

WEBAGENT-R1: Training Web Agents via End-to-End Multi-Turn Reinforcement Learning

Anonymous ACL submission

Abstract

While reinforcement learning (RL) has demonstrated remarkable success in enhancing large language models (LLMs), it has primarily focused on single-turn tasks such as solving math problems. Training effective web agents for multi-turn interactions remains challenging due to the complexity of long-horizon decision-making across dynamic web interfaces. In this work, we present WEBAGENT-R1, a simple yet effective end-to-end multi-turn RL framework for training web agents. It learns directly from online interactions with web environments by asynchronously generating diverse trajectories, entirely guided by binary rewards depending on task success. Experiments on the WebArena-Lite benchmark demonstrate the effectiveness of WEBAGENT-R1, boosting the task success rate of Qwen-2.5-3B from 6.1% to 33.9% and Llama-3.1-8B from 8.5% to 44.8%, significantly outperforming existing state-of-the-art methods and strong proprietary models such as OpenAI o3. In-depth analyses reveal the effectiveness of the thinking-based prompting strategy and test-time scaling through increased interactions for web tasks. We further investigate different RL initialization policies by introducing two variants, namely WEBAGENT-R1-ZERO and WEBAGENT-R1-CoT, which highlight the importance of the warm-up training stage (*i.e.*, behavior cloning) and provide insights on incorporating long chain-of-thought (CoT) reasoning in web agents.

1 Introduction

Reinforcement learning (RL) has emerged as a promising approach for training large language models (LLMs), as exemplified by recent advances such as DeepSeek-R1 (Guo et al., 2025; Team et al., 2025; Yang et al., 2025a). However, existing works have primarily focused on single-turn, non-interactive tasks such as mathematical reasoning (Shao et al., 2024; Zeng et al.,

2025). Their effectiveness in multi-turn, interactive environments—particularly in complex scenarios requiring long-horizon decision-making and domain-specific skills, such as web browsing (Zhou et al., 2024a; He et al., 2024a; Chae et al., 2025)—still remains underexplored.

Unlike static environments, web tasks pose unique challenges for LLM agents due to their dynamic nature and diverse solution spaces. Early works on web agents primarily relied on prompting-based methods (Wang et al., 2024b; Sodhi et al., 2024; Fu et al., 2024; Zhang et al., 2025; Yang et al., 2025b) or behavior cloning (BC), which imitates demonstrated trajectories via supervised fine-tuning (Yin et al., 2024; Hong et al., 2024; Lai et al., 2024; He et al., 2024b; Putta et al., 2024). Despite their initial success, these methods lack the ability to explore diverse strategies or learn from trial and error, limiting the generalizability of web agents. To address this issue, recent works explored applying RL for better policy training. However, most of this line of research has heavily relied on offline or iterative off-policy RL solutions (Peng et al., 2019; Pan et al., 2024; Qi et al., 2025), which break the end-to-end interaction between the web agent and environment, and introduce additional complexities such as trajectory filtering (Bai et al., 2024), outcome reward model training (Qi et al., 2025), or iterative optimization procedures (Zhou et al., 2024b). These constraints hinder their practicality for real-world deployment.

Meanwhile, several concurrent works have explored end-to-end RL with on-policy updates for training LLM agents in multi-turn interactive scenarios, such as simulated games and coding environments (Wang et al., 2025; Cao et al., 2025). Unlike off-policy RL that trains on data generated by older versions of the agent, on-policy RL collects training data directly from the agent’s current behavior. This ensures that the learning process is better aligned with the agent’s most recent actions,

often leading to more stable and effective learning (Schulman et al., 2015, 2017). It also eliminates the need for additional overheads in off-policy RL (e.g., maintaining a replay buffer and filtering outdated trajectories), and enables the agent to behave adaptively based on its *own* past decisions—a key advantage in interactive environments where early decisions can significantly affect next steps.

These benefits are particularly desirable in online web environments, which often involve complex interplay between tasks due to dynamic changes of the environment. For instance, consider a situation where the agent is first tasked to log out of a user account and then to edit the user’s profile. These tasks are inherently interdependent: once the agent logs out, it loses access to the profile page. If the agent is trained using off-policy data collected from an earlier version that never logged out, it has no opportunity to learn the login behavior and may incorrectly assume continued access and generate invalid actions, ultimately leading to task failure. End-to-end RL helps avoid such pitfalls by allowing the agent to learn proper behaviors in response to environmental state changes on-the-fly.

In light of this, we propose WEBAGENT-R1, an end-to-end multi-turn RL framework for training web agents. Specifically, our design addresses several key challenges in this setting. First, at each step, the environmental observation (e.g., HTML content) can span thousands of tokens, causing the accumulated context over long horizons to incur substantial memory overheads. To mitigate this, we introduce a dynamic context compression mechanism, which adaptively adjusts the contexts across turns, ensuring scalability and preventing out-of-memory issues. Second, existing RL solutions for LLM agents are not well-suited for multi-turn scenarios. Inspired by group relative policy optimization (GRPO) (Shao et al., 2024), we extend it to multi-turn settings (M-GRPO) and employ an asynchronous trajectory rollout strategy to further improve training efficiency by generating multiple trajectories in parallel. These designs enable efficient RL training and lead to state-of-the-art performance on the WebArena-Lite benchmark, as shown in Figure 1. Extensive ablation studies and analyses further validate the effectiveness of our key design choices, revealing an effective test-time scaling strategy for multi-turn interactive web tasks, and shedding lights on understanding the roles of behavior cloning and long chain-of-thought (CoT) reasoning in RL-based web agent training.

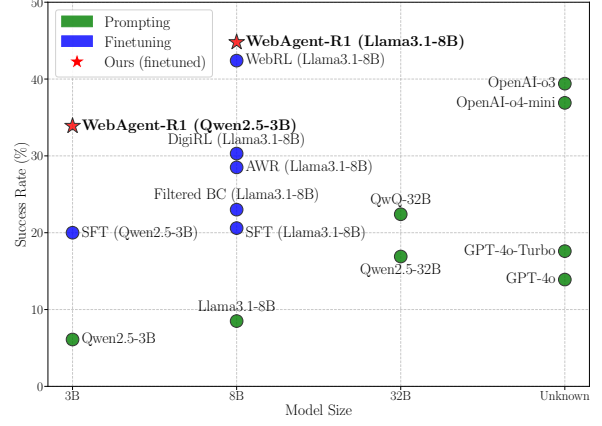


Figure 1: Comparison between existing methods and our WEBAGENT-R1 on the WebArena-Lite benchmark. Our method outperforms both strong prompting-based and finetuned baselines, achieving superior performance across various model sizes.

Our contributions are summarized as follows:

- We implement an end-to-end multi-turn RL framework for training web agents, with dynamic context compression and asynchronous trajectory rollout mechanisms to achieve training efficiency.
- Based on the proposed M-GRPO algorithm, our method substantially improves task success rates of web agents—boosting Qwen-2.5-3B from 6.1% to 33.9% and Llama-3.1-8B from 8.5% to 44.8%—surpassing previous state-of-the-art results on the WebArena-Lite benchmark.
- Extensive analyses and ablation studies underscore the crucial role of behavior cloning, validate the effectiveness of thinking-based prompting and test-time scaling strategies, and provide actionable insights on incorporating long-CoT reasoning in web agents.

2 WebAgent-R1

2.1 Problem Formulation

We formulate the web task as a Partially Observable Markov Decision Process (POMDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. At each time step t , the agent first observes a state $s_t \in \mathcal{S}$ from the environment \mathcal{E} , represented as the text-only HTML content of the current web page. Then, it generates an action a_t from a predefined action space \mathcal{A} , which includes commonly used web operations. The environment dynamics $\mathcal{T}(s_{t+1}|s_t, a_t)$ represent how the web page changes in response to actions. The

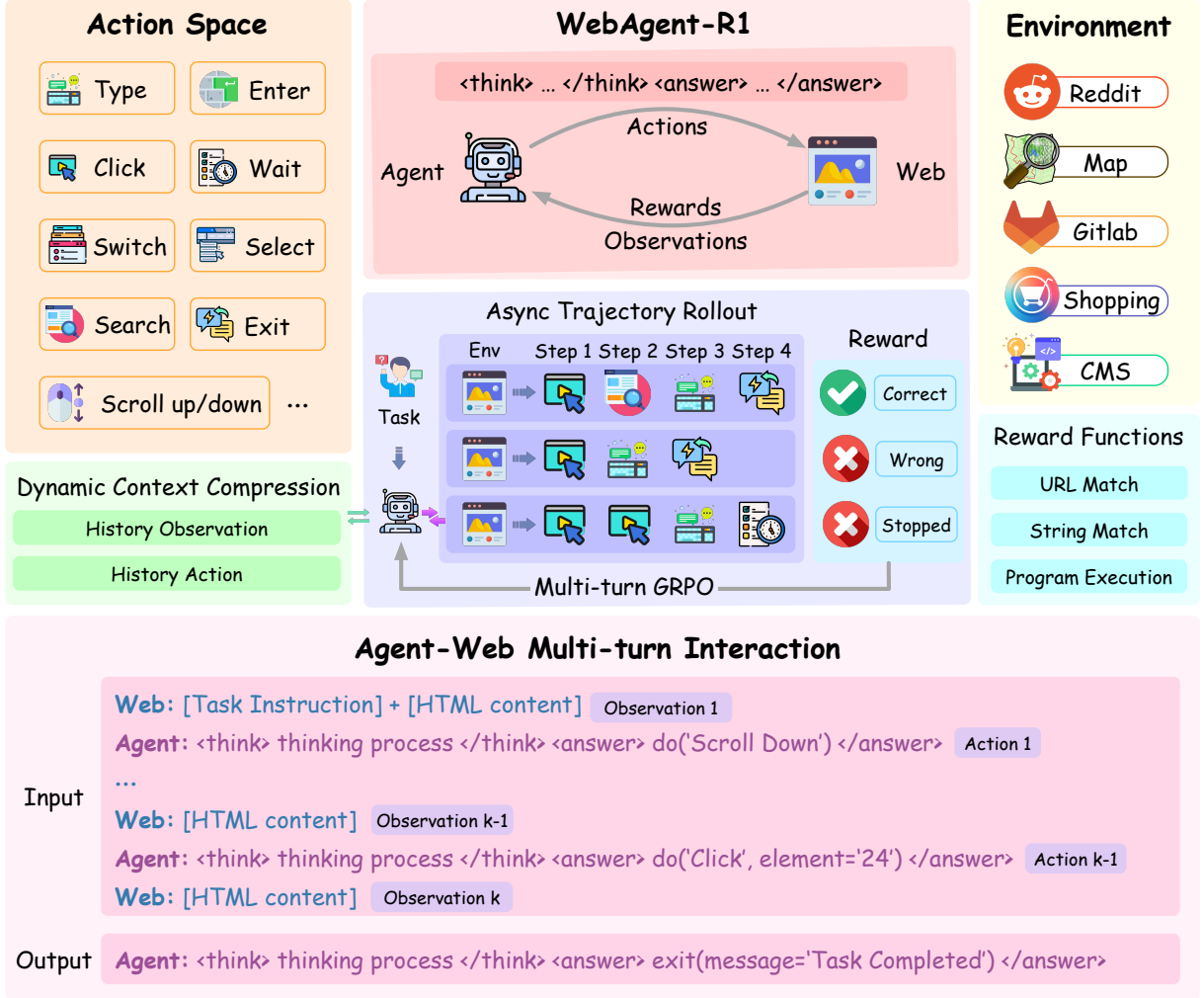


Figure 2: **(Top)**: Overview of the end-to-end multi-turn RL training framework used in WEBAGENT-R1. **(Bottom)**: An input/output example of agent–web interaction at the k -th step. The interaction continues until either the maximum number of steps is reached or the agent generates an `exit()` action to signal task completion.

agent interacts with the environment until either the task is successfully completed or the maximum number of steps is reached. At the end, the agent receives a binary outcome reward $r_t \in \{0, 1\}$ from reward functions \mathcal{R} .

Following prior work (Qi et al., 2025), we adopt WebArena (Zhou et al., 2024a) as the web environment over other simulated or static environments such as WebShop (Yao et al., 2022) or Mind2Web (Deng et al., 2023) for greater practicality—It provides a realistic, self-hostable environment for web agents, along with rule-based rubrics that automatically check for indicators of success in the final state (e.g., confirmation messages or expected content on the page). Note that some prior works (Liu et al., 2025; He et al., 2024a) incorporate web page screenshots as additional visual inputs, whereas our work focuses solely on text-based decision-making over HTML. Other efforts, such as Yang et al. (2025b), explore opti-

mizing the action space or prompt design without model fine-tuning. These directions are orthogonal to our investigated problem and can be conceptually integrated with our method as future work.

2.2 Behavior Cloning

To initialize the web agent, we first apply behavior cloning (BC) using a fixed dataset of expert demonstrations $\mathcal{D} = \{(h_t, a_t)\}$, where h_t denotes the full interaction history up to time step t , defined as $h_t = (s_1, a_1, s_2, a_2, \dots, s_t)$. The policy π_θ is trained via supervised fine-tuning (SFT) to imitate expert actions conditioned on this history:

$$\mathcal{L}_{BC} = -\mathbb{E}_{(h_t, a_t) \sim \mathcal{D}} [\log \pi_\theta(a_t | h_t)]$$

This warm-up stage enables the agent to acquire basic web interaction skills defined in the action space. As indicated in our ablation study (§ 3.4), this BC-trained policy provides a crucial foundation for subsequent reinforcement learning optimization.

Table 1: Comparison of different methods for training web agents. *Trial-and-Error* indicates whether the method supports learning through interactions with the environment (i.e., reinforcement learning). *On-Policy* denotes whether the training data is collected from the current policy. *Replay Buffer Free* indicates methods that do not require selectively sampling trajectories from a replay buffer, a complexity common in off-policy RL. *Self-Sufficient* means no external training signals required (e.g., WebRL trains an additional outcome reward model to label new data generated by GPT-4). As shown, our method is the only one that enables end-to-end RL with on-policy updates while avoiding additional complexities such as maintaining a replay buffer and being free from external supervision.

Method	Trial-and-Error	On-Policy	Replay Buffer Free	Self-Sufficient
Behavior Cloning (SFT)	✗	✗	✓	✓
AWR (Peng et al., 2019)	✗	✗	✗	✓
DigiRL (Bai et al., 2024)	✓	✗	✗	✓
WebRL (Qi et al., 2025)	✓	✗	✗	✗
WEBAGENT-R1	✓	✓	✓	✓

2.3 End-to-End Multi-Turn Reinforcement Learning

As illustrated in Figure 2, our end-to-end multi-turn RL framework trains web agents through on-line interactions guided by rule-based outcome rewards. To enable efficient and scalable training, we implemented two key mechanisms: *dynamic context compression* to reduce memory overhead, and *asynchronous trajectory rollout* to improve sampling efficiency. Based on the BC-trained policy, we further fine-tune the agent using an extension of GRPO (Qi et al., 2025) in the multi-turn settings, termed *M-GRPO*. Our implementation can be viewed as a minimalist approach that supports efficient multi-turn RL training while maintaining generality, with potential for future extensions (e.g., incorporating fine-grained reward shaping mechanisms for intermediate steps).

Dynamic Context Compression In web tasks, each observation s_t often contains thousands of tokens. Across multi-turn interactions, the accumulated context grows rapidly, leading to excessive memory usage and potential out-of-memory issues, making training impractical. To address this, we propose a dynamic context compression strategy. As new observations arrive, earlier ones are simplified to reduce the context length while preserving the complete action history. Let the interaction history at step t be $h_t = (s'_1, a_1, s'_2, a_2, \dots, s_t)$, where each s'_i is a simplified template (e.g., “Simplified HTML”) representing prior observations. When the agent executes an action a_t and receives a new observation s_{t+1} , the updated history becomes $h_{t+1} = (s'_1, a_1, s'_2, a_2, \dots, s'_t, a_t, s_{t+1})$, where s_t is replaced by its simplified version s'_t . This allows the agent to maintain a compact yet informative context of past interactions. Since the

context evolves dynamically, we also update the loss masks accordingly to ensure that the loss is correctly computed only on the action tokens during the M-GRPO optimization.

Multi-turn GRPO Inspired by GRPO, we extend its standard form to multi-turn RL settings and introduce multi-turn group relative policy optimization (M-GRPO). Specifically, for each task q , we first sample a group of trajectories $\{\tau_1, \tau_2, \dots, \tau_G\}$ and then optimize the policy model π_θ by minimizing the following loss:

$$\mathcal{L}_{\text{M-GRPO}}(\theta) = -\frac{1}{G} \sum_{i=1}^G \frac{1}{|\tau_i|} \sum_{j=1}^{|\tau_i|} \left(\frac{1}{|a_{i,j}|} \sum_{t=1}^{|a_{i,j}|} [\tilde{A}_{i,j,t} - \beta \mathbb{D}_{\text{KL}}(\theta)] \right)$$

where $\tau_i = \{a_{i,1}, a_{i,2}, \dots, a_{i,|\tau_i|}\}$ is the sequence of generated actions in the i -th trajectory, $\tilde{A}_{i,j,t} = \min\{r_{i,j,t}(\theta)A_{i,j}, \text{clip}(r_{i,j,t}(\theta), 1-\epsilon, 1+\epsilon)A_{i,j}\}$ is the advantage for the t -th token in action $a_{i,j}$ of trajectory τ_i , $r_{i,j,t}(\theta) = \frac{\pi_\theta(a_{i,j,t}|q, a_{i,j,<t})}{\pi_{\text{old}}(a_{i,j,t}|q, a_{i,j,<t})}$ denotes the importance sampling term, ϵ and β are hyperparameters, and $A_{i,j} = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$ is the group relative advantage, computed using a group of rewards $\mathbf{r} = \{r_1, r_2, \dots, r_G\}$ produced by rule-based reward functions.

Asynchronous Trajectory Rollout Generating a group of trajectories requires repeated interaction with the environment and can be time-consuming. To address this, we introduce an asynchronous trajectory rollout strategy, where multiple independent browser instances $\{\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_G\}$ are instantiated, each maintaining its own context (e.g., cookies). For each task, all instances are initialized with the same starting page, but the agent interacts with them independently, resulting in diverse histories and trajectories. This asynchronous design enables efficient trajectory generation in M-GRPO.

Table 2: Task success rate (SR) comparison across different methods on various websites in WebArena-Lite (Liu et al., 2025; Qi et al., 2025; Zhou et al., 2024a). Baseline performance is reported as the higher value between our reproduced results and those reported in the literature (Qi et al., 2025). The best scores are highlighted in bold.

Method	Reddit	GitLab	CMS	Map	Shopping	Average SR
<i>Prompting Method</i>						
General Model						
Qwen2.5-3B	5.3	13.3	5.7	0	4.4	6.1
Llama3.1-8B	5.3	10.0	5.7	15.4	8.9	8.5
Qwen2.5-32B	10.5	20.0	20.0	19.2	17.8	16.9
GPT-4o	10.5	10.0	20.0	20.0	11.1	13.9
GPT-4o-Turbo	10.5	16.7	14.3	36.7	13.3	17.6
Reasoning Model						
QwQ-32B	15.8	33.3	25.7	15.4	20.0	22.4
OpenAI-o3	36.8	46.7	45.7	38.5	33.3	39.4
OpenAI-o4-mini	47.4	43.3	45.7	26.9	28.9	36.9
<i>Finetuning Method</i>						
Qwen2.5-3B						
Behavior Cloning	42.1	16.7	22.9	26.9	11.1	20.0
WEBAGENT-R1	26.3	53.3	48.6	26.9	24.4	33.9
Llama3.1-8B						
Behavior Cloning	36.8	6.7	20.0	33.3	17.8	20.6
Filtered BC (Pan et al., 2024)	52.6	20.0	31.4	23.3	8.9	23.0
AWR (Peng et al., 2019)	57.9	26.7	31.4	26.7	17.8	28.5
DigiRL (Bai et al., 2024)	57.9	26.7	37.1	33.3	17.8	30.3
WebRL (Qi et al., 2025)	63.2	46.7	54.3	36.7	31.1	42.4
WEBAGENT-R1	47.4	56.7	57.1	23.1	44.4	44.8

Reward Design We use the default rule-based reward functions in the web environment, which assign binary rewards ($r=1$ for success, $r=0$ otherwise) based on task-specific criteria (e.g., reaching a target page). This eliminates the need for outcome reward models (Qi et al., 2025), ensuring a simple and generalizable training setup.

3 Experiments

3.1 Experimental Setup

Web Environment Like prior works (Liu et al., 2025; Qi et al., 2025), we focus on web agents for real-world scenarios, specifically utilizing WebArena (Zhou et al., 2024a), a self-hostable and realistic web environment that supports practical tasks across diverse domains: social forums (Reddit), collaborative coding (GitLab), e-commerce content management systems (CMS), open street maps (Map), and online shopping (Shopping).

Dataset and Evaluation Metrics Following Qi et al. (2025), we use the public 9,460 trajectories for behavior cloning, and adopt WebArena-Lite, a human-verified version of WebArena, for more reliable evaluation. Specifically, we use 165 verified tasks for evaluation and 647 remaining tasks for RL training. Task success rate is calculated using the built-in rule-based rubrics.

Baselines For prompting baselines, we provide a comprehensive comparison with both open-source and proprietary models, including general-purpose models (e.g., Qwen2.5, Llama3.1, GPT-4) and reasoning-specialized models (e.g., QwQ, OpenAI o3 (OpenAI, 2025)), covering various model sizes. For finetuning methods, we employ Qwen2.5-3B and Llama3.1-8B as the backbone model.

More details on the environment and implementation are provided in Appendix A and B. We also provide the prompt templates and qualitative examples in Appendix D and E.

3.2 Main Results

Most LLMs still struggle with web tasks through prompting, highlighting the importance of finetuning for web agents. As shown in Table 2, our experiments reveal the limitations of off-the-shelf models in web tasks. Despite their strong general capabilities, state-of-the-art models such as OpenAI’s o3 achieve only a 39.4% success rate (SR). In contrast, a finetuned 3B model trained with simple behavior cloning achieves a success rate of 20%, outperforming proprietary models like GPT-4o. We speculate that the poor performance of off-the-shelf models is not due to base model size or capability, but rather to insufficient understanding of HTML structure and web-specific behaviors,

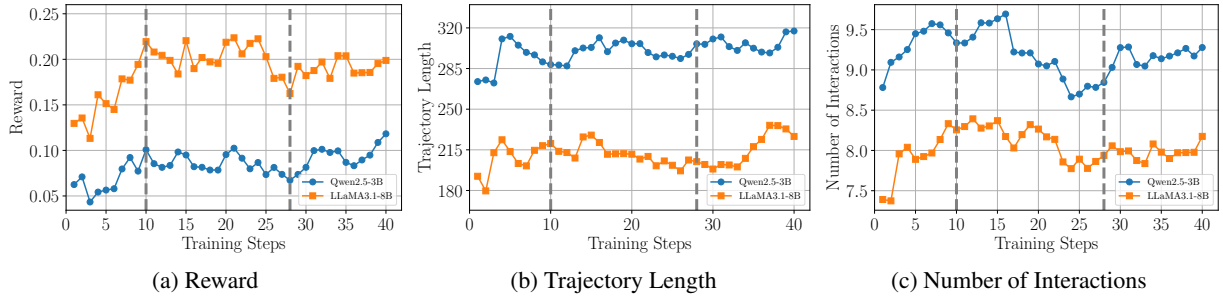


Figure 3: Training dynamics during RL, including rewards, trajectory length, and number of interactions. As indicated by the dashed vertical lines in the figure, the entire process can be broadly divided into three phases: (1) initial skill acquisition, (2) exploration for policy refinement, and (3) final policy stabilization.

as evidenced by the observation that both 3B and 8B models achieve comparable performance after behavior cloning. These findings emphasize the necessity of domain-specific training on web data to develop effective LLM-based web agents.

Reasoning models are better web agents. Compared to general-purpose LLMs, models equipped with explicit thinking capabilities perform significantly better on web tasks, likely due to their ability to decompose high-level goals and explicitly lay out dynamic changes in the web interface. This gap underscores the importance of thinking in web environments, which typically require multi-turn decision-making and dynamic contextual understanding. Motivated by this observation, we further explore the integration of thinking mechanisms into web agents through prompt design (§ 3.5) and training strategies (§ 3.4), which further confirms the advantage of thinking ability for web agents.

Reinforcement learning enables stronger performance for web agents. While behavior cloning via SFT can significantly improve LLM’s performance as web agents (e.g., boosting Qwen2.5-3B from 6.1% to 20%), applying RL on top of the SFT-trained policy leads to additional substantial gains (e.g., further boosting Qwen2.5-3B from 20% to 33.9%). We attribute these improvements to RL’s ability to optimize long-horizon decision-making, explore novel strategies beyond those seen in the SFT data through trial-and-error across dynamic web interactions. While prior RL solutions for web agents, such as DigiRL and WebRL, have also shown performance gains, our method achieves even stronger results, highlighting the effectiveness of our end-to-end multi-turn RL framework.

3.3 Training Dynamics

To understand how the proposed end-to-end reinforcement learning optimizes the behavior of

the web agents, we analyze the training dynamics across three metrics: reward, trajectory length (i.e., the number of tokens in model responses across all multi-turn interactions), and number of interactions. As shown in Figure 3, the learning process can be broadly divided into three distinct phases, separated by vertical dashed lines.

Reward. Phase 1 shows a rapid increase in reward, indicating that the agent quickly learns basic skills and begins to succeed on simpler tasks. In Phase 2, the reward growth plateaus and slightly fluctuates, suggesting that the agent is exploring different strategies and refining its policy. In Phase 3, reward gradually improves again, indicating exploitation and increased stability.

Trajectory Length. Trajectory length increases sharply during Phase 1, then stabilizes in Phase 2. In Phase 3, a modest increase is observed again. This trend suggests that the agent initially learns to produce more detailed outputs, followed by a period of consolidation and later refinement to balance verbosity with task effectiveness.

Number of Interactions. The number of interaction rounds increases during Phase 1 as the agent becomes more proactive, followed by a reduction in Phase 2 as it learns to interact more efficiently. In Phase 3, the interaction count stabilizes, indicating convergence toward a more consistent and effective interaction strategy.

These trends highlight a three-phase learning dynamic commonly observed in RL: (1) initial skill acquisition, (2) exploration for policy refinement, and (3) final policy stabilization. Interestingly, both Qwen2.5-3B and LLaMA3.1-8B follow similar learning patterns, suggesting that our end-to-end multi-turn RL framework effectively scales across model sizes and enables stable policy improvement.

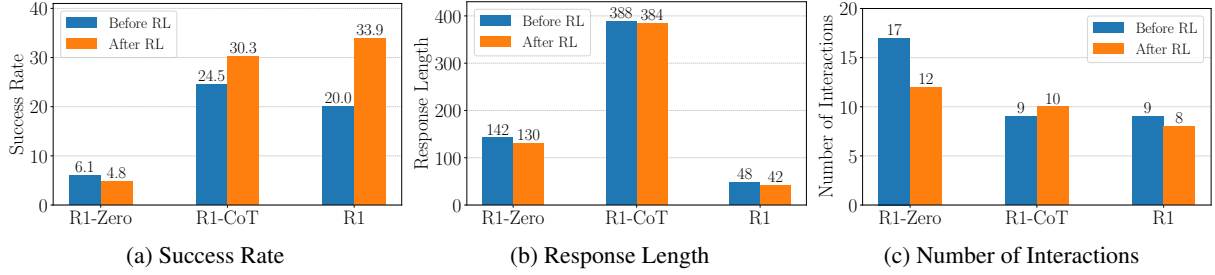


Figure 4: Ablation study on RL initialization policy by comparing WEBAGENT-R1 (R1) with two variants: WEBAGENT-R1-ZERO (R1-Zero), initialized from an off-the-shelf model without SFT, and WEBAGENT-R1-CoT (R1-CoT), initialized from an SFT model trained with long chain-of-thought (CoT) data during behavior cloning. The comparison includes task success rate, single-turn response length, and number of interactions, evaluated both before and after applying RL.

3.4 Ablation Study

To validate key design choices in our framework, we conduct a set of ablation studies using Qwen2.5-3B as the backbone model. Specifically, we introduce two variants, WEBAGENT-R1-ZERO and WEBAGENT-R1-CoT, to study the impact of behavior cloning and long CoT for web agents. The results are presented in Figure 4.

Behavior cloning is crucial for training web agents with RL. WEBAGENT-R1-ZERO skips the behavior cloning stage and starts RL directly from an off-the-shelf model, with an initial success rate of only 6.1%. Surprisingly, the model’s performance even deteriorates slightly after RL. We hypothesize that this is due to the lack of knowledge about web tasks since the model tends to produce incomplete or ill-formed actions (e.g., missing required arguments) and rarely obtains positive rewards during RL. This severely hampers effective exploration and learning, highlighting that behavior cloning is essential for initializing web agents and enabling successful subsequent RL.

Incorporating long-CoT data into behavior cloning leads to more performant web agents. We first augment the behavior cloning (BC) data by generating long-CoT traces using a strong reasoning model (see Appendix C for details), and then apply SFT to obtain a *long-CoT SFT* model (i.e., the WEBAGENT-R1-CoT variant before RL). Compared to the SFT model trained on standard BC data, the long-CoT SFT model achieves a much higher task success rate (24.5% vs. 20%), demonstrating the effectiveness of long-CoT reasoning for web agents.

Limited gains from RL for long-CoT SFT model. While RL shows promising improvements for both the vanilla SFT and long-CoT SFT models, it is

Table 3: Analysis of prompting design. We report the average success rate (SR), single-turn response length, and number of interactions. The result reveals a novel test-time scaling paradigm by increasing the number of interactions for multi-turn interactive web tasks.

Method	SR	Length	# of Interactions
W/o thinking format			
Qwen2.5-3B	3.2	139	6
Llama3.1-8B	4.8	43	7
o4-mini	15.9	56	5
With thinking format			
Qwen2.5-3B	6.1	142	17
Llama3.1-8B	8.5	39	11
o4-mini	36.9	57	10

interesting that the gain is notably smaller for the latter. Specifically, WEBAGENT-R1 improves from 20% to 33.9%, whereas WEBAGENT-R1-CoT improves from 24.5% to only 30.3%. We hypothesize that this is because the deterministic reasoning patterns learned during long-CoT BC may constrain the model’s exploration space during RL, limiting its ability to discover novel strategies compared to standard SFT models with more flexible exploratory behaviors.

3.5 Analysis

Prompting with thinking format unleashes the potential of LLMs as web agents. As shown in Table 3, using the thinking format significantly improves task success rates across models, particularly for stronger ones (e.g., o4-mini improves from 15.9% to 36.9%). Interestingly, while the average single-turn response length remains similar (e.g., 139 → 142 tokens for Qwen2.5-3B), the number of interactions increases substantially (e.g., 6 → 17) with the thinking format. This observation suggests a novel test-time scaling strategy for web tasks—rather than producing longer single-turn responses, the web agent can become more effective by engaging in deeper multi-turn interactions.

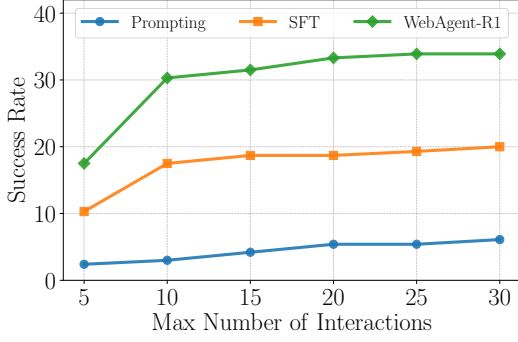


Figure 5: Analysis of test-time scaling with increased max number of interactions. Allowing more interactions enables the web agent to produce longer trajectories and consistently improves the success rate.

Test-time scaling through increased interactions leads to better performance on web tasks.

Building on the above finding, we further investigate how increasing the number of interactions between the web agent and the environment affects performance. As shown in Figure 5, allowing more interaction turns consistently improves success rates across prompting-based, SFT, and RL-based methods. We hypothesize that this form of test-time scaling facilitates deeper exploration and yields longer trajectories, potentially enabling the agent to iteratively refine its actions and make more informed decisions through extended interactions.

4 Related Works

4.1 LLM-based Agents

LLMs have demonstrated promising agentic capabilities, such as breaking down complex tasks into manageable subgoals and reasoning over long horizons (Zhou et al., 2022; Huang et al., 2022; Madaan et al., 2022; Li et al., 2023a,b; Wu et al., 2024; Chu et al., 2025). Building on these capabilities, LLM-based agents have been applied to a variety of real-world interactive tasks, including web navigation (Nakano et al., 2021; Yao et al., 2022; Ma et al., 2023; Gur et al., 2024; Abuelsaad et al., 2024; Lutz et al., 2024; Patel et al., 2024; Putta et al., 2024), general computer use (Li et al., 2020; Deng et al., 2023; Yang et al., 2024), and embodied environments (Puig et al., 2018; Shridhar et al., 2020; Toyama et al., 2021; Fan et al., 2022; Huang et al., 2022). Specifically, our work focuses on text-based web agents that operate in browser-based environments purely based on HTML content, which requires agentic capabilities such as tool use, memory, and decision-making under partial observability (Zhou et al., 2024a; Qi et al.,

2025). Complementary to this line of work, GUI agents leverage additional multimodal inputs such as screenshots, enabling visual-guided interactions with the environment (Lee et al., 2023; Shaw et al., 2023; Zheng et al., 2024; He et al., 2024a,b; Koh et al., 2024; Kil et al., 2024; Lei et al., 2025; Liu et al., 2025). For a comprehensive overview, we refer readers to recent surveys (Wang et al., 2024a; Hu et al., 2025; Ning et al., 2025).

4.2 Reinforcement Learning for LLMs

Recent advances like DeepSeek-R1 (Guo et al., 2025) highlight the strong potential of RL in enhancing LLMs. However, most prior work focuses on single-turn tasks such as math problems (Shao et al., 2024), with limited exploration in multi-turn settings (Zhou et al., 2024b, 2025). Recent efforts have made some progress in this direction, such as training LLM agents to repeatedly use search engines (Jin et al., 2025; Sun et al., 2025; Chen et al., 2025; Song et al., 2025), but typically constrain actions to simple API calls without real environment interaction. A few concurrent works, such as RAGEN (Wang et al., 2025) and SkyRL (Cao et al., 2025), have applied RL to more dynamic settings like simulated games and coding environments (Jimenez et al., 2024). However, real-world web environments remain largely underexplored. Our work fills this gap by providing a practical framework and offering actionable insights for training web agents with end-to-end RL.

5 Conclusion

This work introduces WEBAGENT-R1, an end-to-end multi-turn RL framework for training web agents. We extend the standard GRPO to multi-turn settings, termed M-GRPO, and implement dynamic context compression and asynchronous trajectory rollout mechanisms for efficient training. Empirically, WEBAGENT-R1 achieves new state-of-the-art results on the WebArena-Lite benchmark. Our findings underscore the critical role of behavior cloning in initializing web agents, providing a strong foundation for effective RL. We further analyze training dynamics and explore the effects of thinking-based prompting and test-time scaling strategies, showing that increasing interaction depth consistently enhances web agents. Future work includes exploring multi-modal inputs and extending our approach to broader GUI-based tasks beyond web environments, such as computer use.

Limitations and Potential Risks

Despite the effectiveness of WEBAGENT-R1, our current approach has several limitations that suggest directions for future work. First, we consider only textual input for the web tasks. Incorporating additional visual input (e.g., screenshots) may enhance performance since visual information, such as layout and colors, can be helpful for effective navigation and decision-making. Second, our method relies on rule-based outcome rewards to guide RL training. While effective in our setting, such reward functions may not be readily available in other interactive scenarios, such as open-ended travel planner agents, where task goals are ambiguous and no clear reference or verifiable outcome is available. Lastly, like existing web agents, our model is trained with a fixed set of predefined actions (e.g., click, type), which can limit its flexibility when encountering interactive elements that require unseen operations. Enabling dynamic adaptation to new operations remains an open challenge for web agents.

In terms of potential risks, such agents should be used with caution when deployed in real-world environments, especially those involving administrative privileges. For example, when interacting with content management systems (CMS) in a production environment, the agent may inadvertently perform destructive actions, such as modifying or deleting sensitive business data. To ensure safe deployment, future work should incorporate permission controls, verification prompts, and safeguards to prevent high-impact or irreversible actions.

References

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. 2024. Agent-E: From autonomous web navigation to foundational design principles in agentic systems. *arXiv preprint arXiv:2407.13032*.

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. DigIRL: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495.

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhmaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025.

Skyrl-v0: Train real-world long-horizon agents via reinforcement learning.

Hyungjoo Chae, Namyoung Kim, Kai Tzu iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2025. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *The Thirteenth International Conference on Learning Representations*.

Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025. ReSearch: Learning to reason with search for llms via reinforcement learning.

Zhendong Chu, Shen Wang, Jian Xie, Tinghui Zhu, Yibo Yan, Jinheng Ye, Aoxiao Zhong, Xuming Hu, Jing Liang, Philip S Yu, and 1 others. 2025. LLM agents for education: Advances and applications. *arXiv preprint arXiv:2503.11733*.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*.

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

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and

657	Dong Yu. 2024a. Webvoyager: Building an end-to-end web agent with large multimodal models. In <i>Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 6864–6890.	713
658		714
659		715
660		716
661		
662	Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2024b. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. <i>arXiv preprint arXiv:2410.19609</i> .	717
663		718
664		719
665		720
666		721
667		722
668		723
669	Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In <i>Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition</i> , pages 14281–14290.	724
670		725
671		726
672		727
673		728
674	Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, and 1 others. 2025. Os agents: A survey on mllm-based agents for computer, phone and browser use. In <i>Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics</i> .	729
675		730
676		
677		731
678		732
679		733
680		734
681	Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In <i>International conference on machine learning</i> , pages 9118–9147. PMLR.	735
682		736
683		737
684		738
685		739
686	Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In <i>The Twelfth International Conference on Learning Representations</i> .	740
687		741
688		
689		742
690		743
691		744
692	Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. <i>arXiv preprint arXiv:2503.09516</i> .	745
693		746
694		
695		747
696		748
697	Jihyung Kil, Chan Hee Song, Boyuan Zheng, Xiang Deng, Yu Su, and Wei-Lun Chao. 2024. Dual-view visual contextualization for web navigation. In <i>Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition</i> , pages 14445–14454.	749
698		750
699		751
700		
701		752
702	Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In <i>Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 881–905.	753
703		754
704		755
705		756
706		757
707		758
708		759
709		760
710	Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient	761
711		762
712		763
	memory management for large language model serving with pagedattention. In <i>Proceedings of the 29th Symposium on Operating Systems Principles</i> , pages 611–626.	764
		765
	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 <i>Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining</i> , pages 5295–5306.	766
		767
		768
		769
	Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvasi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screen-shot parsing as pretraining for visual language understanding. In <i>International Conference on Machine Learning</i> , pages 18893–18912. PMLR.	770
		771
	Xuanyu Lei, Zonghan Yang, Xinrui Chen, Peng Li, and Yang Liu. 2025. Scaffolding coordinates to promote vision-language coordination in large multimodal models. In <i>Proceedings of the 31st International Conference on Computational Linguistics</i> , pages 2886–2903.	772
		773
		774
		775
		776
		777
		778
		779
		780
		781
		782
		783
		784
		785
		786
		787
		788
		789
		790
		791
		792
		793
		794
		795
		796
		797
		798
		799
		800
		801
		802
		803
		804
		805
		806
		807
		808
		809
		810
		811
		812
		813
		814
		815
		816
		817
		818
		819
		820
		821
		822
		823
		824
		825
		826
		827
		828
		829
		830
		831
		832
		833
		834
		835
		836
		837
		838
		839
		840
		841
		842
		843
		844
		845
		846
		847
		848
		849
		850
		851
		852
		853
		854
		855
		856
		857
		858
		859
		860
		861
		862
		863
		864
		865
		866
		867
		868
		869
		870
		871
		872
		873
		874
		875
		876
		877
		878
		879
		880
		881
		882
		883
		884
		885
		886
		887
		888
		889
		890
		891
		892
		893
		894
		895
		896
		897
		898
		899
		900
		901
		902
		903
		904
		905
		906
		907
		908
		909
		910
		911
		912
		913
		914
		915
		916
		917
		918
		919
		920
		921
		922
		923
		924
		925
		926
		927
		928
		929
		930
		931
		932
		933
		934
		935
		936
		937
		938
		939
		940
		941
		942
		943
		944
		945
		946
		947
		948
		949
		950
		951
		952
		953
		954
		955
		956
		957
		958
		959
		960
		961
		962
		963
		964
		965
		966
		967
		968
		969
		970
		971
		972
		973
		974
		975
		976
		977
		978
		979
		980
		981
		982
		983
		984
		985
		986
		987
		988
		989
		990
		991
		992
		993
		994
		995
		996
		997
		998
		999
		1000

882	Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue	Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and	939
883	Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu,	Yu Su. 2024. GPT-4v(ision) is a generalist web agent,	940
884	Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, and 1	if grounded. In <i>Forty-first International Conference</i>	941
885	others. 2025. Ragen: Understanding self-evolution	<i>on Machine Learning</i> .	942
886	in llm agents via multi-turn reinforcement learning.		
887	<i>arXiv preprint arXiv:2504.20073</i> .	Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou,	943
		Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue	944
888	Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and	Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Gra-	945
889	Graham Neubig. 2024b. Agent workflow memory.	ham Neubig. 2024a. Webarena: A realistic web en-	946
890	<i>arXiv preprint arXiv:2409.07429</i> .	vironment for building autonomous agents. In <i>The</i>	947
		<i>Twelfth International Conference on Learning Repre-</i>	948
891	Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu,	<i>sentations</i> .	949
892	Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang,		
893	Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah,	Shuyan Zhou, Li Zhang, Yue Yang, Qing Lyu,	950
894	Ryen W White, Doug Burger, and Chi Wang. 2024.	Pengcheng Yin, Chris Callison-Burch, and Graham	951
895	Autogen: Enabling next-gen LLM applications via	Neubig. 2022. Show me more details: Discovering	952
896	multi-agent conversations. In <i>First Conference on</i>	hierarchies of procedures from semi-structured web	953
897	<i>Language Modeling</i> .	data. <i>arXiv preprint arXiv:2203.07264</i> .	954
898	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,	Yifei Zhou, Song Jiang, Yuandong Tian, Jason We-	955
899	Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao,	ston, Sergey Levine, Sainbayar Sukhbaatar, and Xian	956
900	Chengen Huang, Chenxu Lv, Chujie Zheng, Dayi-	Li. 2025. Sweet-rl: Training multi-turn llm agents	957
901	heng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge,	on collaborative reasoning tasks. <i>arXiv preprint</i>	958
902	Haoran Wei, Huan Lin, Jialong Tang, and 41 others.	<i>arXiv:2503.15478</i> .	959
903	2025a. Qwen3 technical report.		
		Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine,	960
904	John Yang, Carlos E Jimenez, Alexander Wettig, Kilian	and Aviral Kumar. 2024b. Archer: Training language	961
905	Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir	model agents via hierarchical multi-turn rl. In <i>Inter-</i>	962
906	Press. 2024. Swe-agent: Agent-computer interfaces	<i>national Conference on Machine Learning</i> , pages	963
907	enable automated software engineering. <i>Advances in</i>	62178–62209. PMLR.	964
908	<i>Neural Information Processing Systems</i> , 37:50528–		
909	50652.		
910	Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor,		
911	Pratik Chaudhari, George Karypis, and Huzefa Rang-		
912	wala. 2025b. AgentOccam: A simple yet strong base-		
913	line for LLM-based web agents. In <i>The Thirteenth</i>		
914	<i>International Conference on Learning Representa-</i>		
915	<i>tions</i> .		
916	Shunyu Yao, Howard Chen, John Yang, and Karthik		
917	Narasimhan. 2022. Webshop: Towards scalable real-		
918	world web interaction with grounded language agents.		
919	<i>Advances in Neural Information Processing Systems</i> ,		
920	35:20744–20757.		
921	Da Yin, Faeze Brahman, Abhilasha Ravichander, Khy-		
922	athi Chandu, Kai-Wei Chang, Yejin Choi, and		
923	Bill Yuchen Lin. 2024. Agent lumos: Unified and		
924	modular training for open-source language agents.		
925	In <i>Proceedings of the 62nd Annual Meeting of the</i>		
926	<i>Association for Computational Linguistics (Volume</i>		
927	<i>1: Long Papers)</i> , pages 12380–12403.		
928	Weihaio Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Ke-		
929	qing He, Zejun Ma, and Junxian He. 2025. Simplertl-		
930	zoo: Investigating and taming zero reinforcement		
931	learning for open base models in the wild. <i>arXiv</i>		
932	<i>preprint arXiv:2503.18892</i> .		
933	Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu,		
934	and Volker Tresp. 2025. Webpilot: A versatile and		
935	autonomous multi-agent system for web task execu-		
936	tion with strategic exploration. In <i>Proceedings of</i>		
937	<i>the AAAI Conference on Artificial Intelligence</i> , vol-		
938	ume 39, pages 23378–23386.		

A Web Environment

WebArena-Lite WebArena (Zhou et al., 2024a) is a realistic, self-hostable web environment for developing LLM-based agents. It comprises 812 real-world web tasks spanning diverse domains, including social forum (Reddit), collaborative coding (GitLab), e-commerce content management system (CMS), open street map (Map), and online shopping (OneStopShop). WebArena-Lite (Liu et al., 2025) is a curated version of WebArena designed for more reliable evaluation. It selects 165 representative tasks for human verification as the evaluation set and uses the remaining 647 tasks for training. It also provides 9,460 trajectories automatically annotated by program-based solvers for behavior cloning. For each website, the authors (Liu et al., 2025) summarize the core functionalities and valid items and construct a set of task prototypes and manually implement rule-based solvers using Playwright scripts for each prototype. The corresponding solvers are executed on the websites to collect ground-truth trajectories. In total, this produces 1,186 valid training samples comprising 9,460 trajectories, released under the Apache License 2.0.

Action Space Agents interact with the environment through a set of predefined actions, including:

- **Click:** simulates a left mouse click on a webpage element.
- **Right Click:** performs a right-click on a specified element.
- **Type:** inputs a text string into an input field.
- **Search:** enters a search query and triggers a search operation.
- **Hover:** moves the cursor over a specific element to reveal tooltips or hidden menus.
- **Scroll Up / Scroll Down:** scrolls the page vertically.
- **Press Enter:** simulates pressing the Enter key, typically after typing.
- **Switch Tab:** changes the current browser tab.
- **Select Dropdown Option:** selects an option from a dropdown menu.
- **Wait:** pauses the agent’s interaction for a brief period.
- **Exit:** terminates the current session with a final message.
- **Go Backward / Go Forward:** navigates backward or forward in the browser history.

Rule-based Metrics In real-world web tasks, there are typically no closed-form solutions, and multiple trajectories may lead to successful task completion. Therefore, we evaluate agents solely based on whether the final goal is achieved and calculate the Success Rate (SR), which indicates whether a task is successfully completed according to the following rule-based evaluation metrics:

- **String Match:** The agent must provide an answer string that matches the expected output.
- **URL Match:** The agent is required to navigate to a specific webpage. Success is determined by comparing the final URL to a reference URL.
- **Program Execution:** The agent must modify webpage content or configuration. Evaluation is performed by executing a rule-based script to extract and verify the final state of the page.

Each task in WebArena is associated with one of these evaluation metrics, along with the corresponding reference answer, target URL, or validation script when applicable. This diverse rule-based metric design ensures consistent evaluation across a wide range of web tasks, while accommodating different task objectives and output formats.

B Implementation Details

We implement our method using Qwen2.5-3B and Llama3.1-8B as the backbone models. By default, we use the instruction-tuned version for both prompting and fine-tuning methods. The reinforcement learning (RL) initialization policy is derived from the supervised fine-tuning (SFT) checkpoint obtained via behavior cloning. Since WebRL leverages additional GPT-4 generated data to train Llama3.1-8B, we ensure a fair comparison by initializing our RL policy with their publicly released checkpoint and applying our end-to-end RL using only the original 647 training tasks, without introducing any extra data.

Our models are trained on a single node of 8 NVIDIA A100 GPUs with 80GB memory via full-parameter fine-tuning. To optimize GPU utilization, we adopt DeepSpeed (Rajbhandari et al., 2020) for distributed training with ZeRO-3 offload, along with gradient checkpointing, FlashAttention-2 (Dao, 2024), and bf16 mixed precision training enabled for computation efficiency. For SFT, we use a learning rate of 5e-5 and a batch size of 128, with a cosine LR scheduler for 5% warmup steps.

Table 4: Comparison of model outputs from WEBAGENT-R1 and WEBAGENT-R1-CoT. We present successful trajectories from both models on the same task (“What are the top-3 best-selling products in Jan 2023?”), showing only the first two steps for clarity (a full trajectory is shown in Figure 6 for additional context). Compared to WEBAGENT-R1, the long-CoT variant WEBAGENT-R1-CoT exhibits a more detailed thinking process.

Model	Model Output
WEBAGENT-R1 (Step 1)	<think> # Element: the ‘REPORTS’ section on the left sidebar. </think> <answer> do(action=“Click”, element=“7”) </answer>
WEBAGENT-R1 (Step 2)	<think> # Element: the ‘Bestsellers’ report option under the ‘Products’ section in the ‘REPORTS’ </think> <answer> do(action=“Click”, element=“27”) </answer>
WEBAGENT-R1-CoT (Step 1)	<think> I need to find the top-3 best-selling products for January 2023. Let me look at the current dashboard. The left sidebar has a “Reports” section (element 7) which is critical for accessing sales data. The dashboard’s main content shows a table with a product (“Sprite Yoga Strap 6 foot”) and quantity (6), but this appears to be static local sales data rather than a report. Since the user needs a specific monthly ranking, the Reports section is the logical path to business intelligence tools that can aggregate and sort data by time period. Clicking “Reports” will likely reveal options for sales analytics, performance tracking, or product rankings required to identify top-selling items in January 2023. </think> <answer> do(action=“Click”, element=“7”) </answer>
WEBAGENT-R1-CoT (Step 2)	<think> Alright, let’s break down my thought process here. The user wants the top-3 best-selling products for January 2023. I previously clicked the “Reports” menu (element 7) to access analytical sections. Now, analyzing the current HTML structure: 1. Locating the Reports Hierarchy : The Reports section’s dropdown (div data-bbox=“88,92,1072,720”) contains nested menus. Under the “Products” subsection (span data-text=“Products”), I see options like “Views”, “Bestsellers” , “Low Stock”, etc. 2. Identifying the Target Element : The element with id=“27” is explicitly labeled ‘Bestsellers’, which directly matches the user’s request. This is the most logical path to access sales performance data. 3. Ruling Out Distractors : Other options like “Orders” or “Tax” under the Sales section don’t specifically target product performance. The “Bestsellers” report is tailored to rank products by sales volume, making it the correct choice. 4. Next Step Justification : Clicking “Bestsellers” (element 27) will likely load a report listing products by sales quantity or revenue. From there, I can extract the top-3 entries for January 2023. Decision : Proceed to click element 27 to access the Bestsellers report, which is the most direct route to fulfill the user’s query. </think> <answer> do(action=“Click”, element=“27”) </answer>

For RL training, we use a constant learning rate of $1e-6$ with a batch size of 16. The KL divergence regularization coefficient β and the clip ratio ϵ are set to 0.001 and 0.2, respectively. The maximum context length and number of new tokens are set to 16,384 and 1024. For efficient LLM rollouts during M-GRPO, we use vLLM (Kwon et al., 2023) with a tensor parallel size of 1 and a GPU memory utilization ratio of 0.7. Rollout sampling is performed with both the temperature and top-p set to 1.0.

C Data Augmentation for Behavior Cloning with long-CoT Traces

As introduced in the ablation study (§ 3.4), we augment the original behavior cloning data by generating long-CoT traces using a strong reasoning

model, QwQ-32B. We then apply SFT to obtain a long-CoT SFT model, followed by RL training to obtain WEBAGENT-R1-CoT. As shown in Table 4, WEBAGENT-R1-CoT is able to generate more detailed thinking compared to WEBAGENT-R1.

D Prompt Templates

The prompt used for data augmentation is shown in Table 5. We define the action space in the system prompt, which is presented in Table 6. By default, we use the version with the thinking format.

E Qualitative Examples

In Figures 6–10, we present a real-world successful trajectory generated by WEBAGENT-R1 for each of the five websites in WebArena.

Table 5: Prompt used for long-CoT data augmentation, with example fill-ins for the fields *user intent*, *action history*, *current observation*, *next action*, and *remarks*, all of which are available in the original behavior cloning data. The full HTML content in *current observation* is omitted for clarity.

Long-CoT Data Augmentation Prompt

You are an expert in explaining the behaviors of web agents. The agent is designed to help a human user navigate the website to complete a task. Given the user’s intent, the agent’s action history, the current HTML content of the web page, and the agent’s next action associated with optional remarks, your goal is to explain the decision-making process from the agent’s perspective using first-person narrative (as if the decision is being made in real time).

User Intent: “Establish a new discussion platform called ‘VirtualRealityVanguard’. It should serve as a cutting-edge forum where VR aficionados can engage in conversations about the newest trends, games, and applications in the virtual reality realm. Please ensure the sidebar features the following tags: virtual reality, technology, trends, gaming.”

Action History:

```
[
  {
    "round": "0",
    "observation": "** Simplified html **",
    "remarks": "# Element: the 'Forums' link at the top center",
    "action": "do(action='Click', element='1')",
  },
  {
    "round": "1",
    "observation": "** Simplified html **",
    "remarks": "# Element: the 'Create forum' button next to the 'List of forums' title",
    "action": "do(action='Click', element='17')",
  },
  {
    "round": "2",
    "observation": "** Simplified html **",
    "remarks": "# Element: the 'Name' field at the top of the page",
    "action": "do(action='Type', argument='VirtualRealityVanguard', element='12')",
  }
]
```

Current Observation: <html> ... </html>

Next Action: do(action="Type", argument="VirtualRealityVanguard", element="14")

Remarks: # Element: the ‘Title’ input field in the middle of the page

Now, please explain the agent’s thinking process using a first-person narrative.

Table 6: System prompt for web agents. By default, we use the version with the thinking format (highlighted in gray). For the variant without the thinking format (discussed in § 3.5), the gray part is simply removed.

System Prompt

You are a professional web browsing agent assistant that can fulfill user's high-level instructions. Given simplified html of the browsed webpage at each step, you plan operations in python-style pseudo code using provided functions.

You should first think about the reasoning process as an internal monologue and then decide an action. The reasoning process and answer are enclosed within `<think>` `</think>` and `<answer>` `</answer>` tags, respectively, i.e., responding in the following format: `<think>` ... `</think>` `<answer>` ... `</answer>`.

More details about the code action: Your action should be readable, simple. Please generate **ONLY ONE ACTION** in one round. Predefined functions are as follows:

```
def do(action, argument, element):
    """A single browsing operation on the webpage.
    Args:
        :param action: one of the actions from ["Click", "Right Click", "Type", "Search", "Hover",
        "Scroll Up", "Scroll Down", "Press Enter", "Switch Tab",
        "Select Dropdown Option", "Wait"].
        :param argument: optional. Only for "Type", "Search", "Switch Tab", and
        "Select Dropdown Option", indicating the content to type in, page number (start from 0)
        to switch, or key to press. "Search" action is equivalent to "Type" action plus "Enter".
        :param element: optional. Only for "Click", "Right Click", "Type", "Search",
        "Select Dropdown Option", and "Hover". Should be specific element id in the HTML.
    Returns:
        None. The webpage will be updated after executing the action.
    """

def exit(message):
    """Ending the browsing process if the assistant think it has fulfilled the goal.
    Args:
        :param message: optional. If user's instruction is a question, return assistant's answer
        in the message based on the browsing content.
    Returns:
        None.
    """

def go_backward():
    """Go back to the previous page."""

def go_forward():
    """Go forward to the next page."""
```

Examples:

- `<think>` # Element: the 'REPORTS' section on the left sidebar `</think>`
`<answer>` `do(action="Click", element="7")` `</answer>`
- `<think>` # Element: the 'Period' dropdown, middle center `</think>`
`<answer>` `do(action="Select Dropdown Option", argument="Month", element="20")` `</answer>`
- `<think>` # Element: the 'From' date picker input field, middle center `</think>`
`<answer>` `do(action="Type", argument="01/01/2023", element="22")` `</answer>`

REMEMBER:

- You can generate **ONLY ONE ACTION** in one round.
- If you have multiple potential actions to explore, you should generate other actions in separate rounds.
- Don't generate an operation element that you do not see in the screenshot.
- Use "# Element" to describe the element you choose in the HTML.
- Use "# Note" to record information useful to answer the instruction if needed.
- If you find yourself fallen into some sort of loop, try to use another method or change your action.
- If you think a page is still loading or still playing animation and you want to wait a while, use "Wait" action
- You are acting in a real world, try your best not to reject user's demand. Solve all the problem you encounter.
- If you think you didn't get expected webpage, you should try using more precise and locative description of the element.
- You should **NEVER** try to use the browser's address bar at the top of the page to navigate.
- Your answer shouldn't be in a code snippet format. Just write the function name and its arguments.
- If you use do function to perform "Click", "Right Click", "Type", "Search", "Select Dropdown Option", and "Hover", the param element must not be None.

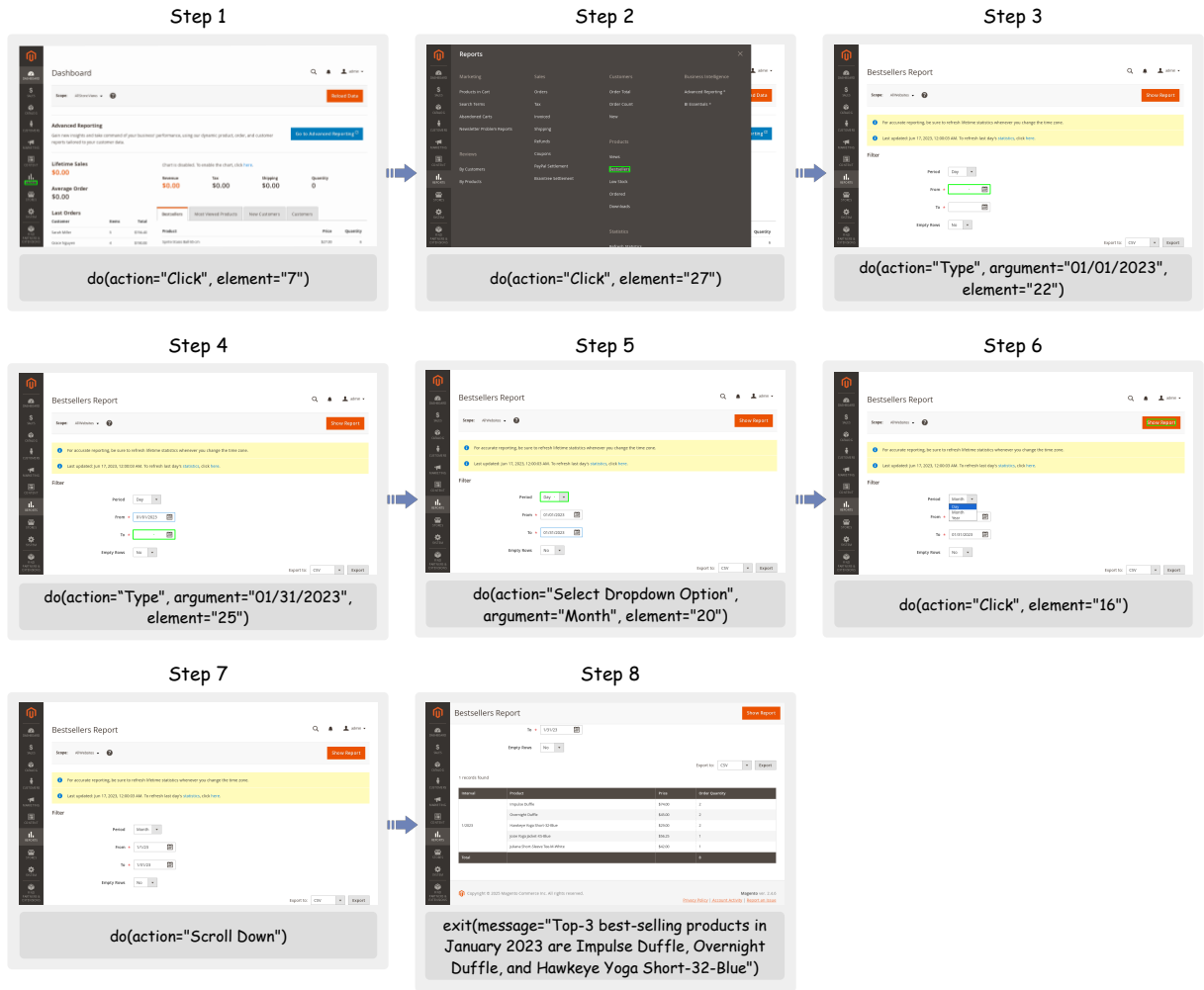


Figure 6: A real-world example of a successful trajectory generated by WEBAGENT-R1 on the CMS task: “What are the top-3 best-selling products in Jan 2023?”.

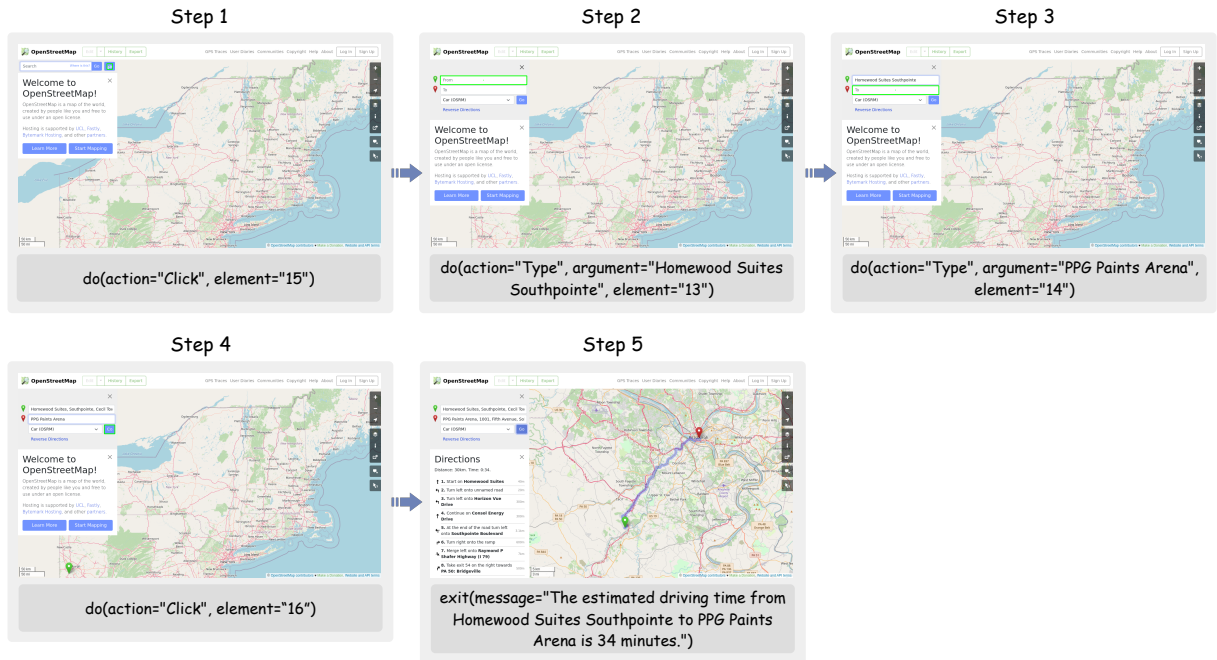


Figure 7: A real-world example of a successful trajectory generated by WEBAGENT-R1 on the Map task: “From my stay at Homewood Suites Southpointe, what’s the estimated driving time to reach PPG Paints Arena?”.

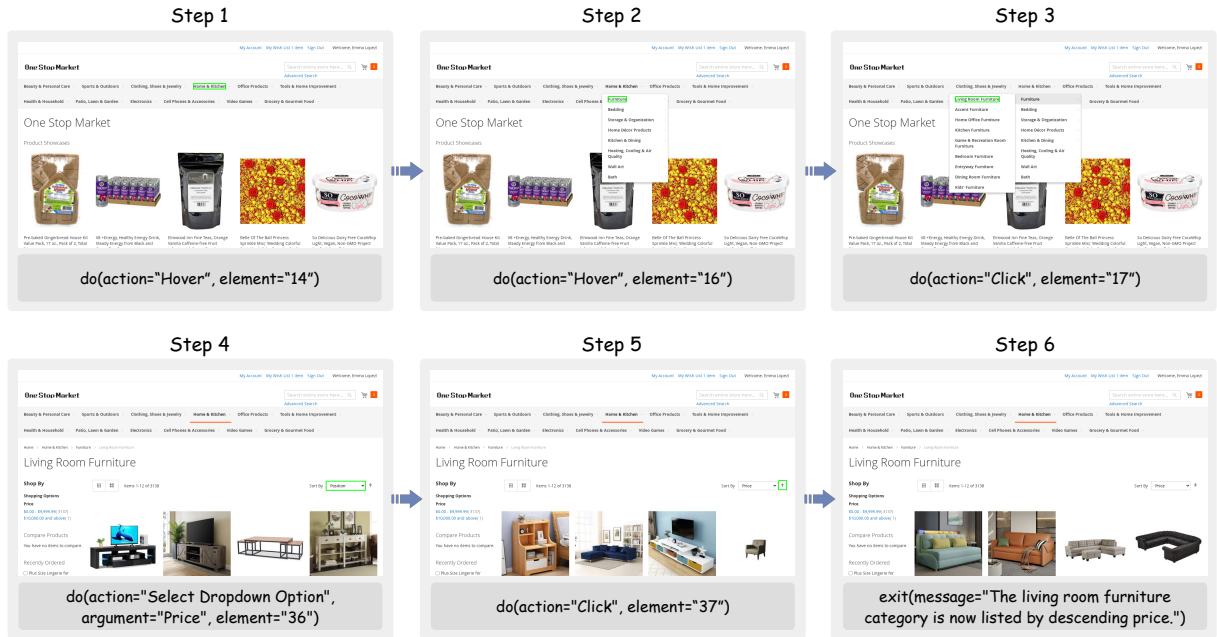


Figure 8: A real-world example of a successful trajectory generated by WEBAGENT-R1 on the Shopping task: “List products from living room furniture category by descending price”.

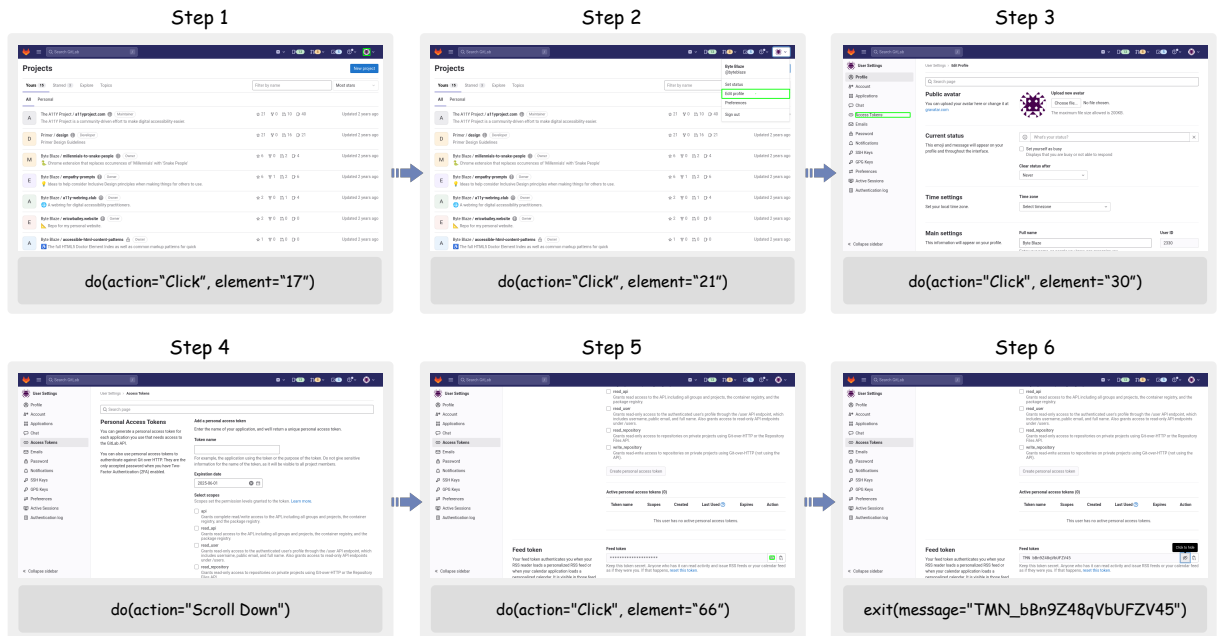


Figure 9: A real-world example of a successful trajectory generated by WEBAGENT-R1 on the GitLab task: “Get me my RSS feed token”.



Figure 10: A real-world example of a successful trajectory generated by WEBAGENT-R1 on the Reddit task: “Edit my post on Star Trek Starfleet Academy series by adding a line to the body that says “Every watch makes me feel like a kid again””.