WebSynthesis: World Model-Guided Monte Carlo Tree Search for Efficient WebAgent Trajectory Synthesis

Anonymous ACL submission

Abstract

Recent advances in large language models (LLMs) have greatly empowered web agents, yet navigating complex and dynamic environments still demands stronger trajectory-level learning. Existing self-improving approaches rely on real-world GUI trajectories but suffer from environment instability and high data collection costs. To overcome these issues, we introduce WebSynthesis, a framework for scalable trajectory synthesis and training. By leveraging a learned world model for virtual web simulation, WebSynthesis enables efficient, reversible tree-based planning to generate diverse, high-quality trajectories that iteratively refine the agent's policy. Experiments demonstrate that agents trained on WebSynthesis achieve performance comparable to or surpassing those trained on large-scale real-world data.

1 Introduction

Automated agents powered by large language models (LLMs) have demonstrated remarkable potential for web navigation tasks (Ferrag et al., 2025; Zhang et al., 2024a,c; Yang et al., 2025; Sun et al., 2024; Fang et al., 2025; Lai et al., 2024; Wei et al., 2025). Web agents, such as Operator (OpenAI) and Manus (Manus AI), are designed to interact with real-world web environments through a multi-turn perception (reading) and action (writing) loop to successfully complete user-defined tasks. Therefore, training these agents on high-quality trajectories is crucial to improve their agentic capabilities (Zheng et al., 2024; Sun et al., 2024; Xu et al., 2024). However, collecting large-scale, multi-step trajectories for web navigation remains challenging. The dynamic and heterogeneous nature of web interfaces makes manual demonstration collection labor-intensive and unsustainable (Gu et al., 2024a; Chae et al., 2024; Xu et al., 2024; Qin et al., 2025). Furthermore, training agents directly through live web interactions is often unpredictable and costly due to environmental noise and the frequent API requests, rendering reinforcement learning (RL) methods prohibitively expensive (Qi et al., 2024; Lai et al., 2024; Wei et al., 2025).

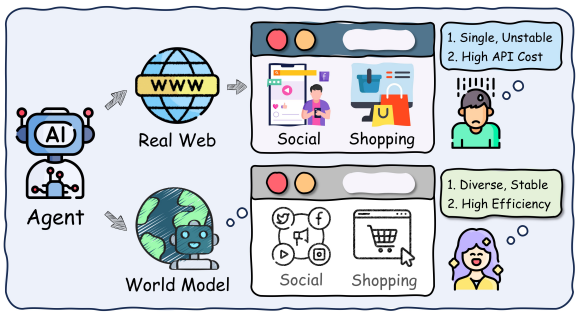


Figure 1: Comparison between real web and world model environments for agent exploration. While real web interactions are costly and unstable, the world model provides a diverse, stable, and efficient simulation space for trajectory generation and agent training.

Recent studies have explored trajectory-level self-evolving approaches, where agents iteratively improve by learning from autonomously generated trajectories (Sun et al., 2024; Zhou et al., 2024; Xu et al., 2024; Fang et al., 2025). This paradigm reduces reliance on human annotations and may uncover novel interaction patterns beyond human demonstrations (Fang et al., 2025). Despite their effectiveness, existing methods face two key challenges: (1) accurately modeling the complexity of modern web environments remains difficult (Zhou et al., 2023; Gandhi and Neubig, 2025), causing failures and unrealistic behaviors in imperfect simulators; and (2) trajectory synthesis is often computationally expensive (Putta et al., 2024), with some pipelines requiring hundreds of real-world interactions.

To address these limitations, we present **WebSynthesis**, a novel framework that integrates world modeling (Chae et al., 2024; Gu et al., 2024a; Fang et al., 2025) with search-based trajectory genera-

tion (Koh et al., 2024b; Zhang et al., 2024c). As shown in Figure 1, the world model serves as a high-fidelity proxy for the web, enabling diverse and low-cost trajectory generation. To enhance task relevance, we employ Monte Carlo Tree Search (MCTS) (Świechowski et al., 2022), guiding exploration toward specific goals. By simulating multiple action paths and retaining coherent, successful outcomes, WebSynthesis produces trajectories that are both goal-directed and diverse.

We adopt a two-stage curriculum to train the policy agent. In the first stage, the agent develops fundamental UI understanding for rapid adaptation to complex web interfaces. In the second stage, the agent undergoes trajectory learning via Supervised Fine-Tuning (SFT) on synthetic data. This curriculum allows agents to safely practice diverse tasks in virtual environments, improving generalization and accelerating self-improvement beyond the data limitations of prior methods. Experimental results demonstrate that WebSynthesis achieves strong performance with limited synthetic data. With only 4,000 synthetic trajectories, our approach attains a 20.15% overall Pass@3 success rate, surpassing OS-Genesis (Sun et al., 2024) (18.66% with 7,400 real-world trajectories) and AgentTrek (Xu et al., 2024) (11.94% with 20,000 tutorial-guided synthetic trajectories).

Our main contributions are: (1) proposing WebSynthesis, which combines an LLM-based world model with MCTS to synthesize diverse, controllable web trajectories entirely offline; (2) designing a two-stage curriculum to systematically enhance UI understanding and navigation capabilities; and (3) demonstrating that 4,000 synthetic trajectories are sufficient to outperform methods trained on much larger real-world datasets.

2 Related Works

Data synthesis approaches address data scarcity by synthesizing rich training trajectories. However, these methods face several challenges: (1) Self-directed exploration often yields low-diversity trajectories, leading to learning stagnation as agents repeatedly encounter similar patterns (Zhou et al., 2024; Gandhi and Neubig, 2025; Xie et al., 2025). (2) Rule-based or tutorial-guided strategies typically cover only predefined task templates, leaving numerous edge cases underexplored (Xu et al., 2024). (3) Accurately modeling the full complexity of web environments remains inherently challenging (Zhou et al., 2023; Koh et al., 2024a), resulting in limited scenario coverage or unrealistic behaviors. (4) Existing synthesis efforts primarily focuses on generating successful completion trajectories but largely overlooks the critical aspect of error recovery (Wu et al., 2025), leading to agents that lack robustness when encountering mistakes in real-world settings (Hu et al., 2025).

World models have been incorporated into the "simulate-before-act" paradigm (Gu et al., 2024b) for web navigation. Representative methods such as WebDreamer (Gu et al., 2024a) and WMA (Chae et al., 2024) employ LLMs as world models to predict the outcomes of candidate actions in advance, thereby preventing irreversible errors and reducing real-time interaction costs. Nevertheless, their effectiveness relies heavily on the fidelity of the learned world model, which constrains their reliability in online planning. To address this, WebEvolver (Fang et al., 2025) extends the paradigm by co-evolving the agent and the world model, enabling the agent to refine its decision-making within an increasingly accurate simulated environment. Our work integrates LLM-based world modeling with goal-directed search. In particular, the world model is trained to generate detailed next-state observations, which empowers the policy agent to conduct efficient and diverse trajectory exploration through tree-based planning in a virtual environment.

3 Methodology

In this section, we present the details of the WebSynthesis pipeline, which includes both the data collection and the curriculum learning process.

3.1 Problem Formulations

Following the standard reinforcement learning (RL) framework (Ding et al., 2024; Cao et al., 2025; Moerland et al., 2023), we model web navigation as a partially observable Markov decision process (POMDP), which consists of four main components: the observation space ($\mathcal{O}$), action space ($\mathcal{A}$), transition function ($\mathcal{T}$), and reward function ($\mathcal{R}$). In our approach, we define the policy agent, world model, and process reward model as follows:

Policy Agent: At each time step t, the policy agent π selects an action a_t based on the current observation o_{t-1} and the user's instruction q.

World Model: The web world model ω acts as the transition function $\mathcal{T}$, mapping a state-action pair

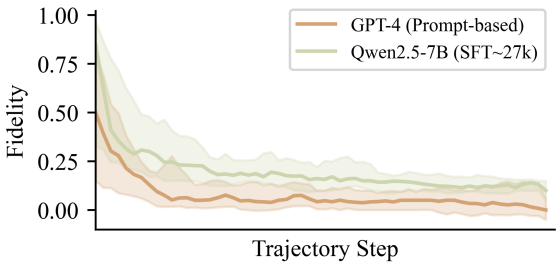


Figure 2: The snowballing effect of the world model prediction errors.

(o_{t-1}, a_t) to the next observation o_t.

Process Reward Model: The reward model γ evaluates the actions taken by the policy agent based on the current observation o_t, the previous trajectory $\{o_{i-1}, a_i\}_{i=1}^{t-1}$, and the user's instruction q.

The objective of web navigation is formulated as:

$$\arg\max_{a_{0:T}} \mathbb{E}\left[\sum_{t=0}^{T} \gamma\left(o_t, a_t; \mathsf{q}, o_{0:t-1}, a_{0:t-1}\right)\right] \quad (1)$$

where $o_t \sim \omega(\cdot|o_{t-1}, a_t)$, $a_t \sim \pi(\cdot|o_{t-1}, \mathsf{q})$, indicating that the policy agent π processes the user query q and engages in multi-step interactions within the world model environment. The term $\arg\max_{a_{0:T}}$ refers to the search algorithm. Within a fixed time budget T or step budget, the objective of the search is to identify a trajectory that maximizes the reward. Consistent with prior work, we represent web page observations using the accessibility tree (A11y), which captures a structured set of accessibility-related states and properties.

3.2 Analysis of World Model Prediction Errors

While world models like WebDreamer (Gu et al., 2024a) and WMA (Chae et al., 2024) can reduce costly real-time interaction, their reliability in online trajectory planning is limited by simulation drift from compounding errors. To more intuitively showcase the prediction errors, we consider two types of world models: (1) a CoT prompt-based model (GPT-4) and (2) a model fine-tuned on state transition data (Qwen2.5-7B; see Appendix C). As shown in Figure 2, replaying a ground-truth trajectory (human-annotated data from WebArena (Zhou et al., 2023)) within the world model reveals an exponential decay in next-state fidelity. This demonstrates that minor transition errors, recursively amplified at each step, cause predicted states to drift rapidly from reality, weakening the feasibility of multi-step planning. We reflect this phenomena to the snowballing effect. Furthermore, we also observe that the fine-tuned model slightly outperforms GPT-4 in multi-step rollouts, yet remains susceptible to the snowballing effect.

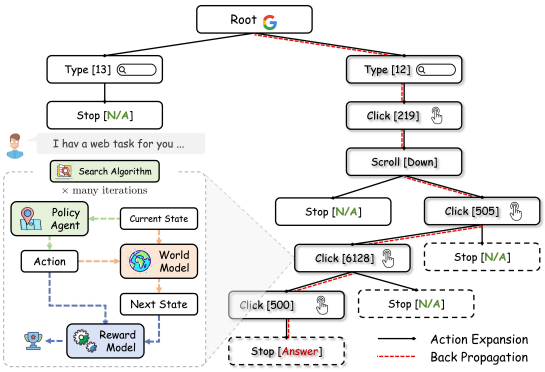


Figure 3: Pipeline of the world model-guided Monte Carlo Tree Search (WebMCTS). The solid lines represent action expansion, while the red dashed lines indicate backpropagation of high-value trajectories. The process involves iterative simulations using the policy agent, world model, and reward model to evaluate and expand potential action paths.

3.3 World Model-guided MCTS

Previous analysis suggests that linear rollouts of the world model quickly accumulate prediction errors. To address this, we adopt the Monte Carlo Tree Search (MCTS), which explores multiple candidate trajectories and alleviates the snowballing effect. An overview of the framework is provided in Figure 3. We formulate the world model-guided Monte Carlo Tree Search (**WebMCTS**) as an iterative algorithm comprising three key stages: node selection, action expansion, and backpropagation. At each iteration, the search tree is expanded based on the candidate actions proposed by the policy agent π. Each node in the tree represents a specific action a_t and stores the predicted next-state observation o_t from the world model, the estimated reward v_C from the reward model, and the visit count n_C.

Node Selection Following standard MCTS settings (Zhang et al., 2024b; Świechowski et al., 2022), node selection is guided by the Upper Confidence Bound (UCB) strategy, which balances exploration and exploitation throughout the search process. The UCB score for node C is defined as:

$$\mathrm{U}_C = v_C + \epsilon \cdot \sqrt{\frac{\ln n_P}{n_C}}$$

where n_P denotes the number of visits to the parent, and ϵ is a constant that controls the level of exploration. At each iteration, the algorithm selects the child node with the highest UCB score to continue the search.

Action Expansion Given a node C selected through the UCB strategy, candidate actions are sampled based on its associated state. To encourage exploration and maintain search tree diversity, WebMCTS samples at least three distinct actions during each expansion step. Since different branches of the search tree may reach the same web page at different stages during the search, we employ a caching mechanism to ensure consistency in the state predictions of the world model. Specifically, we maintain a hash table that maps each URL to its corresponding page state which was previously generated by the world model. When the same URL is encountered again, the cached state is reused, avoiding redundant state generation and improving both consistency and efficiency during the expansion.

Backpropagation Finally, value backpropagation is initiated from the selected node C, and the values of its ancestor nodes are updated using a weighted average approach. Specifically, its visit count n_C and estimated value v_C are updated as follows:

$$n_C \leftarrow n_C + 1, \text{ and } v_C \leftarrow \frac{\sum_i n_{C_i} \cdot v_{C_i}}{\sum_i n_{C_i}}$$

Through repeated iterations, this process constructs a web action tree within the world model, capturing a diverse set of possible interaction paths.

3.4 Error Filtering in Node Selection

In practical applications of model-based reinforcement learning (e.g., web navigation with POMDP dynamics), the world model inevitably suffers from prediction errors (e.g., undefined UI interactions). Such errors introduce stochasticity into the value estimates of MCTS nodes, raising a critical question: *Can MCTS reliably identify the optimal node in the presence of prediction noise?* To address this, we investigate how prediction errors propagate during node selection within the MCTS framework.

Theorem 1 *Consider two nodes C_1 and C_2 in a Monte Carlo Tree Search (MCTS) process, where the true values satisfy $v_1^* > v_2^*$ (i.e., $\Delta_v = v_1^* - v_2^* > 0$). The value predictions for both nodes are subject to random errors δ_1 and δ_2, which are independently distributed according to a normal distribution $\mathcal{N}(0, \sigma^2)$. Under these conditions, the probability of incorrectly selecting the suboptimal node C_2 decreases exponentially as the total number of explorations n_{total} increases. Specifically, the error probability follows:*

$$P(C_2 | v_1^* > v_2^*) \sim e^{-\kappa \cdot n_{total}}$$

where $\kappa > 0$ is a constant that depends on Δ_v and the model error variance σ.

The formal proof is provided in Appendix A. According to Theorem 1, the probability of selecting a suboptimal node decreases exponentially as the total number of explorations. This result implies that even with imperfect world model predictions, such as errors in modeling page transitions after a `click` action, the exponential decay in selection errors ensures that, as MCTS explores more paths (i.e., as n_{total} increases), the truly optimal trajectory (e.g., the shortest or most reliable action sequence) will eventually dominate the search. This confirms the effectiveness of our method in synthesizing high-quality trajectories within simulated web environments.

3.5 Curriculum Learning of the Policy Agent

The training of the policy agent consists of two main components: UI fundamental understanding and behavior cloning.

3.5.1 UI Fundamental Understanding

In headless browsing, the policy agent interacts with structured text data such as HTML, the DOM, and the accessibility tree. Yet the complexity of TextUI scenarios often hinders full contextual understanding, even with complete page data. To mitigate this, we collect and synthesize a set of TextUI datasets for the initial stage of supervised fine-tuning. Specifically, we adopt the approach of ProphetAgent (Kong et al., 2025), randomly exploring the online web environment to collect numerous transition triples (o_{t-1}, a_t, o_t), where action a_t maps observation o_{t-1} to o_t. Following UI-TARS (Qin et al., 2025), we derive three core TextUI capabilities: (1) dense captioning, (2) element functionality prediction, and (3) state transition captioning. Dataset construction details and instruction templates are given in Appendix B.

3.5.2 UI Behavior Cloning

Based on WebMCTS, the state–action tree includes not only the successful task-completing trajectory $\tau = o_t, a_{t=0}^N$ but also meaningful yet unsuccessful

attempts. We categorize these into two types: (1) valuable trajectories, which positively contribute to policy training, and (2) rollback trajectories, failed exploration paths that enhance robustness and error-awareness.

Valuable Trajectory: Since the initial policy agent may produce redundant or meaningless actions, we first prune the action tree before collecting training trajectories. We then apply depth-first search (DFS) to locate nodes whose values exceed a predefined threshold. For each identified target node, we trace the path backward to the root node and extract the corresponding trajectory for training. Appendix D details the collection process.

Rollback Trajectory: The core idea of rollback is to enable the policy agent to dynamically assess the effectiveness of its actions and their outcomes at each step along the trajectory. When an action deviates from the user's intent, the agent should recognize the negative consequence of the incorrect decision, revert to the previous valid state o_{t-1}, and resume the reasoning process from that point. In WebMCTS, the action tree preserves multiple parallel branches, enabling discovery of diverse rollback trajectories within a single search episode, unlike traditional linear rollback methods (Zhang et al., 2025; Qin et al., 2025), which typically yield only one trajectory per run. To extract rollback trajectories, we process each intermediate node C on the value trajectory. We identify the set of unsuccessful siblings S of C and their parent P, then use GPT-4 to generate a reflection for the worst node in S, with "`go_back`" as the corresponding next action. The historical action sequence at P is revised to incorporate this correction, forming a new path from P to C. This yields rollback paths of the form $S \rightarrow P \rightarrow C$, capturing alternative reasoning and corrective feedback from failed attempts.

4 Experiments

4.1 Experimental Settings

Model Settings: We use Qwen2.5-7B-Instruct as the base model for both the policy agent and world model training, and GPT-4 as the reward model for tree search. The collection of world model training data is carried out in an online setting (Zhou et al., 2023; Koh et al., 2024a; Gandhi and Neubig, 2025; Xu et al., 2024), with further details shown in Appendix C. All training is conducted using low-rank adaptation (LoRA) fine-tuning with a rank of 16.

Evaluation Benchmarks: For Web navigation tasks, WebArena (Zhou et al., 2023) serves as a dynamic benchmark to evaluate autonomous agents. It covers domains such as e-commerce (Shopping), content management (CMS), social forums (Reddit), and software development (GitLab). Policy agents are required to make sequential decisions starting from an initial state, and task success is evaluated through string matching or URL matching between the agent's final output and the ground-truth answer. We follow the standard WebArena setup and adopt the following action space: `click`, `type`, `hover`, `scroll`, `goto`, `go_back` and `stop`. The original WebArena environment consists of 812 challenging tasks derived from 241 task templates. Given the cost of evaluation, we use 165 test cases from WebArena-Lite (Liu et al., 2024).

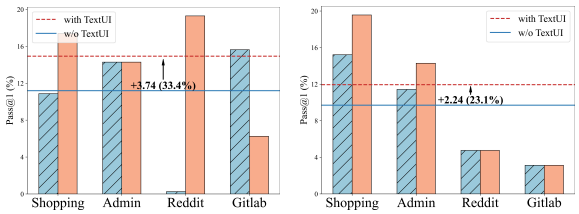
Baseline Settings: We compare our approach with the following baseline models: (1) Qwen2.5-7B-Instruct, LLaMA3.1-8B-Instruct, and GPT-4, all evaluated using chain-of-thought (CoT) prompting (Zhou et al., 2023); (2) OS-Genesis (Sun et al., 2024), which uses 7.4k real trajectories collected from WebArena; (3) AgentTrek (Xu et al., 2024), which generates web agent trajectories by leveraging publicly available tutorials and contains 57k trajectory records. To ensure a fair comparison of headless browsing capabilities, we use only the text-based state representation from the trajectories as input for all methods. For AgentTrek, we sample 20k trajectories from the original dataset to match the action space in WebArena. Both OS-Genesis and AgentTrek are fine-tuned on Qwen2.5-7B.

4.2 Main Results

WebSynthesis demonstrates clear advantages over chain-of-thought (CoT) prompting methods. As shown in Table 1, it outperforms both open-source models (e.g., Qwen2.5-7B-Instruct, LLaMA3.1-8B-Instruct) and closed-source GPT-4 across WebArena subsets, achieving a higher overall score (14.93% vs. 13.58% for GPT-4) and substantially outperforming Qwen2.5-7B-Instruct (2.24%). Compared to OS-Genesis-7B, which is trained on 7.4k real world collected trajectories, WebSynthesis attains +3.74% higher Pass@1 despite using only 4k samples. These results highlight that high-quality trajectories synthesized by WebMCTS can rival or even surpass real data, while requiring far fewer samples.

Model	Param Size	Shopping	Admin	Reddit	GitLab	Overall
Qwen2.5	7B	2.17	0.00	0.00	6.25	2.24
LLaMA3.1	8B	8.70	5.71	0.00	6.25	5.97
GPT-4	N/A	13.04	17.14	9.52	**18.75**	13.58
Pass@1						
AgentTrek (Xu et al., 2024)	7B	15.22	11.43	4.76	3.13	9.70
OS-Genesis (Sun et al., 2024)	7B	10.87	14.29	0.00	15.63	11.19
WebSynthesis	7B	19.57	17.14	9.52	9.38	14.93
Pass@3						
AgentTrek (Xu et al., 2024)	7B	19.57	11.43	9.52	3.13	11.94
OS-Genesis (Sun et al., 2024)	7B	19.57	**31.43**	0.00	15.63	18.66
WebSynthesis	7B	**28.26**	20.00	**14.29**	12.50	**20.15**

Table 1: Evaluation results (%) on different WebArena subsets. **Bold** numbers indicates the highest score in each column, while underlined values represent the second highest. We report the task success rate for both one-shot (*Pass@1*) and three-shot (*Pass@3*) settings. A task is considered successful if at least one of the sampled attempts meets the evaluation criteria.



(a) OS-Genesis (b) AgentTrek

Figure 4: Evaluation on WebArena with (refer to as the blue bars) and without (refer to as the orange bars) TextUI warm-up. **Bold** annotations highlight the performance improvements observed on OS-Genesis and AgentTrek, showing a +3.74% and +2.24% absolute gain in Pass@1, respectively.

Alg.	World	Reward	Depth	Width	Steps	*Pass@1*
MCTS	GPT-4	GPT-4	3.1	1.7	6.1	15.8
MCTS	Ours	Qwen	5.9	5.4	5.6	13.9
Beam	Ours	GPT-4	6.4	5.4	6.8	15.2
MCTS	Ours	GPT-4	5.5	4.6	**4.7**	**24.1**

Table 2: Impact of individual components in WebMCTS on the quality of the synthetic data. **Alg.** denotes the search algorithm. GPT-4 and Qwen correspond to prompt-based data generation methods, while "Ours" refers to the learned world model in this work. We report the average tree depth and width, the number of evaluation steps per task, and the final performance on the Shopping and Admin scenarios.

4.3 Analysis

TextUI Helps Trajectory Learning: A core component of WebSynthesis is the preliminary TextUI warm-up phase, which is conducted before formal trajectory-level learning. This phase equips the policy agent with fundamental UI understanding skills. To evaluate its effectiveness, we compare two trajectory learning methods, OS-Genesis and AgentTrek, both with and without TextUI warm-up. As shown in Figure 4, models with TextUI adaptation outperform those trained without this warm-up. Notably, OS-Genesis shows a 33.4% improvement over its original performance. This highlights that fine-tuning on real-world trajectories alone is insufficient. Without sufficient familiarity with the structure and layout of text-based UIs, trajectory-level training alone is unlikely to yield optimal performance.

WebMCTS Component Impact on Data Quality: As shown in Table 2, all methods exhibit improved web navigation performance following trajectory fine-tuning, relative to Qwen2.5-7B. This confirms the effectiveness of WebMCTS in generating meaningful web navigation trajectories.

Our analysis reveals that replacing any key component of WebMCTS with a weaker alternative degrades both trajectory quality and downstream performance. Specifically, replacing our learned world model with GPT-4's prompt-based simulation reduces the tree's structural complexity (Depth: 3.1 vs. 5.5; Width: 1.7 vs. 4.6), and leads to longer test-time action sequences (Steps: 6.1 vs. 4.7), ultimately reducing Pass@1 from 24.1% to 15.8%. This highlights the importance of a dedicated world model in supporting reliable state rollouts. Similarly, replacing the reward model with the weaker Qwen API significantly lowers performance (Pass@1 drops to 13.9%), even though the generated search tree remains reasonably wide and deep. This suggests that the reward model plays a decisive role in guiding the search process to-

FU.			BC.		Shopping	Admin	Reddit	GitLab	Overall
Cap.	Func.	Trans.	τ_{roll}	τ_{val}					
			✓		4.35	0.00	0.00	0.00	1.49
				✓	6.52	2.86	9.52	6.25	5.97
			✓	✓	13.04	11.43	0.00	9.38	9.70
✓			✓	✓	17.39	11.43	9.52	0.00	10.45
✓	✓		✓	✓	**21.74**	14.29	4.76	3.13	12.69
✓	✓	✓	✓	✓	19.57	**17.14**	**9.52**	**9.38**	**14.93**

Table 3: Ablation study on the contributions of *UI Fundamental Understanding* (FU.) and *UI Behavior Cloning* (BC.) tasks. "Cap." stands for dense captioning, "Func." represents element functionality, and "Trans." refers to state transition prediction. The trajectories τ_{val} and τ_{roll} denote the valuable and rollback trajectories, respectively. A checkmark ✓ indicates that the corresponding task type was included during training. The shaded row represents the full WebSynthesis configuration.

ward high-reward states, despite comparable structural capacity. Moreover, substituting the search algorithm with Beam Search results in the longest trajectories during evaluation (Steps: 6.8), due to its lack of an explicit node selection mechanism like UCB. Without strategic backtracking, Beam Search often explores unnecessarily long or repetitive paths and fails to terminate early upon reaching near-optimal states. In contrast, WebMCTS achieves both the shortest trajectories (4.7 steps) and the best Pass@1 score, indicating stronger action confidence and execution efficiency. These findings empirically support Theorem 1, and confirm that combining a robust world model, effective reward estimation, and principled exploration is critical to building a scalable synthetic trajectory generator. The experimental settings are shown in Appendix E.

4.4 Ablation Studies

On Fundamental Understanding: We conduct sequential ablation experiments by incrementally adding tasks based on their difficulty levels. As shown in Table 3, the three-stage curriculum learning framework gradually improves the fundamental understanding capabilities of the policy agent. Notably, the introduction of the state transition task in the third stage resulted in a 5.23% performance increase compared to the first two stages. This highlights the significance of modeling the world state transition process to improve policy learning. It also emphasizes the importance of guiding the model to understand the dynamics of UI changes early in the training process, thereby providing a solid foundation for subsequent multi-step reasoning and action execution. Additionally, from the task dimension perspective, although introducing dense captioning or functionality descriptions yields some improvement, the overall gains are relatively modest. In contrast, combining these tasks with the state transition task leads to a more substantial performance boost, further emphasizing the multi-dimensional nature of UI understanding.

On Behavior Cloning: Table 3 presents the ablation results comparing different trajectory supervision strategies. We examine four configurations: training only on valuable trajectories (τ_{val}), only rollback trajectories (τ_{roll}), their combination ($\tau_{\text{val}} \cup \tau_{\text{roll}}$), and the complete WebSynthesis pipeline, which incorporates TextUI warm-up prior to combined trajectory training.

Training solely on rollback trajectories (τ_{roll}) results in the weakest performance (1.49% overall), suggesting that without exposure to successful demonstrations, the agent becomes overly cautious and prematurely invokes `go_back` actions. Consequently, the agent loses its ability to explore and complete tasks effectively. This highlights the necessity of learning from target-directed behavior to encourage confident forward execution. In contrast, training only on valuable trajectories (τ_{val}) yields moderate performance (5.97% overall). While this approach helps the agent reach task goals, it fails to expose the agent to real-world error patterns or recovery strategies, thereby limiting its robustness in unforeseen scenarios. Combining both τ_{val} and τ_{roll} leads to a significant improvement (9.70% overall). This suggests that exposure to both successful trajectories and failure recovery paths enhances the agent's reasoning capability, enabling it to navigate uncertain situations more effectively. Finally, the full WebSynthesis method, which incorporates Tex-

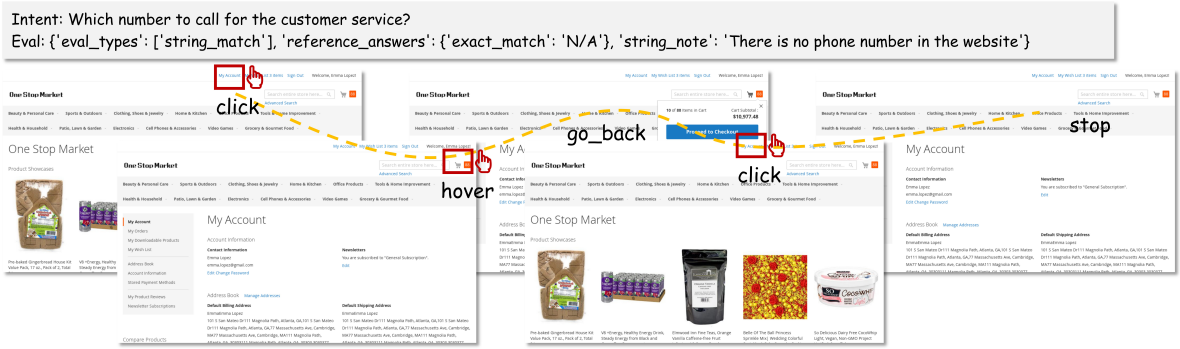


Figure 5: An example of rollback behavior in web navigation.

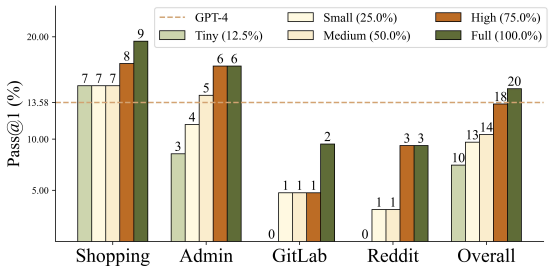


Figure 6: Performance improvement as the amount of synthetic data increases. The bars represent different data scales, ranging from 12.5% (Tiny) to 100% (Full). The dashed line indicates the performance of GPT-4 (13.58%) for reference.

tUI warm-up prior to trajectory-level fine-tuning, achieves the best performance (14.93%). This highlights the importance of UI fundamental understanding in familiarizing the model with text-based UIs, thus enabling more effective policy learning in subsequent training stages.

4.5 Data Scaling Analysis

To investigate the effect of synthetic data scaling on policy performance, we control the proportion of the WebSynthesis dataset used for training. As shown in Figure 6, our evaluation on WebArena reveals a steady performance improvement with the expansion of data scale. Notably, a 7.47% performance gain is observed when scaling from 12.5% (approximately 500 samples) to 100% of the dataset. Remarkably, at 75% of the dataset (around 3k samples), performance already matches that of GPT-4. This suggests that the high-information-density data synthesized by WebMCTS enables significant improvements in sample efficiency, even with fewer examples. These findings reveal that automated synthetic data generation is a viable strategy for narrowing the performance gap with real-world collected data, highlighting its tremendous potential for future scalability. We provide more qualitative analysis in Appendix F.

4.6 Analysis of Synthetic Trajectories

We provide some cases on the WebMCTS synthesized trajectories, as shown in Figures 11, 12, and 13. These results indicate that WebMCTS can synthesize high-consistent web-navigation data and a diverse set of trajectories, substantially reducing reliance on online interaction with the environment. We further observe that WebMCTS sometimes yields "compressed" trajectories, goal-equivalent action sequences that achieve the same outcome with fewer steps. Such trajectories act as beneficial shortcuts: training on them improves the agent's planning efficiency and, therefore, task pass rates.

4.7 Analysis of Policy Agent Behavior

Figure 5 visualize a rollback behavior in web navigation. When deviating from the intended path, the agent autonomously rolls back to a previous state and resumes reasoning, ultimately completing the task. This illustrates the effectiveness of rollback trajectories in improving navigation robustness.

5 Conclusion

In this work, we propose WebSynthesis, a framework that integrates world model learning with Monte Carlo Tree Search (MCTS) to mitigate the limitations of online trajectory synthesis for web agents. Through a two-stage curriculum, the policy agent acquires web navigation capabilities. Notably, the agent trained with WebSynthesis on a small-scale synthetic dataset achieves performance comparable to, or even surpassing that of models trained on large-scale real-world datasets.

6 Limitation

While WebSynthesis showcases the potential of leveraging world models to replace real web environments for offline data collection, a more ambitious and forward-looking direction lies in integrating world models into online reinforcement learning, following paradigms explored in real-world RL domains (Sun et al., 2025; Zhao et al., 2025; Kiran et al., 2021). In this setting, the policy agent could continuously refine its planning and decision-making capabilities by interacting with a simulated environment constructed by the world model. However, several key challenges must be addressed before this vision can be fully realized. First, unlike board games (Schrittwieser et al., 2020) or robotic control tasks (Kober et al., 2013), web interfaces are inherently diverse, partially observable, and non-stationary, making it difficult to learn a stable and generalizable world model. Second, compounding errors in multi-step rollouts (e.g., PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024)) may degrade performance, particularly when the agent relies heavily on imagined trajectories for exploration. Third, integrating model-based environments into online, closed-loop reinforcement learning settings remains an open research problem. It is crucial not only to prevent model collapse over time, but also to ensure that the agent's behavior aligns with human-aligned criteria. Achieving this goal requires advances in both algorithmic frameworks and the fidelity of learned world models.

7 Ethics Considerations

This work aims to advance the development of safe and efficient web agents through world model-guided trajectory synthesis. We acknowledge potential ethical considerations:

Misuse Risks: While our artifacts are intended for research, inappropriate deployment in real-world systems could propagate erroneous actions, unsafe behaviors, or biases. To mitigate this, we license all released artifacts strictly for research purposes and explicitly discourage commercial or production use.

Data Collection: The datasets we use (e.g., WebArena, OS-Genesis, AgentTrek) are public and licensed for research. All collected or synthesized data is anonymized and does not include personally identifiable information.

Bias and Fairness: Benchmarks are drawn from a limited set of web domains and may not fully capture diverse user experiences, particularly those of marginalized or vulnerable populations. We caution against over-generalizing from the reported results.

Environmental Impact: Training large models has non-negligible computational and environmental costs. To mitigate this, we design our framework to achieve strong performance with fewer trajectories and smaller-scale training.

Overall, our contributions are intended to facilitate safe academic exploration of agent learning methods. We encourage future work to incorporate stronger fairness evaluation, robustness guarantees, and safety alignment before considering deployment beyond research settings.

References

Yuji Cao, Huan Zhao, Yuheng Cheng, Ting Shu, Yue Chen, Guolong Liu, Gaoqi Liang, Junhua Zhao, Jinyue Yan, and Yun Li. 2025. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *IEEE Transactions on Neural Networks and Learning Systems*, 36(6):9737–9757.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2024. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*.

Jingtao Ding, Yunke Zhang, Yu Shang, Yuheng Zhang, Zefang Zong, Jie Feng, Yuan Yuan, Hongyuan Su, Nian Li, Nicholas Sukiennik, Fengli Xu, and Yong Li. 2024. Understanding world or predicting future? a comprehensive survey of world models. *Preprint*, arXiv:2411.14499.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. 2025. Webevolver: Enhancing web agent self-improvement with coevolving world model. *arXiv preprint arXiv:2504.21024*.

Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*.

Apurva Gandhi and Graham Neubig. 2025. Go-browse: Training web agents with structured exploration. *Preprint*, arXiv:2506.03533.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, and 1 others. 2024a. Is your

llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*.

Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2024b. Simulate before act: Model-based planning for web agents.

Minda Hu, Tianqing Fang, Jianshu Zhang, Junyu Ma, Zhisong Zhang, Jingyan Zhou, Hongming Zhang, Haitao Mi, Dong Yu, and Irwin King. 2025. Webcot: Enhancing web agent reasoning by reconstructing chain-of-thought in reflection, branching, and rollback. *Preprint*, arXiv:2505.20013.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. 2021. Deep reinforcement learning for autonomous driving: A survey. *Preprint*, arXiv:2002.00444.

Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024a. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024b. Tree search for language model agents. *Preprint*, arXiv:2407.01476.

Qichao Kong, Zhengwei Lv, Yiheng Xiong, Dingchun Wang, Jingling Sun, Ting Su, Letao Li, Xu Yang, and Gang Huo. 2025. *ProphetAgent: Automatically Synthesizing GUI Tests from Test Cases in Natural Language for Mobile Apps*, page 174–179. Association for Computing Machinery, New York, NY, USA.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and 1 others. 2024. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5295–5306.

Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, Jiadai Sun, Xinyue Yang, Yu Yang, Zehan Qi, Shuntian Yao, Xueqiao Sun, Siyi Cheng, Qinkai Zheng, Hao Yu, and 11 others. 2024. Visualagentbench: Towards large multimodal models as visual foundation agents. *Preprint*, arXiv:2408.06327.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. 2023. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, and 1 others. 2024. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*.

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

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, and 1 others. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. 2025. Zerosearch: Incentivize the search capability of llms without searching. *Preprint*, arXiv:2505.04588.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, and 1 others. 2024. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*.

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

Zheng Wu, Heyuan Huang, Xingyu Lou, Xiangmou Qu, Pengzhou Cheng, Zongru Wu, Weiwen Liu, Weinan Zhang, Jun Wang, Zhaoxiang Wang, and Zhuosheng Zhang. 2025. Verios: Query-driven proactive human-agent-gui interaction for trustworthy os agents. *Preprint*, arXiv:2509.07553.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2025. GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5650–5667, Vienna, Austria. Association for Computational Linguistics.

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*.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. 2025. Agentoccam: A simple yet strong baseline for llm-based web agents. *Preprint*, arXiv:2410.13825.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, and 1 others. 2024a. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024b. Rest-mcts*: Llm self-training via process reward guided tree search. *Preprint*, arXiv:2406.03816.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2024c. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. *Preprint*, arXiv:2408.15978.

Zhisong Zhang, Tianqing Fang, Kaixin Ma, Wenhao Yu, Hongming Zhang, Haitao Mi, and Dong Yu. 2025. Enhancing web agents with explicit rollback mechanisms. *Preprint*, arXiv:2504.11788.

Baining Zhao, Ziyou Wang, Jianjie Fang, Chen Gao, Fanhang Man, Jinqiang Cui, Xin Wang, Xinlei Chen, Yong Li, and Wenwu Zhu. 2025. Embodied-r: Collaborative framework for activating embodied spatial reasoning in foundation models via reinforcement learning. *arXiv preprint arXiv:2504.12680*.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. *Preprint*, arXiv:2306.07863.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. 2024. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. *arXiv preprint arXiv:2412.13194*.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2022. Monte carlo tree search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562.

A Error Filtering in Node Selection

Restatement of Theorem 1. *Consider two nodes C_1 and C_2 in a Monte Carlo Tree Search (MCTS) process, where the true values satisfy $v_1^* > v_2^*$ (i.e., $\Delta_v = v_1^* - v_2^* > 0$). The value predictions for both nodes are subject to random errors δ_1 and δ_2, which are independently distributed according to a normal distribution $\mathcal{N}(0, \sigma^2)$. Under these conditions, the probability of incorrectly selecting the suboptimal node C_2 decreases exponentially as the total number of explorations n_{total} increases. Specifically, the error probability follows:*

$$P(C_2 | v_1^* > v_2^*) \sim e^{-\kappa \cdot n_{total}}$$

where $\kappa > 0$ is a constant that depends on Δ_v and the model error variance σ.

Proof A.1 *According to the definition of Upper Confidence Bound (UCB) strategy, the UCB score of each candidate node is given by:*

$$U_{C_1} = v_1^* + \delta_1 + \epsilon \cdot \sqrt{\frac{\ln n_P}{n_{C_1}}},$$

$$U_{C_2} = v_2^* + \delta_2 + \epsilon \cdot \sqrt{\frac{\ln n_P}{n_{C_2}}}$$

where ϵ is a constant controlling the exploration term, n_P is the total number of explorations, and n_{C_1} and n_{C_2} denote the number of visits to nodes C_1 and C_2, respectively.

The event of incorrectly selecting C_2 (denoted as event E) occurs when $U_{C_1} < U_{C_2}$, then we get the following inequality:

$$v_1^* + \delta_1 + \epsilon \cdot \sqrt{\frac{\ln n_P}{n_{C_1}}} < v_2^* + \delta_2 + \epsilon \cdot \sqrt{\frac{\ln n_P}{n_{C_2}}}$$

Rearranging this inequality, we obtain the following condition for the occurrence of error:

$$\delta_1 - \delta_2 < -\Delta_v + \epsilon \sqrt{\ln n_P} \cdot \left(\frac{1}{\sqrt{n_{C_2}}} - \frac{1}{\sqrt{n_{C_1}}}\right)$$

Next, consider the difference between the errors $D = \delta_1 - \delta_2$. Since δ_1 and δ_2 are independent and normally distributed, the difference D follows a

normal distribution with mean 0 *and variance* $2\sigma^2$*, i.e.* $D \sim \mathcal{N}(0, 2\sigma^2)$*. Then the probability density function of* D *is:*

$$f_D(x) = \frac{1}{4\pi\sigma^2} e^{-\frac{x^2}{4\sigma^2}}$$

According to the asymptotic optimality of MCTS, the number of visits to the node with higher true value, C_1*, will grow asymptotically faster than that of the suboptimal node* C_2*. This implies that the term* $\frac{1}{\sqrt{n_{C_1}}}$ *will grow slower than* $\frac{1}{\sqrt{n_{C_2}}}$*, thus we obtain:*

$$\delta_1 - \delta_2 < -\Delta_v + \epsilon\sqrt{\ln n_P} \cdot \frac{1}{\sqrt{n_{C_2}}}$$

As the total number of explorations n_{total} *increases, the number of visits to node* C_2 *also increases, which lead to* $\frac{1}{\sqrt{n_{C_2}}}$ *tending towards zero. Thus, the error condition becomes approximately:*

$$P(E) \approx P(D < -\Delta_v) = \Phi(-\frac{\Delta_v}{\sqrt{2}\sigma})$$

where $\Phi(\cdot)$ *denotes the cumulative distribution function of the standard normal distribution.*

Using the tail properties of the normal distribution, we can estimate the error probability $P(E)$ *as follows. For large values of* x*, we know that:*

$$\Phi(-x) \leq \frac{1}{x\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

Substituting $x = \frac{\Delta_v}{\sqrt{2}\sigma}$*, we obtain:*

$$P(E) \leq \frac{\sigma}{\Delta_v\sqrt{\pi}} e^{\frac{-\Delta_v^2}{4\sigma^2}}$$

This shows that the probability of error decreases exponentially with the value of Δ_v*.*

Finally, as n_{total} *increases, the number of visits to* n_{C_2} *increases, leading to an exponential decay of the error probability. Thus, the probability of error decays exponentially with* n_{total}*, and we have:*

$$P(C_2 | v_1^* > v_2^*) \sim e^{-\kappa \cdot n_{total}}$$

where $\kappa > 0$ *is a constant that depends on* Δ_v *and the model error variance* σ*. Then the theorem has been proved.*

B TextUI Datasets

As described in the main text, we follow the task definitions in UI-TARS (Qin et al., 2025) and construct three core capabilities for understanding TextUI: (1) dense captioning, (2) element functionality description, and (3) state transition prediction, as shown in Figure 7.

B.1 Dense Captioning

To improve the model's global understanding of TextUI inputs, we address a key limitation of purely text-based representations: the lack of visual layout information. Summarizing textual content alone is often insufficient to capture the structure and context of complex user interfaces. To address this limitation, we leverage GPT-4o by providing it with corresponding GUI screenshots, enabling it to generate comprehensive descriptions that capture not only individual UI elements but also their spatial relationships and overall layout. During training, these visual-grounded descriptions are paired with text-based observations and used as input to the model. The corresponding instruction template is provided in Table 9.

B.2 Element Functionality Description

To enhance the recognition and comprehension of specific elements in TextUIs (e.g., `textbars` or `clickable elements`), we generate detailed, structured descriptions for each interactive component. Leveraging GPT-4o's advanced visual understanding capabilities, we prompt it to synthesize functional descriptions using both the text-based representations and the corresponding GUI screenshot. These synthesized descriptions are then paired with the text observations (i.e., A11y), replacing the screenshot as contextual input to form QA pairs. To reduce redundancy caused by the excessive length of A11y inputs, we apply a local compression strategy during preprocessing. Specifically, we retain only the elements adjacent to the target UI component while preserving the original hierarchical structure of the A11y tree. This approach eliminates irrelevant context and significantly improves both the diversity and efficiency of the resulting dataset. The corresponding instruction is shown in Table 10.

B.3 State Transition Prediction

To further enhance the model's understanding of UI elements, particularly in response to user interaction requests, we equip it with the ability to predict changes in the main page content. To achieve this, we extract the differences between consecutive observations and incorporate them into the agent's reasoning process. During training, the agent receives the current page observation and the corresponding action and is tasked with generating layout changes along with a caption for the subse-

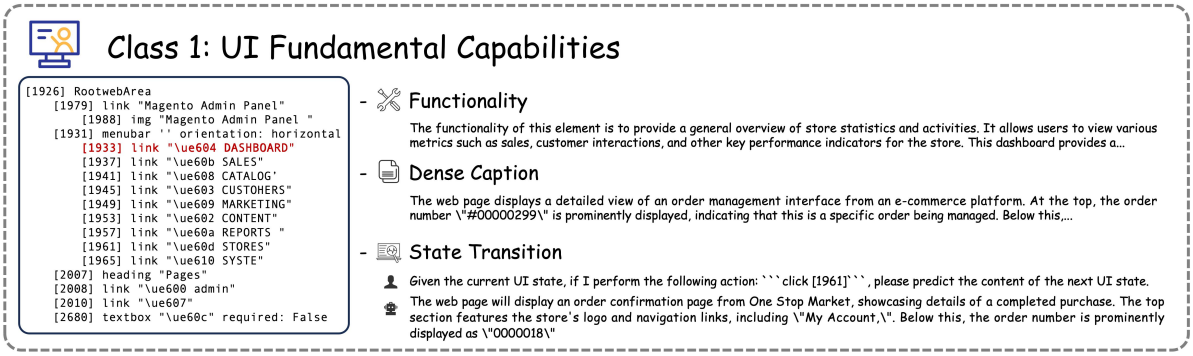


Figure 7: Overview of the UI Fundamental Understanding datasets. It provides single-step, fine-grained annotations, including dense captions, element functionality descriptions, and state transitions predictions, designed to train the model to understand, describe, and predict web page states.

quent UI state. This capability is crucial for tasks that require fine-grained interaction understanding and dynamic state awareness.

B.4 Post-processing of Collected Data

To improve data quality, we apply an instruction enhancement and response refinement process, as shown in Table 8. For different types of data, we prompt GPT-4 to generate various instruction templates to enhance the logical coherence between prompts and responses. For responses with complex or inconsistent formats, we use GPT-4 to reformulate them while preserving their semantic content. Please refer to Table 11 for details.

C Data Collection for the World Model

By definition, the world model is expected to accurately predict the next-state observation given the current state and the action taken by the policy agent. Consequently, the training data for the world model typically consist of state transition triples (o_{t-1}, a_t, o_t). Such data can be collected using two main approaches: (1) random exploration, where the agent interacts with the environment by randomly clicking on interactive elements to generate state transitions; (2) task-oriented exploration, where the agent collects transitions by following user-defined instructions.

Since the world model is designed to support the exploration behavior of the policy agent in WebMCTS, it is necessary to collect transition data tailored for depth-first search rather than remaining at the breadth-first level of the initial interface. This is because a typical web page contains hundreds of interactive elements, making exhaustive exploration computationally prohibitive. To address this issue, we adopt a task-oriented exploration strategy.

Algorithm 1 Data Cleaning Algorithm Based on Action Classification

Load raw data from input file
Categorize data by action type
  - Create dictionary *indices*: action → list of associated states
  - Store entries with only one associated state in *container*
  - Retain entries with multiple states for further processing
Group remaining data by action
  - Create dictionary *action_set_list*: action → list of metadata
for each action in *action_set_list* **do**
  while *action_set* is not empty **do**
    Randomly select *current_state* from *action_set*
    Construct *value_state_list* from remaining entries in *action_set*
    for each *value_state* in *value_state_list* **do**
      Compute similarity between *current_state* and *value_state* using *judgesim*
    end for
    Identify similar states with similarity score exceeding threshold
    if no similar states found **then**
      Remove *current_state* from *action_set*
    else
      Remove *current_state* and all similar states from *action_set*
    end if
    Add *current_state* and its corresponding *next_state* to *container*
  end while
end for
Output cleaned dataset stored in *container*

Specifically, we incorporate open source trajectory data from WebArena (Zhou et al., 2023), VisualWebArena (Koh et al., 2024a), Go-Browser (Gandhi and Neubig, 2025), OS-Genesis (Sun et al., 2024), and AgentTrek (Xu et al., 2024), and split the trajectories into state transition triplets.

Another major challenge is eliminating redundancy and achieving balanced datasets. Among the

80k collected transition triples, we observe several issues: (1) incorrect actions leading to unchanged states, (2) a long-tail distribution of action types, and (3) duplicated state transition pairs. To address the first two issues, we apply pattern-based filtering to eliminate invalid transitions and perform data resampling to mitigate class imbalance caused by infrequent action types. To further reduce redundancy, we propose a data cleaning algorithm based on action classification, as described in Algorithm 1. The algorithm works by grouping transitions by action type and performing iterative similarity comparisons between states within the same group. Only semantically distinct transitions are retained, ensuring that the cleaned dataset captures diverse user interaction patterns.

D Data Collection for the Policy Agent

In this section, we demonstrate the process of extracting complete, trainable trajectories from WebMCTS, including both valuable and rollback trajectories. As discussed in the main text, the initial policy agent may generate redundant or semantically meaningless actions during interactions with the world model. To address this, we first prune the action tree before collecting training trajectories. Specifically, we use GPT-4 to detect nodes with redundant or overly similar actions (e.g., `type [1201] [bus stop near CMU] [1]` and `type [1201] [bus stop near Carnegie Mellon University] [1]`), as these actions may yield identical state predictions from the world model. In these cases, we remove the redundant nodes and merge their child nodes into the respective parent nodes.

In collecting valuable trajectories, we use the node value v_C as the selection criterion, where higher values indicate a greater degree of task progress in the current state. During the downward traversal of the action tree, we maintain an operation history for each value node, consisting of (1) observations from ancestor nodes, (2) the reasoning behind the action, and (3) the action itself. Given the limited context window, we compress historical observations by leveraging the policy agent's action outcomes. Specifically, we encode historical observations as a tree structure and apply a sampling strategy that selects the ancestor, sibling, and child nodes of the current action node. This strategy preserves the hierarchical structure of the accessibility tree while minimizing context consumption. Please refer to Table 17 for details. In our implementation, the historical trajectory is limited to the three most recent operations. Each historical step is separated using $<$ step-i $>$ and $<$ /step-i $>$ tags.

For rollback trajectories, we prompt GPT-4 to generate reflections that trigger the `go_back` action. As shown in Table 18, the reflection is combined with the `go_back` action to form the final rollback behavior.

E Experimental Settings

The action space of WebArena: The action space is listed in Table 5. Following AgentOccam (Yang et al., 2025), we eliminate actions that heavily depend on embodiment knowledge, such as `press`, `new_tab`, `tab_focus`, `tab_close`, and `go_forward`.

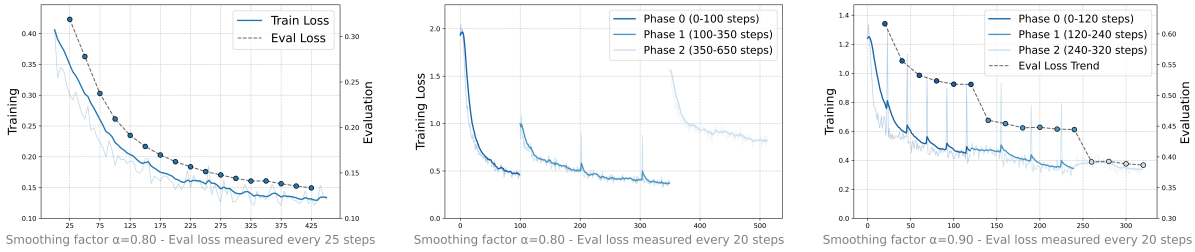
Setup of the world model training: The world model is trained on 4 H20 GPUs using low-rank adaptation (LoRA) fine-tuning applied to the Qwen2.5-7B-Instruct model. The training process consists of 400 steps, with a learning rate of `5e-5`, `warmup_steps` set to 50, a `weight_decay` of `0.01`, and a context length of 8192 tokens.

Setup of WebMCTS: We use the 241 task templates provided by WebArena to generate instructions for running tree searches. In our setup, each component of WebMCTS, including the policy agent and the reward model, is instantiated using GPT-4, while the world model is trained independently. The maximum search time is set to 800 seconds per task. If the time limit is exceeded and the policy agent has not issued a `stop` signal, or if the node value does not surpass the predefined threshold, we resort to a greedy search to determine the output trajectory. Since the policy agent tends to issue a `stop[N/A]` signal during the search, which may limit the size of the search tree. We consider terminating the expansion of the current node when the `stop[N/A]` signal is issued, but do not stop the search process entirely.

Setup of the TextUI warmup and policy agent training: Detailed training configurations are provided in Table 4. In the TextUI warm-up phase, we use a three-stage training strategy to inject fundamental UI understanding across different dimensions. In the subsequent phase, historical data is incorporated to prevent catastrophic forgetting of the model. During the behavior cloning phase, we also employ a three-stage learning strategy, allowing the model to gradually enhance its web naviga-

Class	Phase	Ratio	Steps	LR	Core Target
C1	P0	$[1, 0, 0]$	100	2.0e-4	Grasp the overall semantics of TextUI expressions.
	P1	$[0.4, 0.6, 0]$	250	1.0e-4	Refine its fine-grained understanding of specific UI functions.
	P2	$[0.1, 0.3, 0.6]$	275	1.0e-4	Strengthen its ability to predict page-state transitions.
C2	P0	$[0.7, 0.3]$	120	1.0e-4	Learn to rollback and improve basic navigation capabilities.
	P1	$[0.8, 0.2]$	120	8.0e-5	Increase the weight of basic navigation and reduce the frequency of go_back.
	P2	$[0.9, 0.1]$	80	6.0e-5	Continue to weaken rollback and practice the "one-shot" strategy.

Table 4: Training configuration for (C1) TextUI warm-up and (C2) policy agent behavior cloning. "Ratio" in C1 refers to the proportion of caption:function:transition data; in C2 it refers to valuable:rollback trajectory ratio.



(a) World Model (b) TextUI Warmup (c) Policy Agent

Figure 8: Training and evaluation loss curves for (a) world model, (b) TextUI warm-up, and (c) policy agent. Smoothing factors and evaluation intervals are shown in each subfigure.

Action Type a	Description
`click [elem]`	Click on elem.
`hover [elem]`	Hover on elem.
`type [elem] [text] [0|1]`	Type text on elem.
`goto [url]`	Open a url.
`go_back`	Click back.
`scroll [up|down]`	Scroll up or down.
`stop [answer]`	End with an output.

Table 5: The action space used in our settings.

tion and rollback capabilities. To prevent the policy agent from becoming overly cautious, we adjust the ratio of valuable and rollback trajectories in the training data, enhancing the agent's ability to reach the target in a single attempt.

Prompt and Instruction Template: The instruction format for TextUI tasks is provided in Table 8. Instruction templates for dense captioning, element functionality prediction, and state transition captioning are shown in Table 9, Table 10, and Table 11, respectively. The reflection prompt used in rollback trajectory construction is presented in Table 18. Prompts for the policy agent, world model, and reward model are available in Table 12, Table 13, and Table 14. The policy agent instruction templates for WebSynthesis, OS-Genesis, and AgentTrek are shown in Table 17, Table 15, and Table 16, respectively. For Chain-of-Thought (CoT) prompting, we follow the template design provided by WebArena. Implementation details are available on GitHub.

Setup of TextUI Analysis: The original OS-Genesis and AgentTrek models were trained using the Qwen2.5-7B-Instruct backbone. In our implementation, we first load the LoRA weights from the Class One (TextUI Warm-up) stage and subsequently fine-tune the models using the respective trajectory datasets from OS-Genesis and AgentTrek.

Setup of WebMCTS Analysis: In the WebMCTS framework, the original policy agent and reward model are instantiated using GPT-4, while the world model is trained independently. To analyze the contribution of each component to the quality of the synthetic search tree, we adopt a controlled variable strategy. Specifically, to evaluate the effect of the world model, we replace the trained model with a prompt-based GPT-4 implementation, while keeping the search strategy and reward model fixed. Similarly, to isolate the effects of the reward model and search algorithm, we replace the reward model with Qwen-Plus (the API model released by Qwen) and substitute the search strategy with Beam Search. Experiments are conducted on the shopping and admin subsets of WebArena, using identical task instructions to guide WebMCTS in

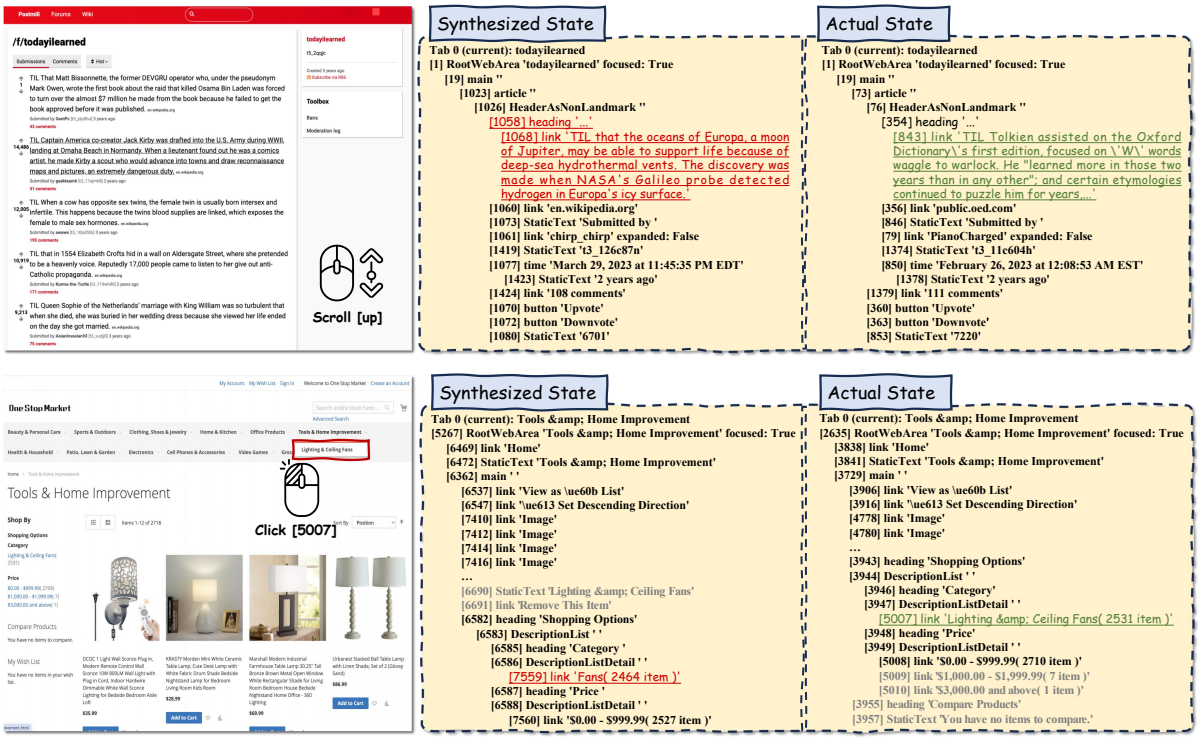


Figure 9: Case studies of world model-synthesized web states: (Top) Reddit scenario, with action: `scroll`; (Bottom) OneShopMarket scenario, with action: `click`. The synthesized states are compared to the actual states observed during interactions, demonstrating how the world model approximates the real web environment. **Black** indicates the same structural contents between the synthesized and actual states.

trajectory synthesis. For all configurations, only valuable trajectories are used for training. Training is conducted for 3 epochs, with a learning rate of `1e-5`, a `warmup_ratio` of `0.05`, a `weight_decay` of `0.01`, and a context length of 8192 tokens.

F Additional Experimental Analysis

F.1 World Model Evaluation

In the WebMCTS process, evaluating the accuracy of the end-to-end simulation of the world model is challenging, particularly due to the cumulative amplification of errors during the propagation, as we cannot guarantee that the final simulation results will align with real-world outcomes. To address this challenge, we evaluate the prediction accuracy of the world model at the individual step level. Specifically, we define the structural correctness (denoted SC) of the A11y as: $|N_{match}|/|N_{total}|$, where $|N_{match}|$ and $|N_{total}|$ denote the number of nodes that match the correct A11y structure and the total number of A11y nodes generated by the world model, respectively. The scale ratio of the generated A11y is defined as: $\mathbb{E}_i[w_i \cdot d_i / w \cdot d]$, where d_i and w_i represent the depth and width of A11y, respectively.

Model	Structural Correctness	Scale Ratio
Qwen	42.00	1.06
GPT-4	51.82	0.87
Ours	**59.80**	**1.04**

Table 6: Comparison of structural correctness and generated scale ratios with Qwen and GPT-4.

As shown in Table 6, our trained world model outperforms both Qwen and GPT-4 in structural correctness, demonstrating its significant ability to replicate real-world web environments. Two examples in Figure 9 further illustrate this. Aside from differences in node labels (i.e., numbers in brackets) and specific content, our model more accurately captures the A11y structure after state transitions. Although the red and green contents reflect individual differences, they are considered reasonable variations as long as the accuracy of the exploration space is maintained. Additionally, the scale ratio, which quantifies the difference in scale between the generated A11y and the ground-truth A11y, shows that our approach better reflects the scale of the original data compared to the other two models, thus avoiding potentially misleading exploration.

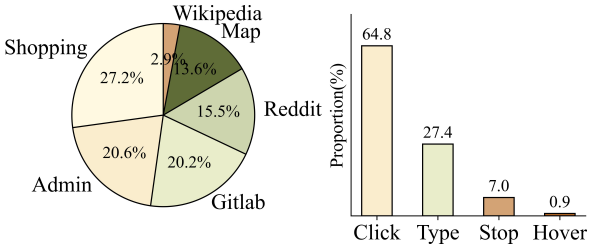


Figure 10: Distribution of web interactions across domains and action types: (Left) The pie chart shows the percentage of interactions across different web domains, with Shopping and Admin receiving the highest shares. (Right) The bar plot illustrates the proportion of different action types, with `click` action dominate.

Avg.	@go_back	@steps	@thinking
P0	1.50	7.89	393.35
P1	1.46	8.62	393.55
P2	**0.95**	**6.85**	**437.35**

Table 7: Average number (Avg.) of `go_back` actions, steps, and thinking length of the policy agent at different stages of behavior cloning. The targets of each stage are referred to in the Table 4.

F.2 Statistical Analysis of Collected Trajectories

In this subsection, we present the distribution of synthetic data across various domains and action types, as shown in Figure 10. The left panel of the figure presents a pie chart that illustrates the proportion of interactions across different web domains. The right panel shows the distribution of action types performed during WebMCTS. Click actions are the most prevalent, comprising 64.8% of all actions, followed by Type actions (27.4%), indicating that textual input is a significant component of user interactions. This distribution aligns with typical web navigation behavior, where clicking and typing are the most common user actions.

To provide further insight into the characteristics of our synthetic data, we include example cases of web interactions in Figure 11 (OneShopMarket), Figure 12 (OpenStreetMap), and Figure 13 (Shopping Admin). These examples illustrate how the policy agent interacts with the simulated web environment, with each simulation concluding once the search node successfully fulfills the user's intention.

F.3 Analysis of the Behavior Cloning Curriculum

As shown in Table 7, our staged re-weighting strategy in the behavior-cloning curriculum significantly suppresses rollback behavior, streamlines navigation, and encourages deeper deliberation. When the rollback weight is reduced from 0.30 in P0 to 0.10 in P2, the average number of `go_back` actions decreases by 37% (from 1.50 to 0.95), indicating that the agent commits to forward decisions instead of relying on trial and error. Meanwhile, the mean path length rises slightly in P1 (8.62 steps) but then falls below the baseline to 6.85 steps in P2, showing that greater emphasis on navigation promotes a more one-shot completion strategy. Finally, the average reasoning length remains stable between P0 and P1 (about 393 tokens) and then rises to 437 tokens in P2, suggesting that the agent offsets the reduced rollback safety net with more extensive pre-action planning. Figure 5 provides two cases in Shopping scenario.

G Artifact Statement

This work makes use of datasets, benchmarks, and pretrained models that are publicly available and explicitly licensed for research purposes, including WebArena (Zhou et al., 2023), VisualWebArena (Koh et al., 2024a), OS-Genesis (Sun et al., 2024), and AgentTrek (Xu et al., 2024). Our use of these resources is consistent with their intended scope as defined by the original providers.

For the artifacts we introduce, namely the WebSynthesis dataset and the pretrained policy/world models, we release them under a research-only license. These resources are intended to support further academic investigation in world model-guided trajectory synthesis and reinforcement learning. They are not suitable for real-world deployment or commercial applications, particularly in safety-critical settings.

All data released are anonymized and do not contain personally identifiable information. Re-identification of individuals is not feasible without disproportionate effort. To mitigate potential misuse, we provide explicit documentation of intended use cases and limitations. We stress that derivatives of these artifacts must remain within research contexts.

G.1 Data and Model Statement

The datasets, benchmarks, and models used and released in this work are intended solely for research and educational purposes. They may be used safely under the following conditions:

Research Use Only: All artifacts are licensed for

non-commercial academic research. They must not be deployed in production systems or used for commercial services.

Prototyping and Evaluation: The datasets and models may be used to prototype algorithms, conduct comparative evaluations, or reproduce our results in controlled research environments.

No Real-World Deployment: Derivatives of our artifacts should not be deployed in real-world applications without additional safety assessments, as they are not designed for robustness in open, uncontrolled environments.

No Sensitive Applications: The artifacts should not be applied in domains where errors may cause harm to individuals or groups, such as healthcare, finance, or social decision-making.

Data Privacy: All datasets are anonymized. They do not contain personally identifiable information. Any attempt to re-identify individuals or link the artifacts to external personal data is prohibited.

By providing these resources, our goal is to enable safe, reproducible, and extensible research on trajectory synthesis and web agent training.

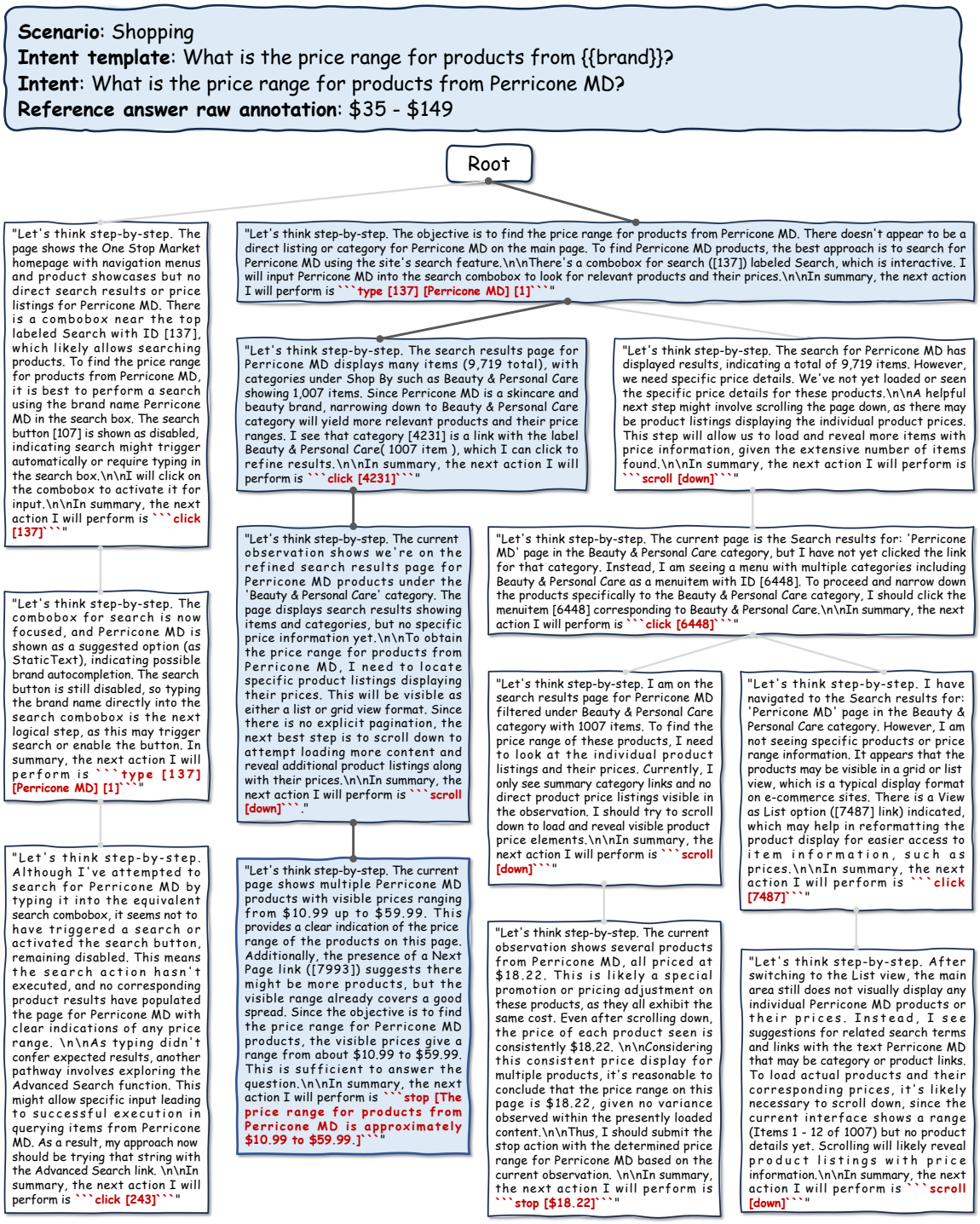


Figure 11: A case study of collected WebMCTS tree in Shopping Scenario.

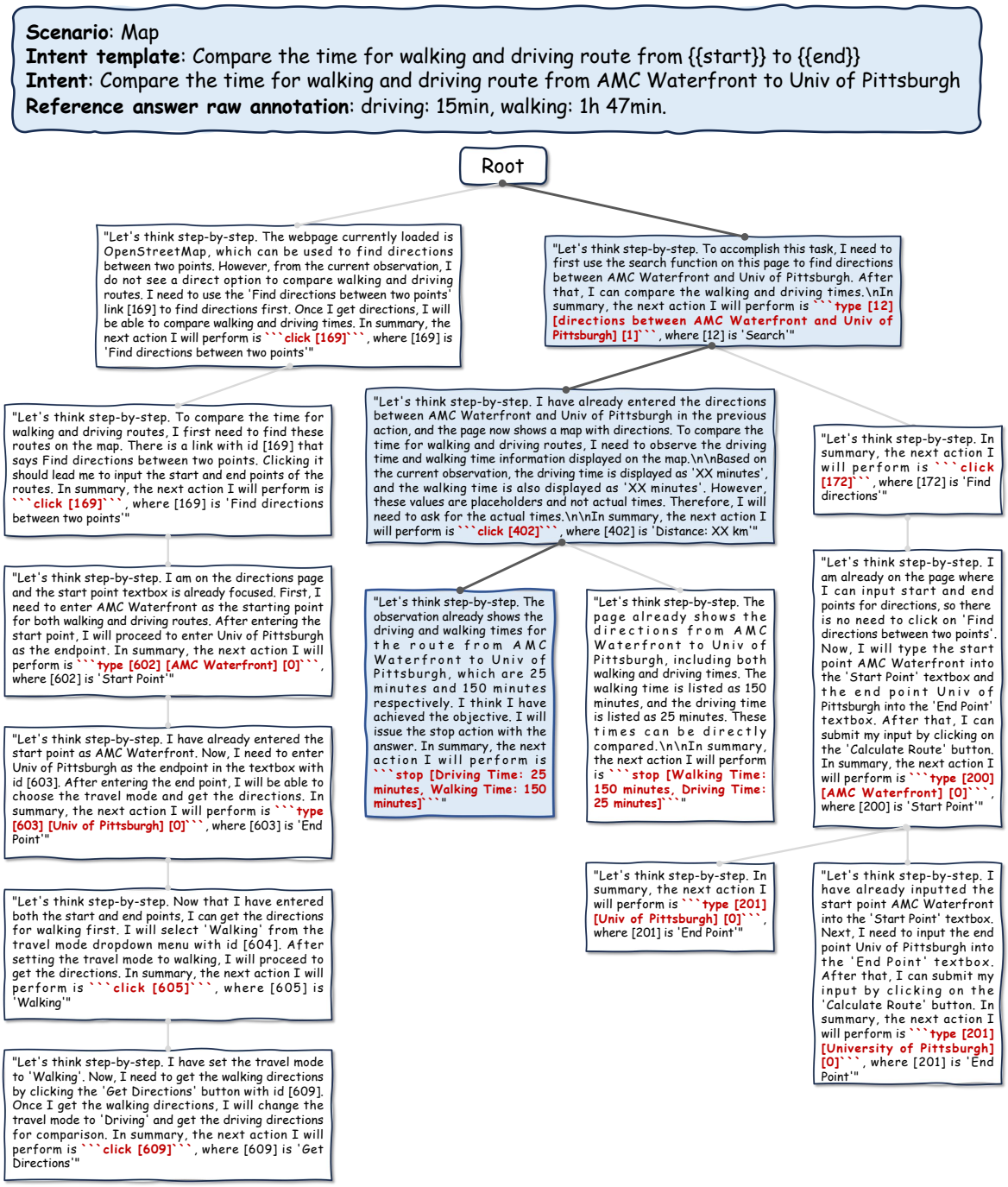


Figure 12: A case study of collected WebMCTS tree in Map Scenario.

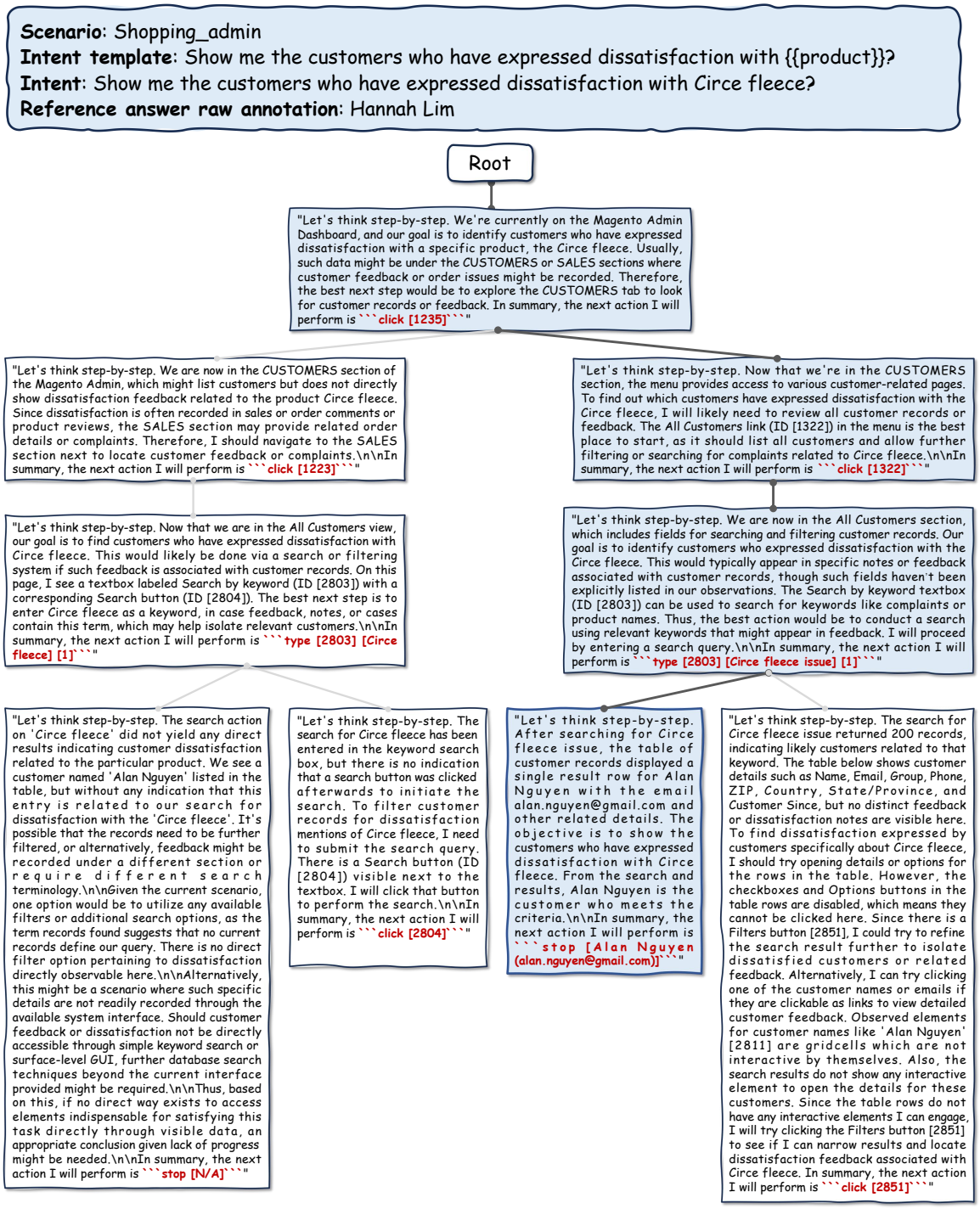


Figure 13: A case study of collected WebMCTS tree in Shopping Admin Scenario.

Instruction of Dense Captioning
Describe in detail the layout, components, and visible content of this web page.
Give a full description of the web page, including its layout and interactive elements.
Generate in detail what is shown in this web page.
Provide a comprehensive description of the current web UI state.
Thoroughly describe the content, layout, and structure of this web page.
Describe the structure and main features present in the given web page.
List all the interactive components present on this web interface.
Summarize the roles of different elements found in the current interface.
Detail the main sections and functionalities available in this interface.
Enumerate and explain all key interface components present in this state.
Describe all the functional parts and sections of the web interface.
Explain what content and interaction options exist in this page layout.
Generate a description of the components and layout in this interface.
Enumerate the key sections and describe their purpose in the interface.
List all actionable elements and describe their roles in context.
Describe all visible text, links, buttons, and forms on the page.
Explain the accessible content and its functional organization.
Describe all labeled elements and how they are grouped.
List and explain all content blocks and their positions on the page.
Summarize the types of elements present and their logical grouping.
Describe the form and functional flow of this interface in detail.
Generate a full explanation of all parts making up the page state.
Summarize the structure and functional blocks in this UI snapshot.

Instruction of Element Functionality
What is the functionality of element {**element**}. Describe what this element {**element**} does.
Explain the purpose of the element {**element**}. What role does the element {**element**} serve?
Clarify the element {**element**} function for me. Tell me how the element {**element**} operates.
What is the element {**element**}'s intended use? Outline the function of this element {**element**}
How is the element {**element**} supposed to work? State the function of the element {**element**}
What task does the element {**element**} perform? Summarize what the element {**element**} is for.
Identify the element {**element**}'s capability. Provide the element {**element**}'s operational purpose.
What action does the element {**element**} enable? What job does the element {**element**} accomplish?
Specify how the element {**element**} functions. Describe the element {**element**}'s core functionality.
What objective does the element {**element**} fulfill? Explain what the element {**element**} is designed to do.

Instruction of State Transition
Given the current UI state, if I perform the following action: ```{**action**}```, predict the content of the next UI state.
Imagine I just executed this action: ```{**action**}``` in the current UI, what does the new UI state look like?
Assuming the user performs the action ```{**action**}``` in the current state, what is the expected next state?
Please describe the next UI state that would result from executing the following action: ```{**action**}```.
What would the UI look like after taking this action: ```{**action**}``` from the current screen?
Based on the current interface, simulate the UI state transition caused by the action: ```{**action**}```.
Suppose I apply this action: ```{**action**}``` in the present UI context. What should the resulting state be?
If the user triggers this event: ```{**action**}```, what changes are expected in the UI?
Predict the subsequent UI state if the action below is performed: ```{**action**}```.
Considering the interface's current status, how would it change after: ```{**action**}```?
Please simulate the updated UI view after taking the action: ```{**action**}```.
What is the outcome of the interface transition when ```{**action**}``` is applied?
Assuming this operation is executed: ```{**action**}```, what should the UI display next?
Predict how the UI evolves when this user interaction occurs: ```{**action**}```.
From the current UI, simulate what happens next if the following is done: ```{**action**}```.
Given this user command: ```{**action**}```, determine the next interface state.
What would the state of the interface be after this interaction: ```{**action**}```?
Please show the updated UI result after applying the following step: ```{**action**}```.
Tell me how the UI would respond to this user action: ```{**action**}```.
When the action ```{**action**}``` is invoked, what's the next logical UI configuration?

Table 8: Instruction enhancement in UI fundamental understanding stage.

You're a helpful web GUI assistant.

OBSERVATION: {**Web State**}

Instruction: Detail the main sections and functionalities available in this interface.

Output: Let's think step-by-step. The web page displays a user interface from an e-commerce platform, specifically the "Pages" section. The layout is organized with a sidebar on the left featuring navigation icons for various functions such as Dashboard, Sales, Catalog, and more. The main area shows a search bar labeled Privacy Policy, indicating that users can search for specific pages. Below the search bar, there are options to manage actions and view filters. A table lists three pages with columns for ID, Title, URL Key, Layout, Store View Status, Created Date/Time Modified Date/Time Action. Each row provides detailed information about individual pages including their titles and statuses. This interface allows users to efficiently manage website content by enabling them to add new pages or modify existing ones through drop-down menus in the action column. Overall, it reflects a structured approach to page management within an e-commerce environment.

Table 9: Instruction template for {**Dense Caption**}.

You're a helpful web GUI assistant.

OBSERVATION: {**Web State**}

Instruction: What job does the element [3230] accomplish?

Output: Let's think step-by-step. The functionality of this element is to filter projects based on specific topics or themes. By selecting 'Topics', users can narrow down their search results to projects that match the chosen category, enhancing the relevance and discover ability of the projects displayed in the project list.

Table 10: Instruction template for {**Element Functionality Description**}.

You're a helpful web GUI assistant.

OBSERVATION: {**Web State**}

Instruction: Predict how the UI evolves when this user interaction occurs: click [7716], where [7716] is 'type:link, text:Submit'

Output: Let's think step-by-step. The user clicked on the "Submit" link, transitioning from a search results page to a submission creation page. This is evident from the removal of content related to search results, such as headings, articles, and links to individual posts, and the addition of elements specifically designed for creating submissions. The new page includes input fields for title and body, radio buttons for submission type (URL or image), a drop down menu for selecting a forum, and a submission button. The navigation bar remains consistent, but now supports submission-related functionalities. In summary, the webpage now displays a submission creation interface on Postmill. The top navigation bar includes links to "Postmill", "Forums", and "Wiki", along with a search bar and a "Submit" button. The main content area is structured for creating a submission, featuring input fields for Title (required), Body, and submission type options (URL or Image). Users can select a forum via a drop down menu labeled "Forum" which allows categorization of the submission. A button labeled "Create submission" is provided to finalize the process, and Markdown formatting is supported with help links available. This layout facilitates users in crafting and submitting content to the platform's forums."

Table 11: Instruction template for {**State Transmission Prediction**}.

You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue.
Here's the information you'll have:

- The user's **objective**: This is the task you're trying to complete.
- The **current observation** of web page. This is a simplified representation of the webpage, refer to as accessibility tree (a11y), providing key information.
- The previous **trajectory**: This is the "observations", "thoughts" and "actions" you have just performed. It may be helpful to track your progress. Each step is splited by <step> </step> tag.

Action Space
The actions you can perform fall into several categories:

Page Operation Actions:
click [id]: This action clicks on an element with a specific id on the webpage.
type [id] [content] [press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless press_enter_after is set to 0.
hover [id]: Hover over an element with id, this action may display hidden information about the element.
scroll [direction=down|up]: Scroll the page up or down. This action will provide new or previously appeared page information.

URL Navigation Actions:
goto [url]: Navigate to a specific url.
go_back: Navigate to the previously viewed page.

Completion Action:
stop [answer]: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Tips:

- If the page has element information that is useful for completing the task, you can perform page actions to explore, but please pay attention to the actual function of each element. It is forbidden to perform function B on an element that only has function A, such as retrieving other information besides geographic information in the search box of OpenStreetMap.
- Fuzzy search is prohibited. Your search must be based on a clear goal. For example, you can search for a pair of Nike shoes, but you are not allowed to search for a pair of shoes that cost around $60.
- If the page doesn't have information that helps you complete the task, you can perform url navigation actions, including the need to jump to a specific page (for example, jump to Reddit), or compare the information on the previous and next pages to help complete the task.
- If you think you have completed this task, please check your trajectory carefully and make a completion action carefully.
- If there is no information on the current page that can help complete the task, please also make a completion action carefully.

Action Rules:
To be successful, it is very important to follow the following rules:

1. You should think step by step and then issue the next action. Start with a "Let's think step-by-step." phrase.
2. You should only issue an action that is valid given the current web page.
3. You should only issue one action at a time.
4. Generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ``` ```. For example, "In summary, the next action I will perform is ```click [1234]```".
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Table 12: Policy Agent Prompt Template.

You are an autonomous intelligent agent tasked with navigating a web browser. You will be given a web GUI-based task. Specifically, you need to predict the next web page observation based on the current observation of the web browser and the given action.
Here's the information you'll have:

- The current web **page observation**, which lists the IDs of all interactive elements on the current web page with their text content if any, in the format [id] [tagType] [text content]. tagType is the type of the element, such as button, link, or textbox. text content is the text content of the element. For example, [1234] [button] ['Add to Cart'] means that there is a button with id 1234 and text content 'Add to Cart' on the current web page. [] [StaticText] [text] means that the element is of some text that is not interactive.

- The given **action**: This is the action you have already performed. The actions you have performed fall into several categories:

Action Space
The actions you can perform fall into several categories:

Page Operation Actions:
click [id]: This action clicks on an element with a specific id on the webpage.
type [id] [content] [press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless press_enter_after is set to 0.
hover [id]: Hover over an element with id, this action may display hidden information about the element.
scroll [direction=down|up]: Scroll the page up or down. This action will provide new or previously appeared page information.

URL Navigation Actions:
goto [url]: Navigate to a specific url.
go_back: Navigate to the previously viewed page.

Completion Action:
stop [answer]: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

IMPORTANT
To be successful, it is very important to follow the following rules:

1. Please think step by step based on the current page observation and the actions taken, and give the maximum possible next page observation.

2. You should ensure the richness of the observations of the web page to be predicted and support the continuous operation process.

3. Please generate the content of the next page in the correct format. Start with the phrase "In summary, the next web page observation is" and then add supplements within ```<your generated contents>```. For example, "In summary, the next web page observation is ```Tab 0 (current): Projects ```".

4. When you think you have achieved the full content prediction of the next page, issue a stop operation with [END]. Do not generate any content after stopping,

Table 13: World Model Prompt Template.

You are an expert in evaluating GUI agent task trajectories. Your task is to assess the quality and effectiveness of task trajectories for GUI manipulation tasks.

A trajectory consists of the following components:

1. **User Instruction**: Describes the user's intended task.

2. **Action History**: Includes two key parts:

   - Reasoning and Action for Each Step: A sequence of actions performed by the agent, including the reasoning thought and final executed action.
   - The accessibility tree of the current web page: This is a simplified representation of the webpage, providing key information.

When evaluating a trajectory, consider these key aspects:

Evaluation Criteria:
1. Trajectory Coherence:

- Do the steps and corresponding actions follow a logical sequence toward the goal?

- Are the actions clearly described and specific?

- Are there redundant or unnecessary actions?

2. Task Completion:

- Does the trajectory successfully achieve the instructed task?

- Are all necessary interactions completed?

- Are error cases handled appropriately?

Scoring Guidelines:
Rate the trajectory on a scale of 1 to 5 based on the evaluation criteria:

- 5: The task is perfectly completed, successfully executing multiple actions to achieve the goal or return the correct answers. The sequence is logically clear with no noticeable redundancies.

- 4: The task is mostly completed, successfully executing multiple actions. However, due to challenges or ambiguities in the instructions, the completion is not perfect, or there are inefficiencies in the process.

- 3: The task is partially completed, with some successful actions executed. However, due to task or environmental constraints, the goal is not fully achieved, or the sequence ends in a loop or error.

- 2: Only a few actions are executed. Although there is an attempt to complete the task, the trajectory deviates from the goal early on or demonstrates significant inefficiencies in execution and logic, e.g., repeat the same action.

- 1: The task fails completely, with no meaningful actions executed at the start. The sequence either falls into an immediate deadlock, a repetitive loop, or demonstrates no value in completing the task.

Or the tasks are completely inaccessible.

Note: If the task is relatively complex, but the trajectory demonstrates valuable attempts, even if the task is not fully completed, consider adjusting the score upward. However, if the task is complex but the trajectory fails to perform actions that contribute meaningfully to task completion, no extra points should be awarded.
You need to judge the score based on the user instruction, agent's actions and the current state of the webpage combined.

Response Format:
Format your response into two lines as shown below:
Reason: <your thoughts and reasoning process for the score>
Score: <your score from 1-5>

Table 14: Reward Model Prompt Template.

Instruction:
<insert user objective here>

Accessibility tree:
<insert current observation here>

Action History:
step-i
What's the next action?

Table 15: Instruction Template of OS-Genesis.

Goal:
<insert user objective here>

 # Observation of current step:
AXTree:
<insert current observation here>

 # History of interaction with the task:
step-i

- ### Action <insert historical action here>

- ### Memory: <insert memory of each step here>

. . .

 # Output Format:

- <think> <insert your thought here> </think>

- <memory> <insert your memory here> </memory>

- <action> <insert your action here> </action>

Table 16: Instruction Template of AgentTrek.

OBJECTIVE:
<insert user objective here>

TRAJECTORY:
<step-i>

- OBSERVATION: <insert historical observation here>

- REASON FOR ACTION: <insert historical thought here>

- ACTION: <insert historical action here>

</step-i>
. . .

OBSERVATION:
<insert current observation here>

What's the next action?

Table 17: Instruction Template of WebSynthesis.

You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue.
Here's the information you'll have:

- The user's **objective**: This is the task you're trying to complete.
- The **current observation** of web page. This is a simplified representation of the webpage, refer to as accessibility tree (a11y), providing key information.
- The previous **trajectory**: This is the "observations", "thoughts" and "actions" you have just performed. It may be helpful to track your progress. Each step is splited by <step> </step> tag.

Action Space
URL Navigation Actions:
go_back: Navigate to the previously viewed page.
Homepage:
If you want to visit other websites. Here is a list of websites you can visit.

- Gitlab: http://gitlab.com
- OpenStreetMap: https://www.openstreetmap.org
- OpenStopMarket: http://onestopmarket.com
- Admin: http://luma.com/admin
- Redit: http://reddit.com

Action Rules:
To be successful, it is very important to follow the following rules:

1. You should think step by step and then issue the next action. Start with a "Let's think step-by-step." phrase.
2. You should only issue an action that is valid given the current web page.
3. You should only issue one action at a time.
4. Generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ``` ``` ```. For example, "In summary, the next action I will perform is ```click [1234]```".
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

OBJECTIVE:
<**Insert your objective here.**>

Previously, the action:
```
<**Insert the failure node action here.**>
```
has been attempted, and this action will not lead to the task completion. Please provide an action for going back to the last observation following the aforementioned format. Give your reason why this action cannot help to complete the task.

Last Observation:
<**Insert the last page observation here.**>

Current Observation:
<**Insert the current page observation here.**>

What's the next action?

Table 18: Reflection Prompt for Rollback Trajectories Collection.