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

Yifei Gao[1], Junhong Ye[1], Yifan Yang[1], Jiaqi Wang[1],
Yi Zhang[3], Ruichen Zhang[4], Jitao Sang[12]*
[1]Beijing Jiaotong University, Beijing, China, [2]State Key Laboratory of AI Safety, Beijing, 100086
[3]Alibaba Business School & Research Center for Complexity Sciences,
Hangzhou Normal University, Hangzhou, China
[4]School of Computer Science and Engineering, Nanyang Technological University
{yifeigao, jtsang}@bjtu.edu.cn

Abstract

Recent advances in large language models (LLMs) have enabled increasingly capable web agents, yet training such agents still relies on high-quality interaction trajectories that are difficult to obtain at scale. We identify two key challenges: (1) Infrastructure Overhead, where network instability and website access restrictions limit data collection scalability; and (2) Constrained Exploration, where irreversible state transitions preclude tree-based search and thus limit trajectory diversity. To address these challenges, we introduce WebSynthesis, a framework for scalable trajectory synthesis. WebSynthesis employs an LLM-based World Model to simulate state transitions without network dependencies, and integrates Monte Carlo Tree Search to enable reversible exploration over the simulated state space. Experiments on WebArena, WebVoyager, and Mind2Web-Online demonstrate that agents trained exclusively on synthesized trajectories outperform those trained on real-world data, providing a viable alternative to costly real-world data collection. Our code is now available at Github.

1 Introduction

Large language model (LLM)-powered web agents have emerged as a promising paradigm for autonomous web interaction, enabling systems to complete complex, multi-step tasks through iterative perception-action loops (Ferrag et al., 2025; Zhang et al., 2024a). Central to advancing these capabilities is training on high-quality interaction trajectories (Zheng et al., 2024). However, acquiring such trajectories at scale remains a critical bottleneck, as current approaches rely on interacting with real websites to collect or generate training data (Qin et al., 2025; Sun et al., 2024; Xu et al., 2024). This reliance on live web environments introduces two fundamental challenges:

(i) Infrastructure Overhead. Scaling real-world web interactions demands stable network connectivity and compliant access to target websites, yet neither condition holds reliably in practice (Qi et al., 2024). Network-level instabilities cause frequent connection timeouts and session failures, fragmenting the data collection pipeline. Concurrently, websites deploy defensive mechanisms such as rate limiting and IP blocking that throttle request throughput (Pahuja et al., 2025; Xu et al., 2024). Even controlled testbeds like WebArena (Zhou et al., 2023) offer no relief, as each simulated website requires dedicated server infrastructure. These compounding factors render large-scale trajectory collection prohibitively expensive (Gandhi and Neubig, 2025), creating a scalability ceiling that current methods cannot overcome.

(ii) Constrained Exploration. Beyond infrastructure costs, real web environments impose a more fundamental limitation on exploration strategies. Live websites do not support state checkpointing or rollback (Dihan et al., 2025; Zhang et al., 2025b); once an action is executed, the resulting state change becomes irreversible (Chae et al., 2024). This irreversibility precludes tree-based search algorithms, which require branching and backtracking from identical states to explore alternative action sequences (Koh et al., 2024b). Consequently, existing methods are confined to linear sampling strategies that traverse the state-action space conservatively (Zhang et al., 2025b). The resulting trajectories exhibit limited coverage and diversity, starving downstream training of the varied experiences necessary for robust generalization.

To address these challenges, we introduce **WebSynthesis**, a framework that synthesizes diverse web agent trajectories entirely offline. WebSynthesis comprises two synergistic components. The **LLM-based World Model (LWM)** learns state transition dynamics from web interactions, enabling prediction of next states given current ob-

*Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 25241–25258

servations and actions. By substituting real website requests with model predictions, LWM eliminates both network instability and access restrictions. Building upon this foundation, **Monte Carlo Tree Search (MCTS)** (Świechowski et al., 2022) exploits LWM's capacity to simulate arbitrary state transitions from any checkpoint. Unlike real environments where actions leave permanent traces, LWM permits MCTS to backtrack and branch freely, enabling systematic exploration that yields high-coverage training trajectories.

We evaluate WebSynthesis on three online benchmarks: WebArena (Zhou et al., 2023), WebVoyager (He et al., 2024), and Mind2Web-online (Xue et al., 2025). In WebArena, WebSynthesis achieves a success rate of 28.34%, substantially outperforming OS-Genesis (18.66%), which trains on large-scale real-world interactions. Similar improvements are observed on WebVoyager and Mind2Web-online, demonstrating that synthesized trajectories can match or exceed the effectiveness of real-world data collection. Notably, despite imperfect state prediction accuracy, agents trained on synthesized trajectories generalize effectively to real websites. These results establish WebSynthesis as a viable alternative to costly real-world data collection and provide insights for designing scalable autonomous web agents.

Our contributions includes:

- We identify two critical challenges in web agent trajectory collection: infrastructure overhead from real-world interactions and constrained exploration due to irreversible state transitions.
- We propose WebSynthesis, which utilizes an LWM to eliminate network dependencies and integrates MCTS to enable systematic state-space exploration through tree search.
- We demonstrate that agents trained on WebSynthesis achieve and sometimes outperform real-world data collection on three benchmarks, providing a viable alternative to costly real-world data collection.

2 Related Works

Data Synthesis. These approaches address data scarcity by synthesizing training trajectories. However, they face several challenges: (1) Self-directed exploration often yields low-diversity trajectories, causing learning to stagnate as agents repeatedly encounter familiar 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 edge cases underexplored (Xu et al., 2024). (3) Accurately modeling the full complexity of web environments remains inherently difficult (Zhou et al., 2023; Koh et al., 2024a), resulting in limited scenario coverage or unrealistic behaviors. (4) Existing synthesis efforts focus primarily on generating successful trajectories while largely overlooking error recovery (Wu et al., 2025), producing agents that lack robustness when mistakes occur in real-world deployment (Hu et al., 2025; Zhang et al., 2025b).

World Models. World models have been incorporated into the "simulate-before-act" paradigm (Gu et al., 2024b) for web navigation. 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. However, both methods fail to accurately predict the structural information of subsequent web states, limiting their applicability to multi-step decision-making. WebEvolver (Fang et al., 2025) introduces co-evolution of the policy model and the world model, enabling the agent to refine its decisions within an increasingly accurate simulated environment. Our WebSynthesis uses a world model to predict the structural webpage states, enabling a continuous tree-search process.

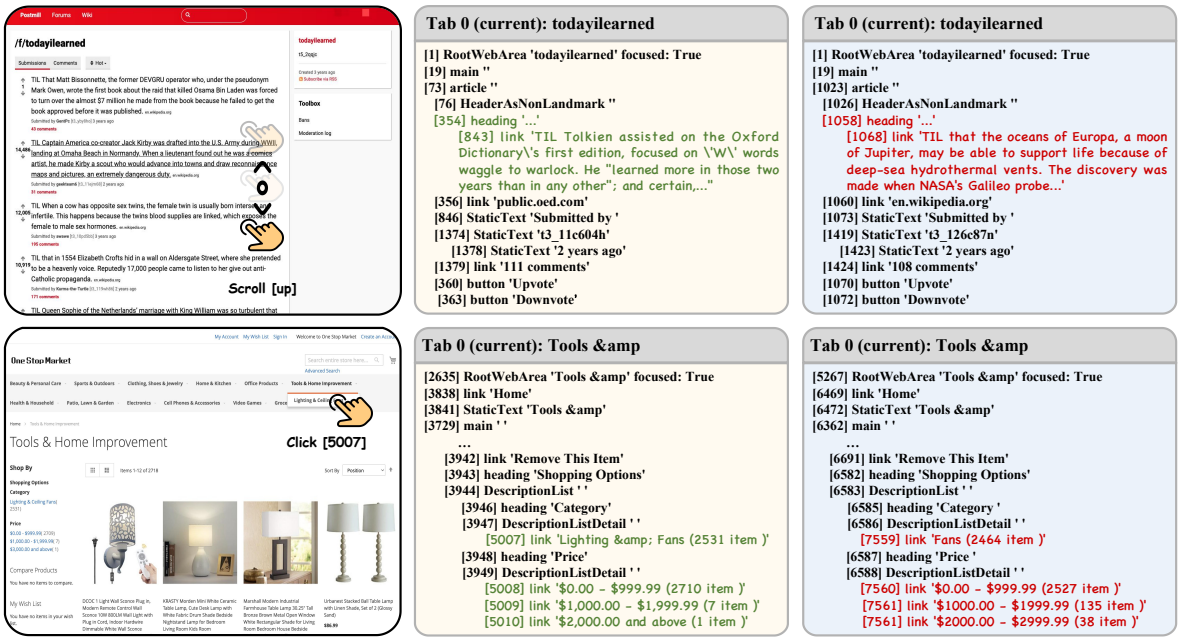
3 Methodology

This section details the WebSynthesis framework. We first formalize the problem setting (§3.1), then introduce our two-stage data synthesis pipeline: world model construction (§3.2) and MCTS-guided trajectory generation (§3.3). In §3.4, we provide theoretical analysis showing that our framework can select high-value trajectories. Finally, we describe the curriculum training strategy in §3.5.

3.1 Problem Formulations

We formulate web navigation as a partially observable Markov decision process (POMDP) (Ding et al., 2024; Cao et al., 2025). Specifically, it comprises four components: observation space $\mathcal{O}$, action space $\mathcal{A}$, transition function $\mathcal{T}$, and reward function $\mathcal{R}$. Based on this formulation, we define the policy model, world model, and reward model as follows:

Policy Model. At each time step t, the policy model π selects an action a_t based on the current



(a) The screenshot of webpage (b) Actual state (c) Synthesized state

Figure 1: Examples of LWM synthesized web page state: (Top) Reddit, with action: `scroll`; (Bottom) OpenShop, with action: `click`. Contexts in red and green indicate the differences between the synthesized and actual states.

observation o_{t-1} and the user's instruction q.

World Model. The web world model w acts as the transition function $\mathcal{T}$, mapping a state-action pair (o_{t-1}, a_t) to the next observation o_t.

Reward Model. The reward model r evaluates the environment feedback based on the current observation o_t, the action a_t taken by the policy model and the user's instruction q.

The objective of web navigation is formulated as:

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

where $o_t \sim w(\cdot|o_{t-1}, a_t)$, $a_t \sim \pi(\cdot|o_{t-1}, q)$, and o_0 denotes the initial web page observation. The policy model π processes the user query q and engages in multi-step interactions within the world model environment. The term $\arg\max_{a_{1:T}}$ refers to the search algorithmwhich aims to identify a trajectory that maximizes the cumulative reward within a fixed step budget T. Consistent with prior work (Zhou et al., 2023; He et al., 2024), we represent web page observations using the accessibility tree (A11y), which captures a structured set of accessibility-related states and properties.

3.2 LLM-based World Model

Accurate state prediction is critical for supporting multi-step exploration, as the world model must continuously respond to the policy model's action requests throughout the search process. As shown in Figure 1, the world model represents each webpage as an accessibility tree and predicts how the tree structure evolves after an action is executed. For instance, when the agent clicks element `[5007]`, the model generates a new A11y reflecting the updated page structure, including newly appeared elements and modified content. To train the world model, we deploy an exploration agent to explore the target websites and record state transitions after each UI event is executed. Each piece of transition is represented as a triplet: (o_{t-1}, a_t, o_t), where a_t denotes the executed action (e.g., click, type, hover, or scroll).

3.3 World Model-guided MCTS

To explore diverse trajectories within the world model, we adopt Monte Carlo Tree Search (MCTS), as shown in Figure 2. We formulate world model-guided MCTS (**WebMCTS**) as an iterative algorithm comprising three stages: node selection, action expansion, and backpropagation. At each iteration, the search tree expands based on candidate

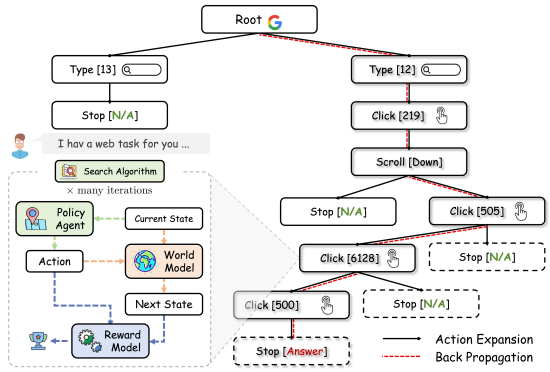


Figure 2: Pipeline of the world model-guided Monte Carlo Tree Search (WebMCTS). The solid lines represent the action expansion, while the red dashed lines indicate backpropagation of high-value trajectories.

actions proposed by the policy model π. Each node 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. Node selection is guided by the Upper Confidence Bound (UCB) strategy (Zhang et al., 2024b; Świechowski et al., 2022), 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 via the UCB strategy, candidate actions are sampled from its associated state. To encourage exploration and maintain diversity, WebMCTS samples at least three distinct actions at each expansion step. Since different branches may reach the same web page at different stages of the search, we employ a caching mechanism to ensure consistent state predictions. Specifically, we maintain a hash table mapping each URL to its corresponding page state previously generated by the world model. When the same URL is encountered again, the cached state is reused, avoiding redundant generation and improving both consistency and efficiency.

Backpropagation. Finally, value backpropagation is initiated from the selected node C, updating ancestor node values via weighted averaging. Specifically, the visit count n_C and estimated value v_C are updated as:

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

Through repeated iterations, this process constructs a web action tree within the world model that captures diverse interaction paths. The Appendix B presents the detailed data synthesis process and provides examples of synthesized search trees.

3.4 Node Selection with Noisy Predictions

In practice, the world model inevitably suffers from prediction errors, such as undefined UI interactions in web navigation. These errors introduce stochasticity into MCTS node value estimates, raising a critical question: *Can MCTS reliably identify optimal nodes under prediction noise?* To address this, we analyze how prediction errors propagate during node selection.

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 increases. This implies that even under imperfect world model predictions, the selection errors diminish rapidly with continued exploration. Consequently, high-value trajectories eventually dominate as MCTS explores more paths, confirming that our framework can reliably synthesize quality trajectories despite prediction noise.

3.5 Curriculum Learning

Considering that complex web navigation requires both understanding UI semantics and executing multi-step actions; training these skills jointly from

scratch is challenging (Qin et al., 2025). Therefore, we employ a curriculum learning strategy to progressively enhance navigation capabilities. Training consists of two stages: user interface understanding and behavior cloning.

3.5.1 User Interface Understanding

Web pages represented as accessibility trees are often lengthy and complex, making it challenging for models to fully comprehend UI semantics (Lin et al., 2025). To address this, we design a preliminary training stage focused on UI understanding. Following ProphetAgent (Kong et al., 2025), we collect transition tuples (o_{t-1}, a_t, o_t) by exploring web environments. We then derive three training tasks: (1) dense captioning, (2) element functionality prediction, and (3) state transition prediction. Dataset construction details are provided in Appendix C.

3.5.2 Behavior Cloning

The search tree constructed by WebMCTS contains both successful trajectories and failed attempts. We leverage both types for training:

Valuable Trajectory. Since the initial policy model may produce redundant or meaningless actions, we first prune the action tree before trajectory collection. We then apply depth-first search (DFS) to locate nodes whose values exceed a predefined threshold. For each identified node, we trace the path back to the root and extract the corresponding trajectory $\tau = \{o_t, a_t\}_{t=1}^{N}$ for training.

Rollback Trajectory. Beyond successful paths, failed exploration branches provide valuable learning signals for error recovery. Unlike traditional linear rollback methods (Zhang et al., 2025a; Qin et al., 2025), which yield only one trajectory per run, WebMCTS preserves multiple parallel branches, enabling discovery of diverse rollback trajectories within a single search episode. For each intermediate node C on a valuable trajectory, we identify its unsuccessful sibling nodes S and their parent P. We then construct rollback trajectories of the form $S \rightarrow P \rightarrow C$, where the agent learns to recognize failed actions, execute `go_back`, and resume from the previous valid state. This exposes the policy model to corrective behaviors that enhance navigation robustness.

4 Experiments

We conduct experiments to validate the efficacy of synthesized trajectories. We first describe the experimental setup (§4.1) and evaluate performance on three online benchmarks (§4.2). We then present ablation studies on each component of WebSynthesis (§4.3) and analyze the effects of world model prediction errors (§4.4). Finally, we examine data scaling behavior (§4.5).

4.1 Experiment Setup

WebSynthesis. We use Qwen2.5-7B/14B-Instruct as base models for SFT training, obtaining WebSynthesis-7B and WebSynthesis-14B. We collect approximately 7.9k trajectories through WebMCTS. Following standard web agent setups, we adopt an action space consisting of `click`, `type`, `scroll`, `goto`, `go_back`, and `stop`.

Baseline Methods. We compare our approach with the following baseline methods: (1) Qwen2.5-7B-Instruct and GPT-4, evaluated using chain-of-thought (CoT) prompting (Zhou et al., 2023); (2) OS-Genesis (Sun et al., 2024), which trains 7.4k real-world trajectories; and (3) AgentTrek (Xu et al., 2024), which generates trajectories from publicly available tutorials. For AgentTrek, we sample 20k trajectories matching the WebArena action space. To ensure fair comparison, all methods use text-based state representations.

Evaluation Benchmarks. We select three live web navigation benchmarks for our experiments: WebArena (Zhou et al., 2023), WebVoyager (He et al., 2024), and Mind2Web-Online (Xue et al., 2025). In these benchmarks, the web agent is required to make sequential decisions from an initial state to complete a given task. For WebArena, we evaluate on 542 queries across three websites (Shopping, Admin, and Reddit), excluding GitLab due to network instability. For WebVoyager,we evaluate on 389 queries from a subset of websites (AllRecipes, Apple, ArXiv, BBC, Coursera, ESPN, Google Map, Huggingface, and WolframAlpha) due to network accessibility constraints. For Mind2Web-Online, we evaluate on the complete set of 300 queries. Evaluation metrics follow each benchmark's protocol: WebArena uses string matching combined with LLM-as-judge, WebVoyager adopts LLM-as-judge, and Mind2Web-Online uses WebJudge (Xue et al., 2025).

4.2 Evaluation Results on Online Benchmarks

To validate whether synthetic trajectories can effectively transfer navigation capabilities to real-world web environments, we evaluate WebSynthesis on three challenging online benchmarks.

Model	All Recipes	Apple	ArXiv	BBC	Coursera	ESPN	Google Map	HF	Wolfram Alpha	WV Avg.	M2W Online
Qwen2.5-7B	11.36	9.30	39.02	12.20	40.48	29.55	39.02	20.93	23.26	23.46	9.27
GPT-4	31.11	23.26	40.62	42.50	**53.14**	38.64	43.66	46.51	44.19	35.71	11.36
AgentTrek	19.84	23.15	38.21	19.56	38.24	18.28	43.21	32.79	38.64	29.31	14.57
OS-Genesis	26.67	16.28	40.48	29.27	40.48	27.27	41.22	34.88	**48.84**	31.40	12.50
WebSynthesis-7B	**42.22**	23.26	33.33	45.00	50.00	27.27	**48.78**	34.88	46.21	35.82	28.79
WebSynthesis-14B	40.00	**25.58**	**42.38**	**53.66**	42.86	**45.45**	46.34	**51.91**	46.67	**39.29**	**31.82**

Table 1: Task success rate (%) on WebVoyager (**WV**) and Mind2Web-Online (**M2W-Online**) (%). **Bold** and Underline numbers indicate the best score and the second best score in each column.

Model	Shopping	Admin	Reddit	Overall
Qwen2.5-7B	2.17	0.00	0.00	2.24
GPT-4	13.04	17.14	9.52	13.58
Pass@1				
AgentTrek	15.22	11.43	4.76	9.70
OS-Genesis	10.87	14.29	0.00	11.19
WebSynthesis-7B	15.34	17.14	12.43	15.22
WebSynthesis-14B	17.82	19.08	15.68	17.28
Pass@3				
AgentTrek	19.57	11.43	9.52	11.94
OS-Genesis	19.57	31.43	0.00	18.66
WebSynthesis-7B	**28.26**	22.86	12.43	25.93
WebSynthesis-14B	26.09	**31.43**	**16.58**	**28.34**

Table 2: Task success rate (%) on different WebArena subsets. We report on 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.

Alg.	World	Reward	Depth	Width	Steps	Overall
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 3: 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 overall performance.

Results on WebArena. As shown in Table 2, WebSynthesis-14B achieves 17.28% Pass@1 and 28.34% Pass@3, substantially outperforming both OS-Genesis and AgentTrek. Notably, WebSynthesis shows the largest improvement from Pass@1 to Pass@3 (+11.06%), compared to OS-Genesis (+7.47%) and AgentTrek (+2.24%), suggesting that WebSynthesis expands the agent's capability boundary for task completion.

Results on WebVoyager and Mind2Web-Online. Real websites present more complex accessibility trees compared to WebArena's controlled environment, posing a greater challenge for generalization. As shown in Table 1, WebSynthesis-14B achieves the highest average scores on both WebVoyager (39.29%) and Mind2Web-Online (31.82%), outperforming OS-Genesis by 7.89% and 19.32% respectively. Performance on certain websites (e.g., Coursera, Wolfram Alpha) is relatively lower, as their accessibility trees contain substantially more URL elements and complex nested structures that are underrepresented in our current seed pages. Incorporating more diverse seed pages during trajectory generation can address this limitation, which we leave for future work. Across all three benchmarks, WebSynthesis demonstrates that agents trained on synthetic trajectories can effectively generalize to real web environments. These results validate trajectory synthesis as a viable alternative to real-world data collection.

4.3 Ablation Studies

To analyze the contribution of each component, we conduct ablation studies on WebArena-Lite (Qi et al., 2024) using Qwen2.5-7B as the policy model. We examine two aspects: the impact of different components in WebMCTS on synthesized data quality (§4.3.1), and the contribution of different training stages and trajectory types in WebSynthesis (§4.3.2).

4.3.1 Ablation on WebMCTS

As shown in Table 3, replacing any key component of WebMCTS with a weaker alternative degrades both trajectory quality and downstream performance.

World Model. Replacing our learned world model with GPT-4's prompt-based simulation reduces the

UI Understanding			Behavior Cloning		Overall
Cap.	Func.	Trans.	τ_{roll}	τ_{val}	
			✓		2.4
				✓	9.6
			✓	✓	15.6
✓			✓	✓	16.8
✓	✓		✓	✓	18.4
✓	✓	✓	✓	✓	**24.1**

Table 4: Contribution of each training data. "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. The shaded row represents the full WebSynthesis configuration.

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.

Reward Model. Replacing the reward model with Qwen2.5-72B-Instruct significantly lowers performance (Pass@1: 13.9%), despite maintaining comparable tree depth and width. This indicates that the reward model plays a decisive role in guiding search toward high-reward states.

Search Algorithm. Substituting MCTS with Beam Search results in the longest evaluation trajectories (Steps: 6.8) but reduced performance (Pass@1: 15.2%). Without strategic backtracking, Beam Search tends to explore unnecessarily long paths and fails to terminate early upon reaching near-optimal states.

In contrast, the full WebMCTS achieves both the shortest trajectories and the best Pass@1, confirming that combining a robust world model, effective reward estimation, and principled exploration is critical for scalable trajectory synthesis.

4.3.2 Ablation on WebSynthesis Training

On UI Understanding. As shown in Table 4, the two-stage curriculum learning framework gradually improves the UI understanding capabilities of the policy model. Notably, the introduction of the state transition task in the third stage resulted in a 5.7% 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 4 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 (2.4% 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 (9.6% 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 (15.6% 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 TextUI warm-up prior to trajectory-level fine-tuning, achieves the best performance (24.1%). This highlights the importance of UI understanding in familiarizing the model with text-based UIs, thus enabling more effective policy learning in subsequent training stages.

4.4 Analysis of World Model Fidelity

Although WebSynthesis demonstrates effective generalization to real-world environments, a natural question arises: does world model fidelity matter? To investigate this, we evaluate state prediction quality using ground-truth trajectories of length

Model	depth 1		depth 2		depth 3		depth 4		depth 5	
	STR	Sim.	STR	Sim.	STR	Sim.	STR	Sim.	STR	Sim.
GPT-4	51.82	50.19	40.18	6.20	12.79	4.55	20.18	4.81	16.05	3.90
Qwen2.5 (SFT∼27k)	64.32	**82.36**	51.87	19.59	41.38	16.74	39.65	12.09	28.66	8.80
WebMCTS	**67.59**	81.95	**57.87**	**40.52**	**53.47**	**34.85**	**42.77**	**32.50**	**38.66**	**31.90**

Table 5: State prediction quality with oracle policy guidance over 5 steps. The policy model executes ground-truth actions while the world model generates state feedback. STR: structural correctness; Sim.: similarity to actual states.

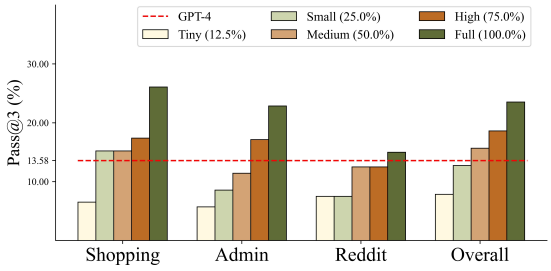


Figure 3: Performance improvement as the amount of synthesized data increases. The bars represent different data scales, ranging from 12.5% (Tiny) to 100% (Full). The dashed line indicates the performance of GPT-4 (13.58%) for reference.

5. Starting from the initial state, we feed each (state, action) pair to the world model and compare the predicted next state against the actual state. We evaluate three settings: (1) GPT-4 based world model with low temperature for stable predictions, (2) our fine-tuned world model (SFT) with low temperature, and (3) WebMCTS, which generates diverse candidate states (temperature=1.0) and selects the highest-quality state through tree search. Following prior work, we measure structural correctness (STR) and similarity (Sim.) to evaluate prediction quality.

As shown in Table 5, all methods exhibit performance degradation as depth increases. At depth 5, WebMCTS maintains 38.66% STR and 31.90% similarity, compared to 12.09%/8.80% for SFT and 4.81%/3.90% for GPT-4. The sharp degradation of GPT-4 indicates that prompt-based world models struggle to maintain coherent state representations over multiple steps, as errors accumulate rapidly. The fine-tuned model shows improved robustness due to explicit training on state transitions, but still suffers from error accumulation. In contrast, WebMCTS selectively retains high-quality states during search, preventing erroneous predictions from propagating into subsequent steps. This filtering mechanism explains why agents trained on WebMCTS-generated trajectories generalize effectively despite imperfect world model predictions.

4.5 Data Scaling Analysis

To investigate the effect of data scaling on policy performance, we vary the proportion of WebSynthesis training data. As shown in Figure 3, performance improves steadily with increasing data scale. Notably, at 50% of the dataset (approximately 3.5k samples), performance already matches GPT-4, demonstrating the sample efficiency of WebMCTS-synthesized trajectories. At full scale, WebSynthesis achieves 24.1% Pass@1, substantially outperforming baselines trained on real-world data. These results suggest that synthetic trajectory generation is a scalable approach for training web agents.

5 Conclusion

We present WebSynthesis, a framework that addresses two fundamental challenges in web agent trajectory synthesis: infrastructure overhead from real-world interactions and constrained exploration from irreversible state transitions. By combining an LLM-based World Model with Monte Carlo Tree Search, WebSynthesis enables scalable, offline trajectory generation that eliminates network dependencies while supporting systematic tree-based exploration. Experiments on WebArena, WebVoyager, and Mind2Web-Online demonstrate that agents trained exclusively on synthesized trajectories outperform those trained on large-scale real-world data. These results suggest that diverse exploration through world models can compensate for imperfect environmental fidelity, providing a promising direction for scalable web agent development.

6 Limitation

WebSynthesis demonstrates the potential of leveraging world models to replace real web environments for offline data collection, but our work also

reveals several limitations.

Limited Website Coverage. The world model is trained on a limited set of seed websites, which may not fully capture the diversity of real-world web interfaces. As shown in our experiments, performance on websites with complex accessibility tree structures (e.g., Coursera) remains lower due to insufficient coverage in training data.

Evaluation on General Scenarios Only. This work demonstrates the effectiveness of synthetic data on general web navigation tasks. Future work should explore more challenging scenarios:

- *Robustness enhancement through mixed training*: Combining synthetic data with real data can improve model robustness. For instance, the world model can generate webpage states containing pop-ups, error messages, or other edge cases that are difficult to encounter during normal data collection, enabling agents to handle diverse scenarios.
- *Low-cost simulation for restricted scenarios*. For websites with limited access such as financial services, world models provide a simulation environment that avoids repeated real network requests, which often involve compliance requirements, API costs, and rate limiting risks.
- *Integrating world models into online reinforcement learning.* In this setting, the policy model could continuously refine its decision-making by interacting with a simulated environment constructed by the world model.

7 Acknowledgement

This work is supported by the National Key R&D Program of China (No. 2023YFC3310700) , Open Funding Programs of State Key Laboratory of AI Safety, and the National Natural Science Foundation of China (No. 2576030).

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*.

Mahir Labib Dihan, Tanzima Hashem, Mohammed Eunus Ali, and Md Rizwan Parvez. 2025. Weboperator: Action-aware tree search for autonomous agents in web environment. *Preprint*, arXiv:2512.12692.

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.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand. Association for Computational Linguistics.

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.

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.

Zhiyu Lin, Zhengda Zhou, Zhiyuan Zhao, Tianrui Wan, Yilun Ma, Junyu Gao, and Xuelong Li. 2025. WebUIBench: A comprehensive benchmark for evaluating multimodal large language models in WebUI-to-code. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 15780–15797, Vienna, Austria. Association for Computational Linguistics.

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

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*.

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*.

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*.

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

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.

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

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

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:

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

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 The Collection Process of WebMCTS

We use the 241 task templates provided by WebArena (Zhou et al., 2023) to generate instructions for running tree searches. Task instructions are synthesized from open-source trajectory data provided by AgentTrek (Xu et al., 2024) and GoBrowser (Gandhi and Neubig, 2025), and the initial states are collected from the beginning of each trajectory. As shown in Algorithm 1, 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 greedy search to determine the output trajectory. Since the policy agent tends to issue a `stop[N/A]`

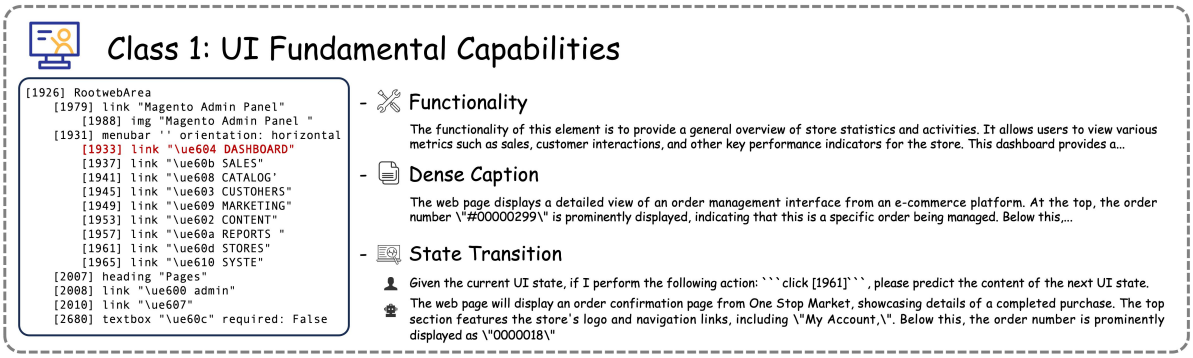


Figure 4: Overview of the UI Understanding datasets. It provides single-step, fine-grained annotations, including dense captions, element functionality descriptions, and state transitions predictions, designed to train the model to understand, describe, and predict web page states.

Algorithm 1: The collection of WebMCTS

Input: task instruction q, initial state o_0,
policy agent π, world model w, reward model r
Output: synthesized search tree $\mathcal{T}$
Init: MaxTime $\leftarrow$ 800s ; // max search time
$\theta \leftarrow 4.5$; // threshold of node value
$\mathcal{T} \leftarrow \{n_0(o_0, q)\}$; // init search tree with root
$t_{start} \leftarrow$ CurrentTime()

while *CurrentTime()* $-t_{start} <$ MaxTime **do**
// Selection: traverse tree using UCB
$n_{cur} \leftarrow \text{SELECT}(\mathcal{T}, n_0)$;
$o_{cur} \leftarrow n_{cur}$.state;
// Expansion: generate candidate actions
$\mathcal{A} \leftarrow \pi(o_{cur}, q)$;
foreach $a \in \mathcal{A}$ **do**
if $a =$ stop[Answer] **then**
return $\mathcal{T}$; // task completed, terminate
else if $a =$ stop[N/A] **then**
Mark n_{cur} as terminal;
else
// State update & value estimation
$o_{next} \leftarrow w(o_{cur}, a)$; $v \leftarrow r(o_{next}, a, q)$;
$n_{new} \leftarrow \text{CreateNode}(o_{next}, a, v)$;
Add n_{new} as child of n_{cur} in $\mathcal{T}$;
if $v \geq \theta$ **then**
return $\mathcal{T}$; // high-value trajectory found
// Backpropagation: update ancestor node values
$\text{BACKPROPAGATE}(\mathcal{T}, n_{cur})$;
return $\mathcal{T}$; // timeout, return current tree

signal during the search, which may limit the size of the search tree, we terminate the expansion of the current node when this signal is issued, but do not stop the search process entirely. We visualize three examples in collection the search tree, as shown in Figure 5, Figure 6 and Figure 7.

C TextUI Datasets

As shown in Figure 4, we follow the task definitions in UI-TARS (Qin et al., 2025) and construct three core capabilities for UI understanding:

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 7.

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 instruction is shown in Table 8.

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

Post-processing of Collected Data. To improve data quality, we apply an instruction enhancement and response refinement process, as shown in Table 6. 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 9 for details.

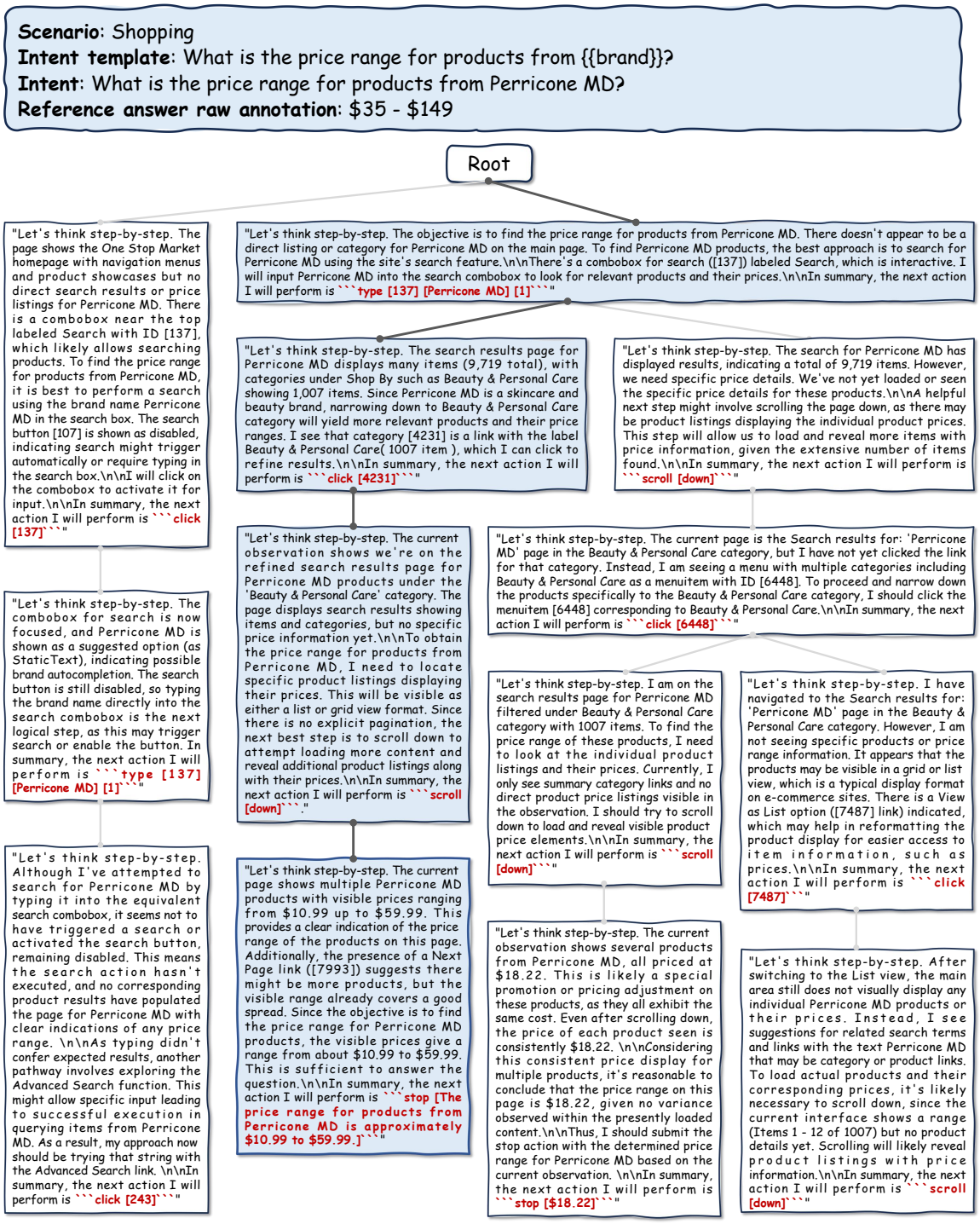


Figure 5: A case study of collected WebMCTS tree in Shopping Scenario.

Scenario: Map
Intent template: Compare the time for walking and driving route from {{start}} to {{end}}
Intent: Compare the time for walking and driving route from AMC Waterfront to Univ of Pittsburgh
Reference answer raw annotation: driving: 15min, walking: 1h 47min.

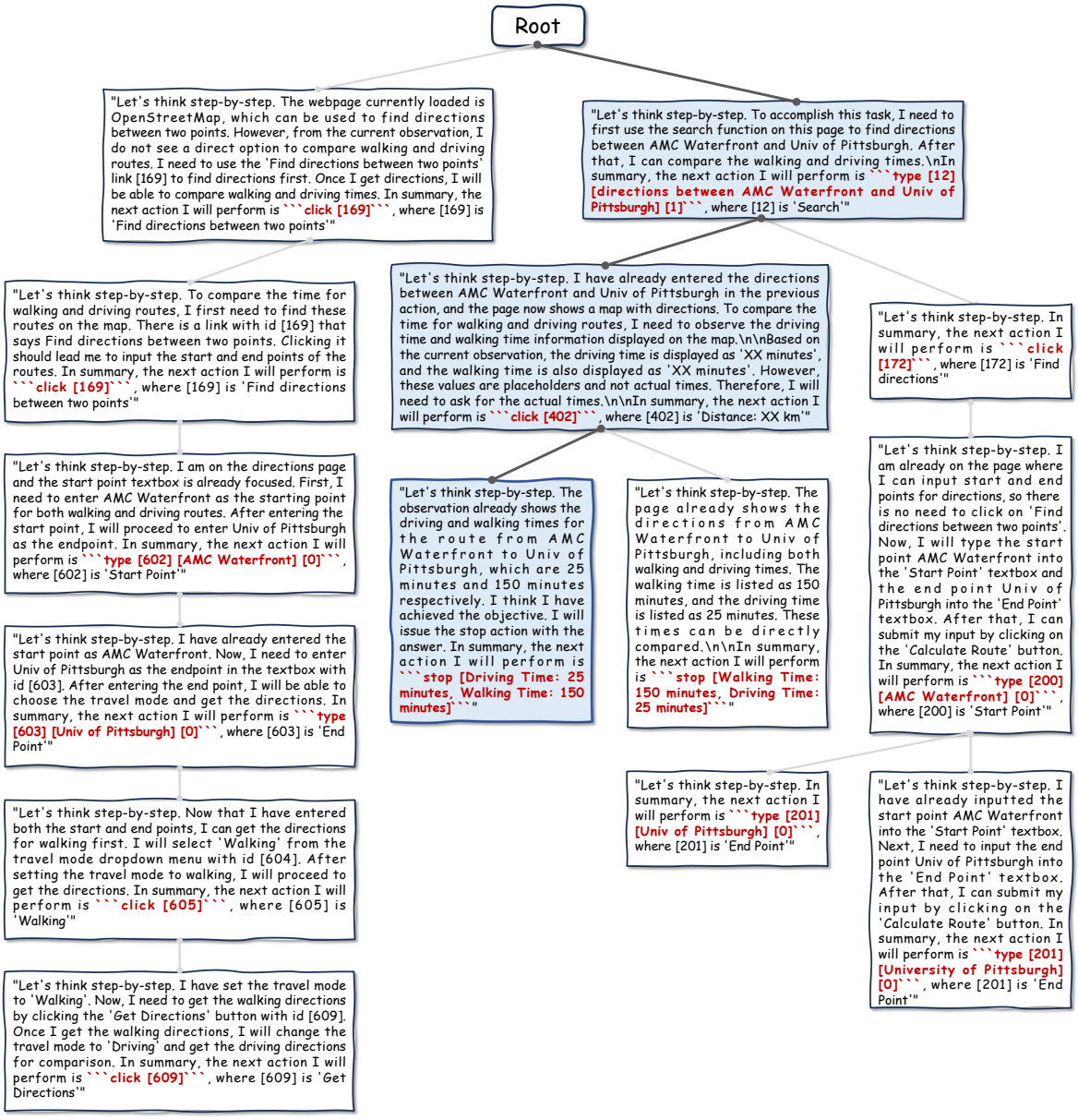


Figure 6: A case study of collected WebMCTS tree in Map Scenario.

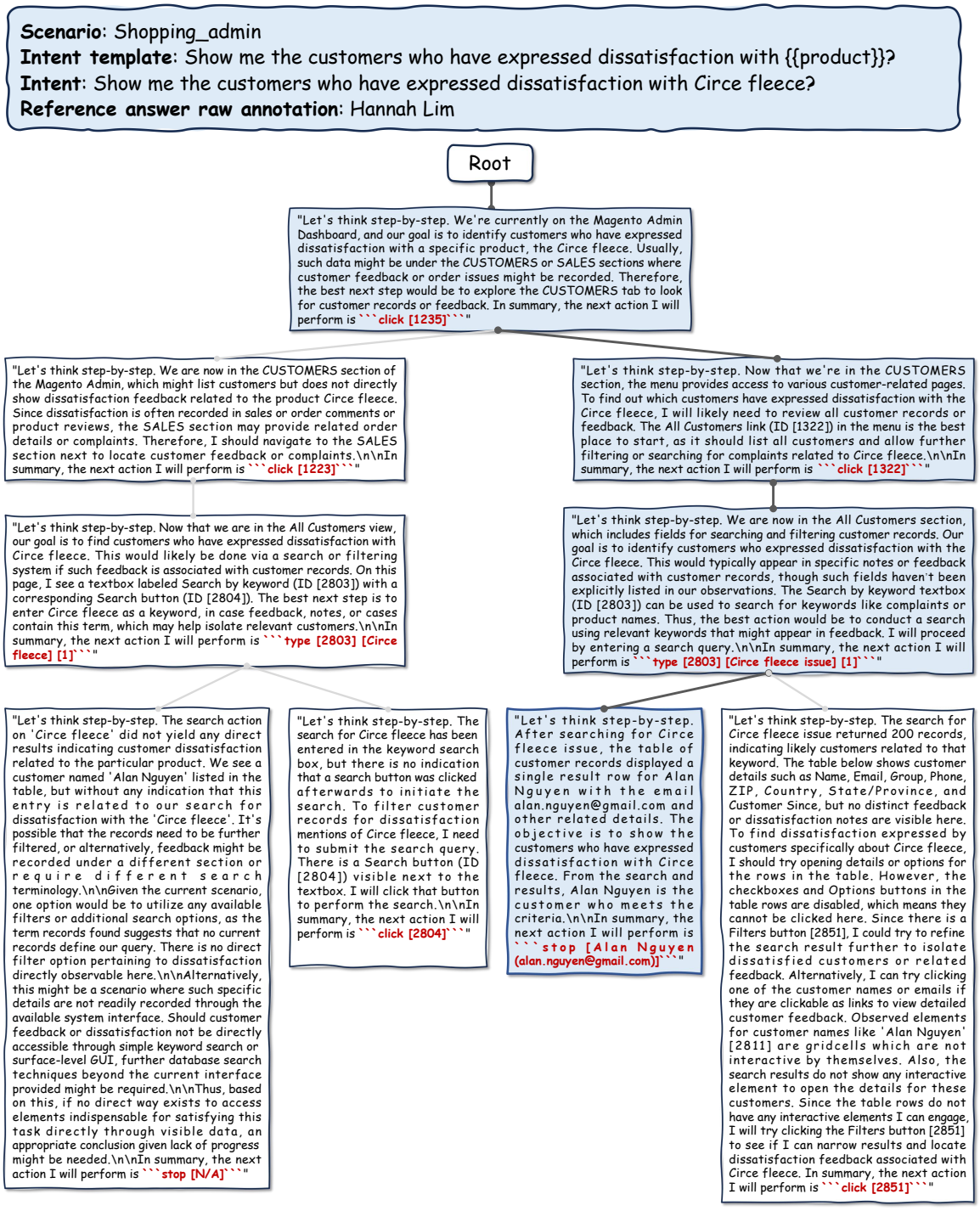


Figure 7: A case study of collected WebMCTS tree in Shopping Admin Scenario.

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 6: 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 7: 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 8: 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 9: Instruction template for {**State Transmission Prediction**}.