., 2024; Lu et al., 2024; Deng et al., 2023; Mitra et al., 2024). Such approaches are labor-intensive and time consuming, which cannot be easily scaled.

This may lead to significant gaps between training data and the real-world environments where agents are deployed. Without abundant experience in the new and unfamiliar environment, it is challenging for agents to adapt and perform effectively.

To address the data scarcity issue in web agents, as shown in Figure 1, synthetic data generation (Xu et al., 2024; Su et al., 2025; Sun et al., 2025; Pahuja et al., 2025) has emerged as a promising solution, which uses LLMs to automatically generate and collect data for training agents. However, we identify two fundamental limitations in existing web agent data generation pipelines that prevent scaling high-quality synthetic data: (1) Task hallucinations: methods like OS-Genesis (Sun et al., 2025) propose tasks from single-step observations, often generating goals that reference non-existent elements or impossible states. (2) Trajectory noise: methods like Explorer (Pahuja et al., 2025) propose a coarse task from the homepage and then continuously refine underspecified tasks during execution, causing agent wandering and producing trajectories where early segments are misaligned with the final task. These issues are not merely quality problems but stem from a design tension: task quality requires environment grounding, yet grounding during execution introduces trajectory noise.

In this paper, we propose `SynthAgent`, a fully synthetic supervision framework that resolves this tension through dual refinement: tasks are refined during execution to mitigate task hallucinations, and trajectories are subsequently refined to mitigate potential noise introduced by task edits. Specifically, our method consists of four stages: (1) First, we synthesize diverse tasks through the proposed categorized exploration that systematically covers functional groups of web elements, improving both task diversity and exploration coverage. (2) Then, during trajectory collection, we apply task refinement triggered by explicit conflict detection, correcting hallucinations while minimizing unnecessary task changes. (3) After collection, we conduct trajectory refinement with global context, mitigating noise and misalignments introduced by task edits or agent wandering. (4) Finally, we fine-tune open-source web agents on the refined synthetic data. The key insight is that task and trajectory refinement are synergistic: task refinement enables feasible execution but introduces noise, which trajectory refinement subsequently mitigates.

In summary, our contributions are three-fold:

(1) We propose `SynthAgent`, a fully synthetic supervision framework for effectively adapting web agents to new environments without test task leakage or human involvement.

(2) We identify a critical tension in existing methods: environment grounding during task execution inevitably introduces trajectory noise. We resolve this through a dual refinement strategy. We also propose categorized exploration to enhance task diversity and coverage.

(3) Extensive experiments show that `SynthAgent` substantially outperforms existing synthetic data methods across diverse websites, highlighting the effectiveness of high-quality synthetic supervision in adapting web agents.

2 Related Work

2.1 Web Agent

Recent advances in LLMs have driven interest in developing agents that combine reasoning and interaction (Wei et al., 2022; Yao et al., 2023a; Wang et al., 2022; Lightman et al., 2023; Guo et al., 2025). ReAct (Yao et al., 2023b) introduces the interleaving of reasoning and actions after observations, while following works (Gao et al., 2023; Hong et al., 2024; Wu et al., 2024; Nakano et al., 2021; Yang et al., 2025b) explore tools integration, planning, and observation-action alignment to enhance agent capabilities. However, these typically rely on human annotation, which is costly to scale and adapt to new environments. Another series of works (Lu et al., 2025; Qiu et al., 2025; Wei et al., 2025b) use reinforcement learning to train web agents, but they often require supervision from annotation and are impractical for complex and realistic websites due to the cost of online training. Using synthetic data to train web agents has gained traction as a scalable alternative, especially for new environments (Wang et al., 2024; Liu et al., 2025).

2.2 Data Synthesis

Data synthesis has emerged as a powerful paradigm for addressing data scarcity across various fields, allowing models to learn from automatically generated examples rather than costly human annotations. Early works such as Self-Instruct (Wang et al., 2023a) and Alpaca (Taori et al., 2023) leverage advanced but closed-source LLMs to bootstrap instruction tuning data to train smaller open-source models. Meanwhile, other studies (Zelikman et al., 2022; Wang et al., 2023b; Ge et al., 2024; Zhao

et al., 2025) explore synthesizing large-scale training data to enhance the reasoning performance of smaller LLMs. These successes demonstrate the promising potential of synthetic data for adapting models to new tasks and domains. However, in agent scenarios, many works focus solely on synthesizing trajectories while directly using tasks from the test set to train the model (Chen et al., 2024b; Zhou et al., 2024; Zhang et al., 2025), raising serious concerns about test set leakage. Fortunately, recent works (He et al., 2024; Su et al., 2025; Sun et al., 2025; Pahuja et al., 2025) have explored data synthesis for web agents from scratch, including both task and trajectory synthesis.

2.3 Synthetic data for Web Agents

Synthetic data has been increasingly used to train web agents (Wang et al., 2023a; Taori et al., 2023; Xu et al., 2023; Wang et al., 2023b). He et al. (2024) introduce self-instruct for agentic task generation, where it mainly operates on the surface of websites, thus the synthesized tasks are often simple and repetitive. AgentTrek (Xu et al., 2024) collects training data by scraping web tutorials from offline corpus, which can be outdated and not feasible for new environments. Synatra (Ou et al., 2024) generates offline HTML trajectories by injecting web knowledge from external sources (e.g., wikiHow) without interacting with real websites, which limits the grounding of synthesized data. NNetNav (Murty et al., 2024) employs an LLM-based trajectory labeler to prune exploration based on instruction hierarchies, but does not address data quality issue. WebSynthesis (Gao et al., 2025) employs a learned world model of web interfaces to simulate interactions, but its LLM-based environment can introduce additional hallucinations.

Two recent works are most related to ours. Explorer (Pahuja et al., 2025) refines underspecified initial tasks during execution, but produces noisy trajectories due to wandering and endless refinement. OS-Genesis (Sun et al., 2025) pioneered reverse task synthesis through random GUI exploration, but suffers from hallucinations in single-step-based task proposals and inefficient coverage. In contrast, `SynthAgent` introduces structural improvements: (1) categorized exploration for systematic coverage and improved task diversity, (2) task refinement that detects and corrects hallucinations based on actual observations, and (3) trajectory refinement with global context that removes noise from task edits or agent wandering.

3 Method

As shown in Figure 2, the proposed `SynthAgent` framework consists of four main steps: (1) Task Synthesis with Categorized Exploration, (2) Task Refinement during Trajectory Collection, (3) Trajectory Refinement, and (4) Agent Fine-tuning. In the following, we describe each step in detail.

Problem Setup. We view a website as a partially observable environment $\mathcal{E}$. The task τ specifies a goal that requires a web agent to interact with the environment to accomplish. At step t, the agent receives a multimodal observation o_t (i.e., textual accessibility tree (Zhou et al., 2024) and visual screenshot of the webpage) and outputs an action $a_t \in \mathcal{A}$ (e.g., CLICK, TYPE, SCROLL and etc.). Let $h_t = (o_1, a_1, \ldots, o_t)$ denote the trajectory, i.e., the sequence of observations and actions taken by the agent to complete the task. In this paper, our goal is to synthesize a dataset $\mathcal{D} = \{(\tau^i, h^i)\}_{i=1}^N$ of tasks τ^i and corresponding trajectories h^i on a previously unseen environment, and then adapt an open-source web agent to this environment by supervised fine-tuning.

3.1 Task Synthesis

A key challenge in task synthesis is achieving both diversity (covering different functionalities) and feasibility (avoiding hallucinated goals). Prior work explores environments via random interactions with the environment (Sun et al., 2025), which produces redundant interactions and limited coverage. We address this through categorized exploration that systematically partitions web elements by their functional roles. Specifically, at each visited page o_t, we classify interactive elements (buttons, inputs, links, etc.) with their names and roles into functional categories (e.g., Account Management, Search & Filters, Shopping Content) by prompting an LLM with the page structure. From each category, we uniformly sample up to 2 unvisited elements to interact with, collecting interaction triplets (o_t, a_t, o_{t+1}) where a_t is the corresponding action for the sampled element and o_{t+1} is the resulting page. We maintain a URL pool to track discovered pages for deeper exploration.

For each triplet, we prompt the LLM to propose a high-level task τ that (1) is achievable from o_t through multi-step interactions and (2) is likely completable in o_{t+1} or subsequent pages. The LLM imagines a broader objective where a_t is one step towards the goal. This design ensures synthe-

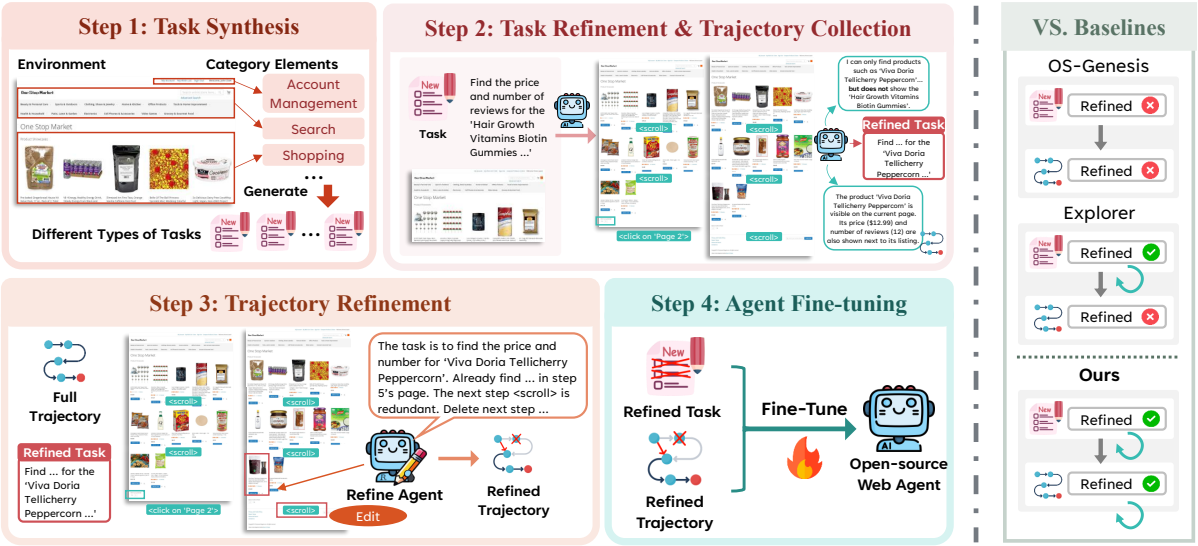

Figure 2: Overview of SynthAgent compared to baseline methods OS-Genesis (Sun et al., 2025) and Explorer (Pahuja et al., 2025). SynthAgent consists of four steps: (1) Task Synthesis, generating diverse tasks via categorized exploration; (2) Task Refinement, refining tasks during trajectory collection to avoid task hallucinations; (3) Trajectory Refinement, globally refining collected trajectories to remove noisy actions; and (4) Agent Fine-tuning, adapting web agents using fully synthetic data. The right panel emphasizes the advantage of our dual refinement strategy over baseline methods, leading to higher-quality synthetic web agent data.

sized tasks are environment-grounded and diverse, covering various functional categories and interaction depths. Statistics show our exploration yields an average of 6 functional categories per page with stable granularity across different websites. See detailed statistics in Appendix A.2.

3.2 Task Refinement

Why task refinement is necessary. After task proposal, OS-Genesis deploys the agent to interact with the environment to collect trajectories for each synthesized task. Note that its proposed task is based solely on a single interaction, thus it often contains hallucinations (e.g., assuming an option or account state that does not exist), causing the agent to fail to complete the task.

For our method, even with categorized exploration, synthesized tasks may contain hallucinations due to limited observations. A natural solution is to refine tasks during execution when conflicts arise. However, how refinement is triggered matters critically: Explorer (Pahuja et al., 2025) combines task proposal and trajectory collection into one stage. It first proposes a coarse task from the homepage and then refines the task continuously at every step. This design implicitly acknowledges that the initial task is underspecified, meaning the following refinement is more about filling in details rather than correcting hallucinations. Empirically, we observe that Explorer frequently changes the task intent during execution (8.6 times vs. 2.0 times of SynthAgent), which often leads to overly long trajectories that fail to complete within the budget (68.3% samples vs. 6.3% samples of SynthAgent). In contrast, we design conflict-triggered refinement that activates only when explicit mismatches are detected, resulting in only 2.0 refinements on average and 6.3% budget exceedance. The key difference is that our initial tasks are already well-specified, so the refinement should focus on correcting hallucinations rather than continuously changing intention.

Refinement during Trajectory Collection. Let τ_t denote the task specification at time t and $h_t = (o_1, a_1, \ldots, o_t)$ the partial trace. The task is likely to contain hallucinations if it conflicts with the observation on the realistic environment. Thus, we trigger refinement when we detect a conflict with a lightweight predicate $\mathcal{C}(h_t, \tau_t)$=true, where

$$\mathcal{C} := \underbrace{\neg\mathsf{ExistsUI}(h_t, \tau_t)}_{\text{invalid goal}} \vee \underbrace{\mathsf{MissingArgs}(h_t, \tau_t)}_{\text{missing details}} \vee \underbrace{\mathsf{Stall}(h_t)}_{\text{stalled}} . \quad (1)$$

The first term "ExistsUI" fires when element implied by τ_t are absent or contradict observations (e.g., referenced item does not exist). The second

term "MissingArgs" detects when τ_t is underspecified and lacks essential parameters (e.g., username when login) that cannot be inferred from h_t. The third term "Stall" detects lack of progress: three consecutive no-op transitions, or the same navigation/error loop encountered twice.

Upon a trigger, we call the LLM to follow four evidence-driven principles to refine the task: (1) concretize missing details, (2) align with actual observations, (3) downscope or simplify when blocked, and (4) preserve a similar task category. The replacement task τ_{t+1} is expected to resolve the conflict while remaining as close as possible to the original intent. The following actions are then taken to continue trajectory collection in order to complete the new task τ_{t+1}. After execution, the final specification is the task $\tau^\star$ along with the collected trajectory h_T. Note that the refinement process can introduce discontinuities and noise into the trajectory, which we address in the next step.

3.3 Trajectory Refinement

Why Trajectory refinement is necessary. Task refinement during execution enables feasible task completion but introduces a side effect: trajectory segments collected under earlier task variants become misaligned with the final task $\tau^\star$. Additionally, even without task edits, agents may wander through dead-ends or repeat no-op actions. This motivates trajectory refinement as a post-hoc step that leverages the global view of the complete trajectory and the final task to remove noise. Critically, task and trajectory refinement are synergistic: the former ensures task feasibility while introducing noise, which the latter subsequently mitigates.

To convert such a trajectory h into a cleaner one aligned with $\tau^\star$, `SynthAgent` introduces a post-hoc offline trajectory refinement step without re-interacting with the environment. Thanks to the global view of full trajectory and final task, we consider following edits: **Remove**(i) deletes task-irrelevant or redundant steps (e.g., repeated Scroll); **Reorder**(i, j) swaps two locally commutable steps when their targets are independent and have no interfering effects (e.g., open filter then set sort); **Drop**(h_T) discards the entire trajectory if too noisy; and **Keep**(h_T) retains the trajectory as is if already well-aligned. These edits are proposed by prompting the LLM with $(\tau^\star, h_T)$, where operations are applied conservatively to avoid unintended state changes. The refined pair $(\tau^\star, h_T^\star)$ is less noisy and more suitable for training.

3.4 Agent Fine-tuning

After obtaining a high-quality dataset of refined task-trajectory pairs $\mathcal{D} = \{(\tau^\star, h^\star)\}^N$, we fine-tune the open-source web agent f to adapt it to such new environment with fully synthetic supervision. We split sample with T steps into a sequence of training examples $\{(\tau^\star, \{o_{\leq t}, a_{<t}\}, a_t)\}_{t=1}^T$, where the model learns to predict the next action a_t given the task description $\tau^\star$ and the historical context of observations and actions $\{o_{\leq t}, a_{<t}\}$. The historical context window is empirically set to 3 considering training cost and inference latency. By using the standard supervised fine-tuning (SFT) paradigm, we optimize the model f as follows:

$$\mathcal{L}_{\text{SFT}} = \mathbb{E}_{(\tau^\star, h^\star)\sim\mathcal{D}} \left[-\sum_{t=1}^{T} \log p_\theta(a_t | \tau^\star, o_{\leq t}, a_{<t}) \right]. \tag{2}$$

After fine-tuning, the agent is expected to better understand the new environment and complete environment specific tasks more effectively, achieving the goal of adaptation.

4 Experiments

4.1 Experimental Setup

Environment & Benchmark. We conduct experiments in two benchmarks: (1) WebArena (Zhou et al., 2024), a suite of five websites: e-commerce (Shopping), content management (CMS), social forum (Reddit), developer platform (Gitlab), and map navigation (Maps), which are stable and controllable. (2) Online-Mind2web (Xue et al., 2025), an online benchmark spanning 136 live websites, which are more realistic and diverse. The environments provide simplified html code (accessibility tree) and screenshots as observations, and accept common browser interactions listed in Table 6. Generally, WebArena is more challenging due to its tasks often requiring authentication, thus involving more complex operations.

Baselines & Models. We compare our approach against several strong baselines for synthesis of web agent data, including: (1) Self-Instruct (Wang et al., 2023a), which directly generates tasks from a few seed examples via prompting LLMs. (2) OS-Genesis (Sun et al., 2025), which synthesizes tasks from single-step environment changes with randomly exploring the environment. (3) Explorer (Pahuja et al., 2025), which synthesizes tasks and trajectories by iteratively refining tasks

during trajectory collection. All baselines are re-implemented using the same LLM of GPT-4.1 [1] for both task and trajectory synthesis. We select two popular open-source multimodal LLMs for agent fine-tuning: Qwen2.5-VL-7B-Instruct (Team, 2025) and UI-TARS-1.5-7B (Qin et al., 2025).

Implementation Details. For both data synthesis and evaluation, we set a maximum step budget of 30 per episode, following recommendations from (Zhou et al., 2024). All methods are synthesizing up to 500 task-trajectory pairs per website for agent fine-tuning. During execution and training, they are using the same prompt template and action space. We also use the same GPT-4.1 for both synthesis and refinement for `SynthAgent`. We use a context window of 3 most recent steps for efficiency. For agent fine-tuning, we mix synthesized data from five websites to train a single model with a learning rate of 1e-5 and batch size of 32 for 3 epochs. More details are in Appendix A.1.

4.2 Main Results

The overall performance of different methods on WebArena is shown in Table 1. From the results, we have the following observations: (1) Training on test set tasks (SFT) provides an informative upper bound for the following experiments using synthetic data. (2) Self-Instruct struggles without environment grounding, showing limited improvements over base models. This indicates the importance of interacting with the environment for creating environment specific tasks. (3) Explorer performs surprisingly poorly, even degrading base model performance. This is primarily because its continuous refinement strategy produces overly long trajectories, 68.3% of which exceed the step budget without task completion. This is because continuous task refinement strategy frequently changes task goals and adds new details during execution, causing the agent to deviate from the original objective. Additionally, the original Explorer operates on pages without authentication, limiting its applicability to complex websites, while our re-implementation added the authentication support for fair comparison (details in Appendix A.4). (4) Our `SynthAgent` substantially improves web agent adaptation with fully synthetic supervision, consistently outperforming all baselines. Compared to base models and OS-Genesis, we achieve average absolute gains of $+10.2$ and $+5.1$, respectively. `SynthAgent` also shows closer gap to the upper bound SFT, demonstrating its effectiveness in synthesizing high-quality data.

Generalization. Beyond the controlled offline WebArena environments with only five websites, we further test the generalization of `SynthAgent` on Online-Mind2Web, without any synthesis for targeted websites or re-training in Table 2. We again observe that `SynthAgent` outperforms other baselines across three different judges and is even competitive with GPT-4.1. These results on both offline (WebArena) and online (Online-Mind2Web) benchmarks suggest that our comprehensive exploration and dual refinement design enables the agent to generalize well on diverse websites rather than overfitting to several fixed environments.

5 Analysis

In this section, we analyze the quality of synthesized data, validate each component's contribution, and examine data scaling of `SynthAgent`.

5.1 Synthetic Data Quality

We first analyze the quality and diversity of synthetic data in Table 3. To better understand the characteristics of synthesized tasks, we also visualize their embeddings using t-SNE (van der Maaten and Hinton, 2008) in Figure 3, where points closer in the plot indicate more similar tasks, while points farther apart indicate greater diversity.

Task Diversity & Quality. Synthesized task diversity is the essential prerequisite for effective agent adaptation. Self-Instruct produces points heavily clustered in a small region of the embedding space, which indicates, without real environment grounding, LLM "imagined" tasks results in narrow task variations. In contrast, OS-Genesis achieves better diversity (83) through random environment exploration. Explorer exhibits the poorest diversity (54) with a highly clustered pattern, indicating that its initial coarse-grained task proposals from the homepage constrain the space of possible tasks. `SynthAgent` achieves the highest diversity score after task refinement, with points closely resembling the distribution of human written tasks. This suggests that our categorized exploration and task refinement strategies effectively covers different functions and interaction depths. Detailed statistics and case study of categorized exploration are provided in Appendix A.2 and B, respectively.

[1] https://openai.com/index/gpt-4-1/

Method	Train on	Shopping	CMS	Reddit	Gitlab	Maps	Overall
GPT-4.1	-	30.91	24.56	15.38	26.79	21.9	25.22
Qwen	-	13.71	8.24	9.43	6.18	5.50	8.80
+SFT	Test Dataset	27.27	12.28	19.23	10.71	12.50	16.37
+Self-Instruct	Synthetic Data	18.18	8.77	3.85	12.50	9.38	11.50
+OS-Genesis	Synthetic Data	14.55	10.53	11.54	16.07	12.5	13.27
+Explorer	Synthetic Data	10.91	3.51	0.00	1.82	3.12	4.44
+Ours	Synthetic Data	**20.00**	**21.05**	**15.38**	**19.64**	**28.12**	**20.80**
UI-TARS	-	12.73	8.77	3.85	7.14	9.38	8.85
+SFT	Test Dataset	25.45	22.81	19.23	21.43	28.12	23.45
+Self-Instruct	Synthetic Data	20.00	8.77	7.69	14.55	12.50	13.33
+OS-Genesis	Synthetic Data	**21.82**	12.28	7.69	14.29	12.50	14.60
+Explorer	Synthetic Data	11.43	4.88	0.00	3.23	0.00	4.96
+Ours	Synthetic Data	20.00	**14.04**	**19.23**	**16.07**	**18.75**	**17.26**

Table 1: Performance comparison on WebArena benchmark. "SFT" means directly use the tasks from test set paired with trajectories collected by GPT-4.1 for agent fine-tuning. The best performance among synthetic methods is highlighted in bold. In most cases, ours `SynthAgent` outperforms other methods with synthetic data.

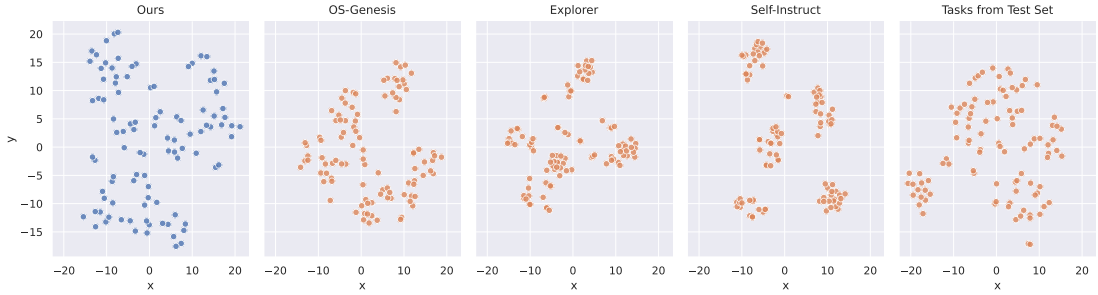

Figure 3: t-SNE visualization of synthesized tasks. Tasks from test set are written by human.

Method \ Judge	GPT-4.1	GPT-5.1	WebJudge	Avg.
GPT-4.1	28.00	18.67	27.00	24.56
Qwen	16.90	6.49	19.48	14.29
Self-Instruct	17.67	13.00	19.67	16.78
OS-Genesis	19.53	11.00	19.33	16.62
Explorer	8.33	3.67	13.33	8.44
`SynthAgent`	**31.67**	**15.67**	**23.33**	**23.56**

Table 2: Task success rate on the Online-Mind2Web benchmark under three different judges (GPT-4.1, GPT-5.1, and WebJudge (Xue et al., 2025)). Our method `SynthAgent` consistently outperforms baselines across judges, and is competitive with the GPT-4.1.

Method	Task			Trajectory	
	Quality	# Refine	Diversity	#Steps	Quality
Self-Instruct	59.3 (51.8)	-	69 (69)	6.3	41.7 (45.7)
OS-Genesis	56.9 (50.2)	-	65 (76)	5.1	52.0 (55.6)
Explorer	73.1 (72.0)	8.6	46 (56)	20.5	36.4 (43.5)
`SynthAgent`	72.6 (71.5)	2.0	82 (88)	7.5	82.6 (83.4)
w/o TR	68.5 (69.1)	-	84 (85)	7.0	70.2 (71.9)
w/o JR	72.0 (70.4)	2.0	82 (88)	8.8	68.7 (75.1)

Table 3: Statistics of synthesized data rated by GPT-5.1 (average across GPT-4.1, GPT-5.1, Qwen3-30B (Yang et al., 2025a) in parentheses). Quality is rated on a scale of 0-100. Diversity is measured by 100 samples as a whole. TR and JR stand for task and trajectory refinement. More results are in Appendix A.5.

Trajectory Quality. Table 3 shows that our method achieves the highest trajectory quality (82.6, avg. 83.4 across judges), substantially outperforming other methods. The improvement is enabled by our dual refinement: task refinement ensures tasks are feasible and grounded in observations, while trajectory refinement with global context mitigates noise from execution and task edits. Notably, our trajectory refinement applies edits conservatively where reorder accounts for only 4.1% of edit operations (vs. 39.3% for remove), gated by explicit independence checks. A head-to-head evaluation on 100 reordered trajectories shows that reordered versions achieve higher win rates (42% vs. 27%) and quality scores (68.9 vs. 62.1), confirming that constrained reordering improves trajectory quality without introducing instability. Detailed results are in Appendix A.3.

To understand why Explorer underperforms despite similar refinement, we analyze trajectory com-

Method	Completed	Failed	Exceeded	Costs $
Explorer	30.5	1.1	68.3	0.22
SynthAgent	96.5	3.5	0.0	0.13
w/o TR	47.4	49.2	3.4	0.09
w/o JR	73.8	17.0	9.2	0.12

Table 4: Comparison between Explorer and SynthAgent in terms of trajectory completion rates judged by GPT-4.1. Completed, Failed, and Exceeded are the percentages of trajectories that successfully completed the task, failed to complete the task, or exceeded the maximum step budgets, respectively. Costs are the average API cost per trajectory.

Method	Shopping	CMS	Reddit	Gitlab	Maps	Overall
Qwen	13.71	8.24	9.43	6.18	5.50	8.80
SynthAgent	**20.00**	**21.05**	**15.38**	**19.64**	**28.12**	**20.80**
w/o CE	18.18	14.04	**15.38**	16.00	25.00	17.26
w/o TR	16.37	14.04	**15.38**	16.07	18.75	15.93
w/o JR	18.18	12.28	11.54	**19.64**	21.88	16.81
w/o TR&JR	**20.00**	14.04	**15.38**	16.07	12.50	15.93

Table 5: Ablation study of SynthAgent. CE, TR, and JR stand for categorized exploration, task refinement, and trajectory refinement, respectively.

pletion rates in Table 4. Explorer's continuous refinement changes task intentions at each step, causing 68.3% of trajectories to exceed the step budget. In contrast, SynthAgent achieves 96.5% completion rate after trajectory refinement with only 60% of Explorer's OpenAI API cost.

5.2 Ablation Study

To quantify the contribution of each component in SynthAgent, we conduct an ablation study in Table 5. Removing categorized exploration (w/o CE) causes a 3.54% drop, with notable degradation on CMS (−7.01%), confirming that systematic coverage of functional groups is essential for task diversity. Removing task refinement (w/o TR) leads to a 4.87% drop, the largest among all components. This is because without correcting hallucinations during execution, many synthesized tasks can be infeasible. Removing trajectory refinement (w/o JR) results in a 3.99% drop, indicating that post-hoc noise removal is crucial for producing clean supervision signals. Interestingly, removing both TR and JR (w/o TR&JR) on Shopping improves while on Maps degrades. This confirms that TR and JR are synergistic: TR introduces some noise to the trajectory due to task edits during execution which JR subsequently removes, and their combination achieves the best overall performance.

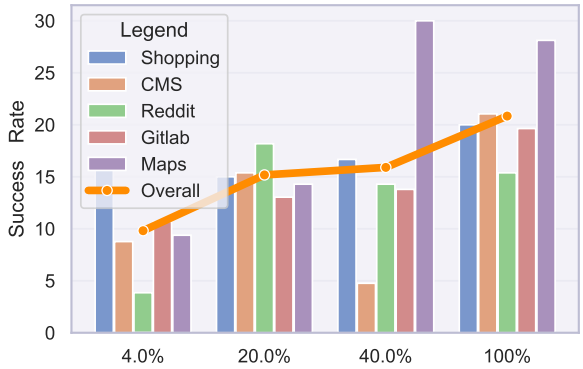

Figure 4: Performance of SynthAgent across different websites with varying synthesis data amounts.

5.3 Data Scaling

To evaluate the scalability of SynthAgent, we measure across different data amounts in Figure 4, scaling from 4% (20 tasks per website) to 100% (500 tasks per website). The results demonstrate consistent performance gains as more synthetic data is used, with the average success rate increasing from approximately 10.6 at 4% data to 20.8 at 100% data. Notably, different websites exhibit varying performance gains, which may be attributed to their inherent task complexity and capacity. For example, Map environment reaches the peak at 40% data, while CMS and Gitlab continue to steadily improve. The overall scaling behavior shows that SynthAgent maintains data quality consistently as the synthesis scales with the website complexity.

6 Conclusion

In this paper, we study how to adapt web agents to new environments where environment-specific tasks and demonstrations are scarce. We identify that existing synthetic data generation methods suffer from severe quality issues: tasks often contain hallucinations and collected trajectories are noisy. To address these issues, we propose SynthAgent, a fully synthetic supervision framework that improves data quality through dual refinement of both tasks and trajectories. In addition, we introduce categorized exploration to systematically cover web elements and interactions, enhancing the diversity and efficiency of task synthesis. Extensive experiments and analysis demonstrate the effectiveness of our approach in adapting web agents to new environments. Beyond adaptation, we believe SynthAgent is a valuable resource for the rapidly emerging field of agentic reinforcement learning where large-scale and diverse tasks are crucial.

Limitations

While this paper demonstrates promising results for adapting web agents, we acknowledge several limitations that may help future research.

(1) We evaluate on both offline (WebArena (Zhou et al., 2024)) and online (Online-Mind2Web (Xue et al., 2025)) benchmarks. WebArena provides realistic complexity with multi-page interactions, authentication, and full functionality, while Online-Mind2Web spans 136 live websites for generalization testing. Other offline benchmarks such as Mind2Web (Deng et al., 2023) and WebShop (Yao et al., 2022) provide only static snapshots or simplified environments, preventing the interactive exploration that `SynthAgent` relies on for data synthesis. Future work may explore deploying data synthesis on live websites, though this presents significant challenges: live sites constantly change and often require CAPTCHA solving, extensive automated visits may trigger security mechanisms, and synthesized data from live websites cannot be easily released due to copyright and privacy constraints, limiting reproducibility.

(2) For task and trajectory synthesis, we rely solely on GPT-4.1 without extensive hyperparameter tuning though we provide quality assessments using different LLMs. We do not optimize agent execution parameters such as the maximum execution steps, exploration depth, or sampling strategies during data collection. Additionally, we do not explore alternative prompting strategies or more advanced LLMs that may further improve synthesis quality.

(3) Our agent fine-tuning employs standard SFT on the synthetic data. We do not explore more advanced training algorithms such as Direct Preference Optimization (Rafailov et al., 2023), or online reinforcement learning methods that could potentially yield better performance. However, these advanced approaches typically require constructing verification mechanisms or reward models, which is beyond the scope of this paper.

Ethics Statement

Our research is conducted entirely on controlled, self-hosted environments. However, the proposed method could potentially be misused on live websites to enable unauthorized data scraping, or account related security issues. We strongly discourage such misuse and emphasize that all deployment of web agents must comply with applicable laws, website terms of service, and ethical guidelines. Future work should incorporate appropriate safeguards and detection of sensitive information. We are committed to responsible research and encourage the community to develop autonomous web agents with consideration for security, privacy, and ethical implications.

Use of AI Assistants

We acknowledge the use of AI Assistants in the writing of this manuscript, especially for correcting grammatical errors and improving sentence fluency. All the generated content was carefully reviewed and verified by the authors.

References

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

Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Liuyi Chen, Yilin Bai, Zhigang He, Chenlong Wang, Huichi Zhou, Yiqiang Li, and 1 others. 2024a. Gui-world: A dataset for gui-oriented multimodal llm-based agents. *arXiv e-prints*, pages arXiv–2406.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024b. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. *Preprint*, arXiv:2211.10435.

Yifei Gao, Junhong Ye, Jiaqi Wang, and Jitao Sang. 2025. Websynthesis: World-model-guided mcts for efficient webui-trajectory synthesis. *Preprint*, arXiv:2507.04370.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2024. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. *arXiv preprint arXiv:2410.19609*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. International Conference on Learning Representations, ICLR.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, and 1 others. 2025. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, and 1 others. 2025. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *arXiv preprint arXiv:2504.01990*.

Xing Han Lu, Zdeněk Kasner, and Siva Reddy. 2024. WebLINX: Real-world website navigation with multi-turn dialogue. In *Forty-first International Conference on Machine Learning*.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, and 1 others. 2024. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*.

Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D Manning. 2024. Nnetnav: Unsupervised learning of browser agents through environment interaction in the wild. *arXiv preprint arXiv:2410.02907*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Tianyue Ou, Frank F Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. 2024. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. *Advances in Neural Information Processing Systems*, 37:91618–91652.

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

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, and 1 others. 2025. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv preprint arXiv:2505.20286*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö Arık. 2025. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. *arXiv preprint arXiv:2501.10893*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2025. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *Preprint*, arXiv:2412.19723.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Qwen Team. 2025. Qwen2.5-vl.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023a. Self-instruct: Aligning language models with self-generated instructions. *Preprint*, arXiv:2212.10560.

Zhaoyang Wang, Shaohan Huang, Yuxuan Liu, Jiahai Wang, Minghui Song, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and 1 others. 2023b. Democratizing reasoning ability: Tailored learning from large language model. *arXiv preprint arXiv:2310.13332*.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025a. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, and 1 others. 2025b. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. 2024. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents. In *Second Conference on Language Modeling*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. 2025b. Agentoccam: A simple yet strong baseline for LLM-based web agents. In *The Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Ruichen Zhang, Mufan Qiu, Zhen Tan, Mohan Zhang, Vincent Lu, Jie Peng, Kaidi Xu, Leandro Z Agudelo, Peter Qian, and Tianlong Chen. 2025. Symbiotic cooperation for web agents: Harnessing complementary strengths of large and small llms. *arXiv preprint arXiv:2502.07942*.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. 2025. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

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. *Preprint*, arXiv:2307.13854.

A Implementation Details

A.1 Environment

Action	Description
`click [id]`	Clicks on an element with the given `element_id`.
`type [id] [text]`	Types the specified text into the field with `element_id`.
`hover [id]`	Moves the cursor over an element with the given `element_id`.
`press [key_comb]`	Presses a keyboard shortcut (e.g., `Ctrl+V`, `Cmd+V`).
`scroll [up|down]`	Scrolls vertically in the given direction.
`goto [url]`	Navigates directly to the specified URL.
`go_back`	Navigates to the previous page in browser history.
`go_forward`	Navigates to the next page in browser history.
`none [summary]`	Produces a summary or final answer without any browser action.
`stop [reason]`	Stops execution when the task is impossible or inconsistent.

Table 6: Available atomic low-level browser interactions and their descriptions for WebArena.

Action Space. All the actions that are executable in the trajectories are shown in Table 6 with regular web actions. The "none" action signals the final step for answering the task. For example, if the task is to find the cheapest product on a shopping website, the last action would be "none" with the summary of the cheapest product as the observation. The "stop" action is used to terminate the trajectory when the agent exceeds the maximum step budget or encounters an unrecoverable error.

Evaluation. WebArena (Zhou et al., 2024) contains 812 challenging web navigation tasks derived from 241 human written task templates, including maps, e-commerce, Reddit forums, and software development. Due to the naturally high cost of time interacting with the browser, we follow prior work (Sun et al., 2025) and use 226 tasks for evaluation, where we include only one task per template and exclude tasks that require cross-website navigation. The numbers of tasks for each website are 55, 57, 26, 56, and 32 for Shopping, CMS, Reddit, Gitlab, and Maps, respectively. Note that for training with tasks from test set (SFT method in Table 1), we still use the full set of 812 tasks to enrich the training data as much as possible.

Online-Mind2Web (Xue et al., 2025) contains 300 realistic and diverse tasks spanning 136 live websites. Each task is requiring the agent to interact with real websites. The evaluation is performed with LLM-as-a-Judge through key point identification, key screenshot identification, and final outcome judgment.

A.2 Categorized Exploration

To understand the robustness and consistency of our categorized exploration, we compute summary

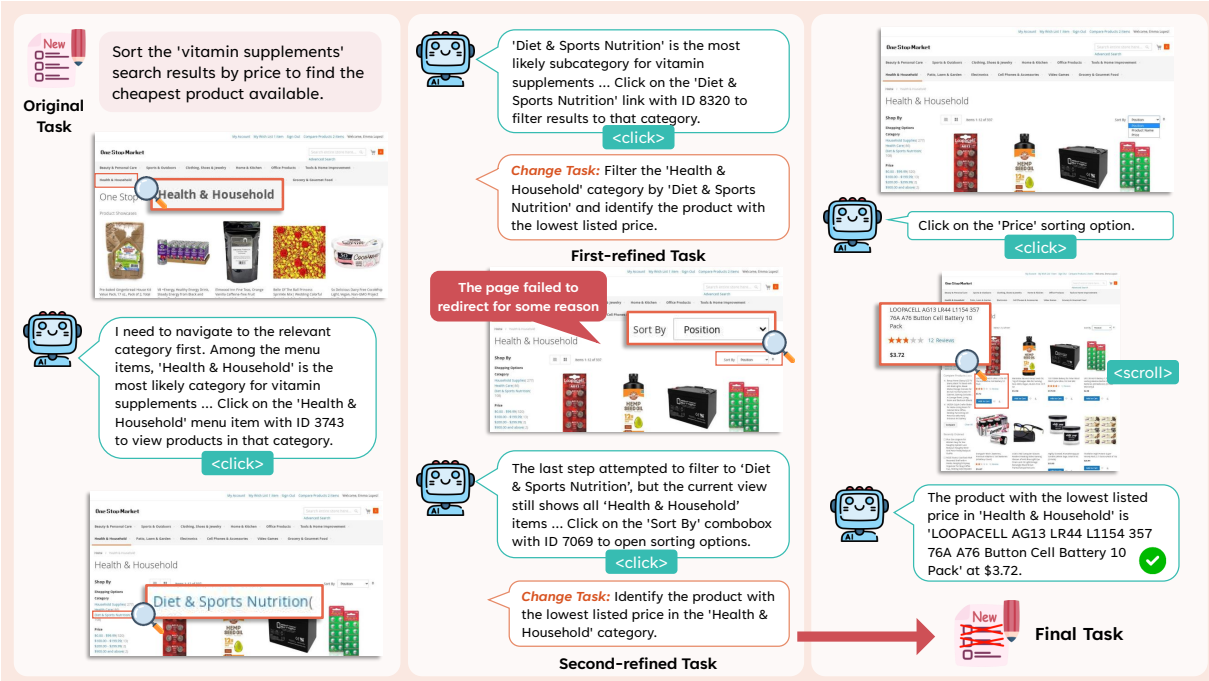

Figure 5: Case study of Task Refinement. During task execution, `SynthAgent` detects when a task becomes infeasible based on the progression of its execution and refines the task accordingly, guided by the observed failure and current environment state.

Env	Avg. Ele	Avg. Static	Avg. Interact.	Avg. Cat.	Avg. Ele/Cat.
Shopping	115.5	62.3	53.3	7.9	6.7
CMS	84.2	37.6	46.6	6.6	7.1
Reddit	78.6	39.7	38.9	5.3	7.3
Gitlab	111.2	48.9	62.3	5.8	10.7
Maps	53.9	24.0	29.9	4.2	7.1
Overall	88.7	42.5	46.2	6.0	7.8

Table 7: Statistics of categorized exploration across all pages visited during task synthesis. "Avg. Ele" denotes the average number of DOM elements per page, "Avg. Static" and "Avg. Interact." are the averages for non-interactive and interactive elements, respectively, "Avg. Cat." is the average number of functional categories identified by the LLM, and "Avg. Ele/Cat." is the average number of elements per category.

Operation	Remove	Reorder	Drop	Keep
Count	982.5	41.8	11.0	1463.9
Percentage	39.3%	1.7%	0.4%	58.6%

Table 8: Distribution of trajectory refinement operations. Reorder accounts for only 4.1% of edit operations (Remove + Reorder), indicating conservative application.

Metric	Original	Reordered
Win Rate (%)	27.0	42.0
Tie Rate (%)	31.0	31.0
Quality Score	62.1	68.9

Table 9: GPT-5.1 head-to-head evaluation of 100 reordered trajectories. Reordered trajectories achieve higher win rates and quality scores.

statistics across all pages visited during task synthesis. For each environment, we measure the average number of elements per page, the split between static and interactive elements, the number of categories, and the average number of elements per category in Table 7.

The relatively stable category granularity across environments suggests that the proposed categorized exploration behaves consistently despite large differences in page structures.

A.3 Trajectory Refinement Statistics

To understand the behavior of our trajectory refinement, we analyze the distribution of edit operations and the effect of reordering in Table 8.

The reorder operation is applied only when an explicit independence check between steps is satisfied, ensuring that swapping does not affect the environment state. Despite its conservative usage (4.1% of edits), reordering provides measurable quality improvements when applied.

A.4 Baseline Implementation

All baselines are re-implemented using GPT-4.1 for fair comparison. For Self-Instruct (Wang et al., 2023a), we follow the standard prompting approach

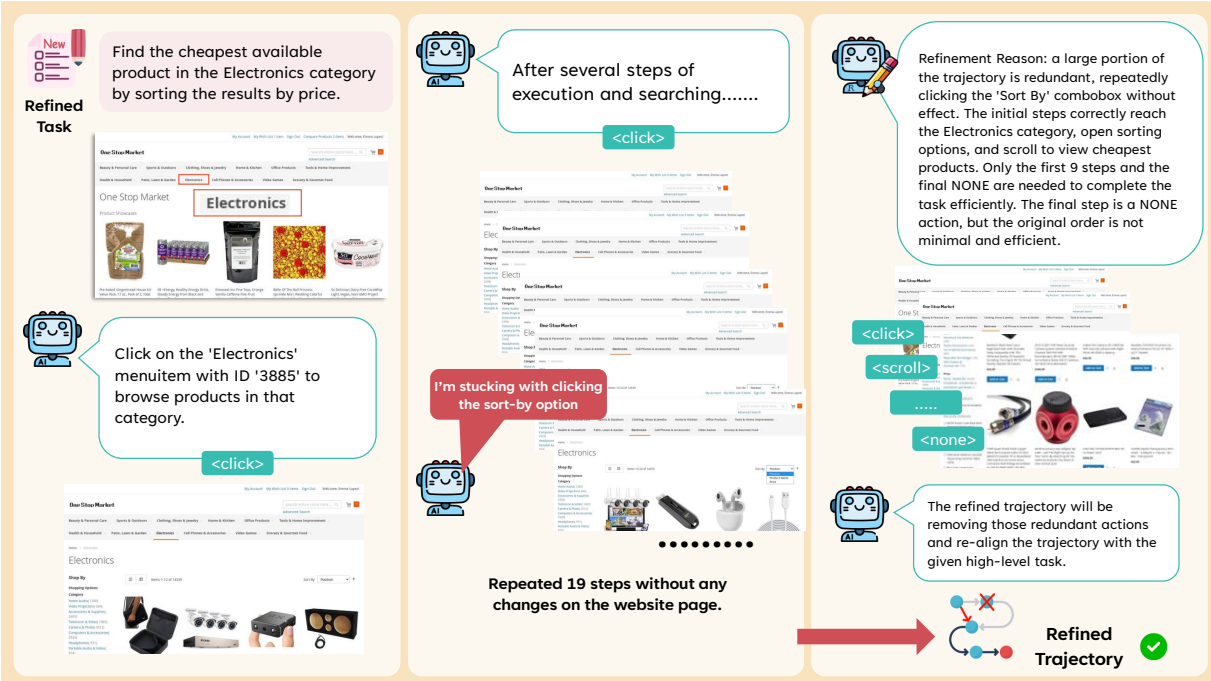

Figure 6: Case study of Trajectory Refinement. After trajectory collection, SynthAgent evaluates the trajectory and refines it by reordering steps or removing unnecessary actions to ensure the trajectory consistently align with the task objective and ends with a successful completion.

to generate tasks from seed examples, as similar implementation by Sun et al. (2025). For OS-Genesis (Sun et al., 2025), we adapt the generation part of the released codebase to WebArena, using random exploration to collect interaction triplets and generate tasks from single-step observations.

For Explorer (Pahuja et al., 2025), we believe our implementation is a faithful adaptation of the released codebase with minimal changes for WebArena compatibility (e.g., aligning browser operations). We additionally implement authentication handling, which the original Explorer does not support, to allow operation on login-gated pages. We have carefully investigated Explorer's poor performance: as shown in Table 4, the dominant failure mode arises from its overly aggressive refinement strategy (8.6 refinements per trajectory on average) that produces long trajectories and frequently exhausts the step budget (68.3% exceed the budget). Through trajectory inspection, we confirmed this behavior matches the algorithmic design rather than an implementation error.

A.5 Trajectory Quality Evaluation

To ensure our quality assessments are not dependent on a single model, we evaluate synthesized data using three different judges: GPT-4.1, GPT-5.1, and Qwen3-30B (Yang et al., 2025a). Results with GPT-5.1, GPT-4.1, and Qwen3-30B are shown in Table 3, Table 10, and Table 11, respectively.

Method	Task			Trajectory	
	Quality	# Refine	Diversity	#Steps	Quality
Self-Instruct	49.2	-	69	6.3	56.6
OS-Genesis	39.8	-	83	5.1	54.1
Explorer	69.8	8.6	54	20.5	48.1
SynthAgent	70.2	2.0	95	7.5	92.5
w/o TR	68.2	-	86	7.0	78.1
w/o JR	69.5	2.0	95	8.8	86.3

Table 10: Trajectory quality rated by GPT-4.1.

Method	Task			Trajectory	
	Quality	# Refine	Diversity	#Steps	Quality
Self-Instruct	46.8	-	68	6.3	38.9
OS-Genesis	53.9	-	80	5.1	60.7
Explorer	73.0	8.6	68	20.5	45.9
SynthAgent	71.8	2.0	88	7.5	75.0
w/o TR	70.5	-	85	7.0	67.5
w/o JR	69.8	2.0	88	8.8	70.3

Table 11: Trajectory quality rated by Qwen3-30B.

Across all three different judges, SynthAgent consistently produces higher-quality task-trajectory data compared to baselines. While absolute scores vary by judge, the relative ranking of methods remains stable, indicating that our improvements are robust across different evaluation models.

A.6 Task Generation

For SynthAgent, the generation process begins by categorizing low-level actions into distinct functional groups through systematic exploration of the web environment. The agent interacts with the interface and analyzes all visible elements, distinguishing between interactive and non-interactive components and assigning each to an appropriate category. This structured categorization enables the model to generate contextually grounded low-level actions within different functional groups, thereby enhancing its generalization ability. The detailed prompt for this process is shown in Table 7, which is largely adopted from OS-Genesis (Sun et al., 2025). Using the low-level actions, we further generate high-level tasks following Table 8.

A.7 Trajectory Generation

To ensure higher-quality trajectories, SynthAgent divides the collection process into three stages. In the first stage, given the set of collected high-level tasks, SynthAgent executes them in a chain-of-thought (CoT) manner (Wei et al., 2022), as illustrated in Table 9. The model reasons step-by-step over the current web context to generate the next low-level action until the task is completed or deemed infeasible. During the second stage, as the agent interacts with the environment, each high-level task is dynamically analyzed and refined using the task refinement prompt (i.e., prompt at Table 10). SynthAgent continuously evaluates whether the task remains valid given the current observation and refines it when inconsistencies, missing parameters, or execution stalls are detected. In the final stage, SynthAgent assesses the collected trajectories and refines them by dropping those deemed unachievable or reordering intermediate steps when a revised sequence better fulfills the task objective using prompt at Table 11.

A.8 Agent Framework

Our agent execution framework builds upon ReAct (Yao et al., 2023b), instantiating a policy that interleaves CoT reasoning with environment actions in a browser-based setting with textual accessibility tree and visual screenshot as observations. Specifically, given a high-level instruction, the agent performs a sequence of steps to accomplish the task. At each step, the agent first takes observations from the environment based on the history actions and current state which includes the URL, accessibility tree, a set of candidate interactive elements and screenshot, then uses CoT reasoning to generate the next action to take. Execution continues until the model explicitly outputs a termination ("none" action in Table 6), the task is judged unachievable by itself ("stop" action in Table 6), or a preset step budget (30) is exhausted. The used prompt template is shown in Table 9.

A.9 Tasks Analysis

We use LLMs-as-a-judge (Zheng et al., 2023) to evaluate the quality of the generated tasks. We prompt the LLM with Table 12 to assesses how broadly and non-redundantly the tasks cover different user intentions for specific website.

B Case Study

Categorized Category Sets. To further illustrate the functional breadth captured by our categorized categorization, we list representative category sets for each website:

- **Shopping**: Global Navigation, Search and Discovery, Shopping Options / Filtering, Sorting and Viewing Options, Product Browsing, Product Details and Purchase, Cart and Comparison, User Account Management, Product Reviews & Feedback, Subscription.
- **CMS**: Global Navigation, User Account Management, Order Management, Product Management, Tab Navigation, Reporting & Analytics, Search & Filters, Table Controls & Pagination, Admin Controls, Help and Information.
- **Reddit**: Navigation, User Account & Profile, Notifications & Submissions, Content Submission, Content Interaction (Posts & Voting), Comments / Threading / Visibility, Moderation & Toolbox, Forum Subscription & Management, Search & Discovery, Footer / Site Info.
- **Gitlab**: Menu & Utility Actions, User/Account Actions, Repository High-Level Actions, Search & Filtering, Commit History Navigation, Alerts and Notifications, Code Commenting/Review, Project Management, Navigation & Site Structure, Issue Management.
- **Maps**: Authentication/Login, Account Registration, Map Editing, Navigation & Page

Switching, Sharing, Map Control, Community and Support, Map Data, Directions Input & Management, Sidebar/Popup Controls.

Across these environments, our method consistently identifies meaningful, task-relevant functional groups (e.g., navigation, search, account management, content interaction), which supports the claim that categorized exploration is both expressive and robust enough to guide task synthesis beyond simple random exploration.

Task Refinement. Figure 5 illustrates an example of the task refinement process in `SynthAgent`. The original task "Sort the 'vitamin supplements' search results by price to find the cheapest product available", was initially executed based on the agent's understanding of the web interface. However, during interaction, the agent detected that the page failed to redirect to the intended subcategory ('Diet & Sports Nutrition'), making the original goal inconsistent with the current observation. `SynthAgent` automatically analyzed this discrepancy and refined the task to align with the accessible context, updating it to "Identify the product with the lowest listed price in the 'Health & Household' category.". This refinement ensures that high-level task remain executable and contextually valid for existing trajectory.

Trajectory Refinement. Figure 6 presents the trajectory refinement process in `SynthAgent`. The agent's intermediate sequence of the trajectory for the task "Find the cheapest available product in the Electronics category by sorting results by price" contained significant noise and inefficiencies. During execution, the agent became stuck attempting to interact with a non-functional sort option, resulting in 19 repeated steps with no progress on the webpage. The collected trajectory included this futile loop along with redundant scroll actions that did not contribute to task completion. Our trajectory refinement step, equipped with the global view of the full trajectory and final task objective, identified and removed the repetitive clicking attempts on the unresponsive sort interface, consolidated the necessary scroll actions. The refined trajectory retained only the essential 9 steps needed to successfully complete the task: clicking the Electronics category, navigating through sorting options, and scrolling to identify the cheapest product. This refinement ensures that the trajectory aligns precisely with the high-level task while removing noisy actions that would be harmful for the agent during fine-tuning.

```
You are a GUI (Graphical User Interface) Web Agent expert capable of grouping interactive elements
from a web page into high-level user task categories.


**Information**
Current Page:
- URL:
{url}
- Accessibility Tree (only current view, not full page):
{page_context}
- Elements:
{elements}
- Screenshot of the page (only current view, not full page):
<image is provided in the attachment>


**Your Goal**
1. Fully explore the current page and its content to understand its functionality and layout.
2. Categorize ALL provided {element_num} elements into different categories (list[dict]) based on
their natural purpose.
3.  Add a {const_uninteractive_category} category (list[int]) for non-interactive elements that
cannot be CLICK, TYPE, or HOVER.
4. For each category (except {const_uninteractive_category}), decide:
{ "action": choose from [CLICK, TYPE, HOVER], "element_id": id_of_element (int), "value": if TYPE,
provide text to type; else ", "low-level_instruction": concise description of the action }
Example low-level instructions:
- "Click on the 'Add to Cart' button next to the product to add it to your shopping cart."
- "Type 'OpenAI' into the search bar to find relevant articles."
- "Scroll down to view the latest blog posts on the homepage."


5. Provide an appropriate and meaningful value for "value" if the action is TYPE. Examples:
- For a search box, generate a realistic search query.
- For a textbox, generate plausible text according to context.


**Output Requirements**
Return ONLY a JSON dictionary (no commentary) with the following format:
{
"Analysis": "your analysis of the current page state and elements",
"Categorization": {
"category_1": [
{ "action": "xxx", "element_id": int, "value": "xxx or empty string", "low-level_instruction": "a
concise instruction" }, ...other actions... ],
"category_2": [...],
"category_3": [...],
...{different categories}...,
"{const_uninteractive_category}": [element_id_1, element_id_2, ...]
}
}
RETURN ME THE DICTIONARY I ASKED FOR WITHOUT ANY COMMENTARY.
"""
```

Table 7: Prompt for web elements categorization.

```
You are a GUI (Graphical User Interface) expert capable of analyzing interface changes and
envisioning executable tasks or instructions. Given a GUI interface change caused by an action
(e.g., clicking or typing) and the corresponding element highlighted in red boxes, you are required
to analyze the interface and generate related tasks.


Your task is to envision tasks based on the current action and the resulting changes in the
screenshots. The output should include three components:


1. Sub-Instruction: Create a natural language instruction for the current action based on the
interface changes it caused. The instruction should be concise, clear, and actionable, incorporating
specific details critical to the task, such as elements, file names, timestamps, or other relevant
content visible in the screenshots. For example:
- "Click on the 'Add to Cart' button next to the product to add it to your shopping cart."
- "Type 'OpenAI' into the search bar to find relevant articles."
- "Scroll down to view the latest blog posts on the homepage."


2. Analysis: Carefully analyze the before-and-after screenshots step by step, focusing on the
changes caused by the action. Then, examine key elements in both screenshots and consider
possible operations based on these elements. For example: "The previous screen displayed the
main interface of a shopping website, featuring multiple product categories and several showcased
items. After clicking the 'Sign Up' button, the interface transitioned to a login page where an
email and password can be entered to log into an account. The login page also provides other
options, such as recovering a password, creating a new account, or logging in with a Google account."


3. High-Level Instruction: Based on the before-and-after screenshots, the action, and the analysis,
generate a high-level task that you believe can be completed within the current interface. There
are three types of tasks:
- Information seeking: The user wants to obtain certain information from the webpage, such as
product details, reviews, map information, or route comparisons. Please propose clear and specific
questions that need an explicit answer, and avoid asking for summary-type questions, such as
"summarize the information about a product."
- Site navigation: The user wants to navigate to a specific page or state.
- Content modification: The user wants to modify the content of a webpage or its settings.


The high-level instruction should be creative. You need to deeply analyze the elements and
executable actions on the interface to generate realistic, valuable, and executable tasks that
can be completed within the current GUI. The instruction should be specific, actionable, and
goal-oriented, ensuring the task can be completed on the current GUI by including all critical
specifics such as file names, relevant timings, or required details.


Below is a brief description of the current website: {website_intro}
Here are some examples of High-Level Instruction for reference: {task_examples}


Current Action: {current_action_str}
Website Name: {website_name}


Before-action Screenshot: <image is provided in the first attachment> (the action's target element
is highlighted in red box if applicable)
After-action Screenshot: <image is provided in the second attachment>


Please generate tasks that can be completed on the current platform, and avoid tasks that are
unrelated to the current website.


You ONLY need to return a JSON dictionary formatted as follows (no extra commentary):
{
"Sub-Instruction": "xxx",
"Analysis": "xxx",
"High-Level-Instruction": "xxx"
}
RETURN ONLY THE DICTIONARY I ASKED FOR.
}
```

Table 8: Prompt for agentic task generation, adopted from OS-Genesis (Sun et al., 2025).

```
You are a GUI (Graphical User Interface) Web Agent expert capable of long-horizon planning and
executing high-level tasks on a website. Based on the observations and the high-level task to
complete, generate the next low-level instruction.


**Information**
1. High-Level Task (your ultimate goal to finish):
"{high_level_task}"


2. Current Page (only current view, not full page, you may need to scroll to see more):
- URL:
{url}
- Accessibility Tree (Page Context):
{page_context}
- Elements (addressable in this view):
{elements}
- Screenshot (only current view, not full page):
{img_info}


3. History of Actions ({hint_for_history}):
{previous_state_action}


**Critical Rules for Success**
1. Issue only actions valid for the current observation (elements, accessibility tree, screenshot).
2. Propose ONE atomic action per item in your Potential-Actions list; actions must be independently
executable.
3. Prefer element IDs from the current Elements list for CLICK/TYPE/HOVER.
4. Provide meaningful non-empty value if action ∈ {TYPE, SCROLL, GOTO, NONE, STOP}.
5. If the task is complete, use NONE with the final answer in value; do not propose further actions.
6. Be concise, avoid redundant/risky actions; each action must advance the task.
7. If the task is hallucinated/low-quality/impossible, cautiously choose STOP based on
observations/history.


Pseudo-code for deciding STOP:
if high_level_task lacks required info → STOP
if high_level_task contains hallucinations → STOP
if task is inappropriate/harmful → STOP
if multiple (≥3) similar attempts already failed → STOP
else → consider NON-STOP actions


8. First write a "state_observation_summary", then do step-by-step "reasoning", then decide
"next_action".
9. Expect MULTIPLE steps; choose the next action that changes state; continue iteratively.
10. You MUST actively decide the next step; do not choose NONE/STOP unless certain of
finish/impossibility.
11. In "reasoning", explicitly apply the STOP vs NON-STOP pseudo-code.
12. Actively explore alternatives before STOP if current approach stalls.
13. Choose elements strictly from "Elements (addressable in this view)"; justify this choice in
"reasoning".
14. If the page doesn't change after an action, consider SCROLL to reveal more elements.
15. Special note: when typing a date, use "MM/DD/YYYY".


**Output Requirements**
Return ONLY a JSON dictionary (no commentary) with:
{
"state_observation_summary": "1-3 sentence summary of the current state relevant to the task",
"reasoning": "step-by-step reasoning to decide the next action; include rule-based justification
and STOP check",
"next_action": {
"low-level_instruction": "concise string, e.g., click the button with ID 'submit'",
"action": { "type": "XXXX", "element_id": <int or " (if int, it MUST exist in Elements)>, "value":
<string or " per spec> }
}
```

Table 9: Prompt for long-horizon planning and next-action generation of web agent.

```
You are a GUI (Graphical User Interface) Web Agent expert specializing in analyzing interface
changes and determining whether a high-level task should be refined based on a series of agent
actions and observations.

## High-Level Task Categories

1. **Information Seeking** – User aims to retrieve specific information from the website.
- Examples:
- "What is the most expensive product in the 'Electronics' category?"
- "What are the top 5 posts in the 'Technology' forum?"
- "Summarize the reviews for the product 'iPhone 11'."

2. **Site Navigation** – User aims to reach a specific page or site state.
- Examples:
- "Go to the billing page to check the latest transactions."
- "Navigate to the 'Contact Us' page and fill out the form to express interest in joining the
company."
- "Find the wiki page of 'the youngest person to receive a Nobel Prize'."

3. **Content Modification** – User aims to change site content or settings.
- Examples:
- "Create a user account with username 'bob2134' and password '128nxc18zxv'."
- "Post a new article titled 'The Future of AI' in the 'Technology' forum."
- "Create a code repo named 'Agent' and add a README with the text 'This is a code repo for an
intelligent agent.'"

## Refine Rules

### When to REFINE the task
Refine the task if the following situations are observed (cite triggers in "Analysis"):
1. **Invalid or Inconsistent Goal** – target entity/page/action does not exist, cannot be located,
or conflicts with observed facts.
2. **Insufficient Executable Details** – essential parameters are missing and cannot be inferred.
3.  **Stalled or Repetitive Execution** – three or more consecutive actions show no meaningful
change, or same error repeats.

### When NOT to REFINE
- Goal is valid and consistent with observations.
- Essential parameters are available or can be inferred.
- Actions show measurable progress.
- No persistent or repetitive failures detected.

### How to REFINE
If refinement is required (your analysis must reference the below rules):
1. **Concretize Missing Details** – add essential parameters from history or observation.
2. **Align with Reality** – replace hallucinated entities with actual ones found on the site.
3. **Downscope the Goal** – adjust to the next achievable milestone.
4. **Preserve Task Type** – keep within same category unless required otherwise.

## Goal
Ensure the refined task is either already completed or highly likely to complete within the next
1-2 steps.

## Output Requirements
- Format: JSON dictionary only, no commentary.
- Fields:
- "Analysis": Step-by-step reasoning.
- "Need-to-Refine": "yes" or "no".
- "High-Level-Task": Refined task if "yes", else empty string.

## Information
1. Current High-Level-Task: "{current_high_level_task}"
2. Previous High-Level-Tasks (oldest to newest):
<start_previous_high_level_tasks>
{previous_high_level_tasks}
<end_previous_high_level_tasks>

3. History of Actions ({hint_for_history}):
<start_action>
{previous_state_action}
<end_action>

4. Current Page (only current view):
- URL: "{curr_url}"
- Page Context:
<start_context>
{curr_state_context}
<end_context>
- Screenshot: "{img_info}"
–

You ONLY need to return a JSON dictionary formatted as follows (no commentary):
{
"Analysis": "step-by-step reasoning",
"Need-to-Refine": "yes or no",
"High-Level-Task": "refined task if yes, otherwise empty"
}
RETURN ONLY THE DICTIONARY I ASKED FOR.
```

Table 10: Prompt for refining tasks during trajectory collection.

```
You are a GUI (Graphical User Interface) Web Agent expert. Your job is to analyze a high-level task
and its trajectory (sequence of states and actions), assign a quality score, and decide one of:
- "keep": keep the trajectory as-is (already minimal, ordered, and ends with a correct NONE action
with a non-empty value).
- "refine": reorder or delete steps to make the trajectory succeed (final step must be NONE with a
non-empty explanation).
- "drop": discard the trajectory entirely (e.g., irreparable, hallucinatory, impossible, unsafe, or
missing critical information).


A trajectory is structured as:
"Length of trajectory, High-level task, summary of state1, action1, summary of state2, action2,
..."


### Scoring Rubric (0-100)
Evaluate the trajectory on:
1. Goal Alignment (0-25): Steps relevant to the high-level task.
2. Logical Order (0-25): Steps follow a coherent and sensible sequence.
3. Efficiency (0-25): Avoids redundant or unnecessary actions.
4. Success Likelihood (0-25): Likely to end successfully with NONE (non-empty value).

Note: The score is advisory; the final decision (keep/refine/drop) depends on qualitative judgment.


-
### Decision Policy
- Always ensure kept/refined trajectories end with a NONE action and non-empty value.
- If refining, reorder or delete existing steps (do not add new ones).
- Replace STOP with NONE if success is achievable.
- If dropping, do not fabricate NONE; instead, provide a clear drop_reason.

**Indexing and Deletion Rules:**
- Let the trajectory contain K (observation, action) pairs, indexed 0..K-1.
- Reordering: return indices in a new order.
- Deletion: omit indices.
- No duplicates or out-of-range indices. Do not invent new steps.


### Input
{trajectory}


### Output Requirement (STRICT)
Return ONLY one JSON object (no extra text, no code fences):
{
"task": "<exact high-level task string>",
"score": <int>,  // 0-100, advisory only
"decision": "keep" | "refine" | "drop",
"order": [<int>, ...],  // indices in final order
"modify_end": <true|false>,
"append_end": <true|false>,
"final_none_value": "<non-empty explanation for final NONE>",
"drop_reason": "<reason if dropped>",
"modification_reason": "<brief rationale for keep/refine/drop>"
}

### Additional Constraints
- "task" must match the high-level task exactly.
- If decision = keep: order = [0,1,...,K-1], and final step already NONE.
- If decision = refine: must end with NONE and valid order.
- If decision = drop: order = [], final_none_value empty, provide drop_reason.
```

Table 11: Prompt for trajectory quality evaluation and refinement.

```
You are a GUI (Graphical User Interface) Web Agent evaluation expert. Your job is to evaluate the
**diversity** of a set of high-level tasks generated for a single web environment.
Each task represents a distinct user goal that can be accomplished on the same webpage (e.g.,
"Search for a product", "Sort results by price", "View product details"). Your evaluation should
judge how **broad, non-overlapping, and complementary** these tasks are relative to one another.
—
## Input
The input contains multiple high-level tasks that share the same web environment:
{task_list_block}
—
## Scoring Rubric (0-100)
Evaluate the **diversity** of the provided task set using the following criteria:
1. **Intent Variety (0-25):** Do the tasks represent different user intents (e.g., information
seeking vs. navigation vs. modification)?
2. **Action Diversity (0-25):** Do the tasks require different types of GUI interactions (e.g.,
clicking, typing, scrolling, submitting forms)?
3. **Goal Coverage (0-25):** Do the tasks explore different meaningful aspects or functionalities
of the environment?
4. **Redundancy Minimization (0-25):** Are there minimal duplicate or near-duplicate tasks (i.e.,
no rephrasing of the same goal)?
—
## Output Requirement (STRICT)
Return ONLY one JSON object (no extra text, no code fences):
{
"score": <int>,  // 0-100 total diversity score
"subscores": {
"intent_variety": <int>,  // 0-25
"action_diversity": <int>,  // 0-25
"goal_coverage": <int>,  // 0-25
"redundancy_minimization": <int>  // 0-25
},
"analysis": "<short reasoning describing overall diversity and possible overlaps>",
"representative_examples": ["<one or two tasks that illustrate high or low diversity>"]
}
—
## Additional Constraints
- Evaluate ONLY diversity, not task quality or feasibility.
- Consider whether tasks collectively span multiple distinct purposes, operations, or workflows.
- Do NOT propose new tasks, rephrase them, or rewrite anything.
- Do NOT remove or modify any input tasks.
- The reasoning should briefly summarize what aspects contribute most to or detract from diversity.
```

Table 12: Prompt for evaluating diversity of high-level tasks in a single web environment.