# OpenWebRL: Demystifying Online Multi-turn Reinforcement Learning for Visual Web Agents

**Rui Yang**[1†*], **Qianhui Wu**[2‡*], **Yuxi Chen**[1], **Hao Bai**[1], **Wenlin Yao**[2], **Hao Cheng**[2], **Baolin Peng**[2], **Huan Zhang**[1], **Tong Zhang**[1], **Jianfeng Gao**[2]
[1]University of Illinois Urbana-Champaign, [2]Microsoft Research
https://openwebrl.github.io

## Abstract

Building capable visual web agents requires long-horizon reasoning, precise grounding, and robust interaction with dynamic real-world websites. Despite rapid progress, the strongest systems remain largely proprietary, while open agents still depend heavily on supervised post-training over large collections of curated web trajectories. This dependence creates a major scalability bottleneck: high-quality demonstrations are expensive to collect, and static datasets offer limited coverage of the diverse, ever-changing open web. Although online reinforcement learning (RL) has shown promise for text-based agents, its potential for training visual web agents directly on live websites remains largely underexplored. In this paper, we introduce **OpenWebRL**, an open framework for training visual web agents with online multi-turn RL on real websites. OpenWebRL covers the full training pipeline, including scalable live-browser infrastructure, supervised initialization, multimodal context management, trajectory-level success judging, and efficient multi-turn policy optimization. Using this framework, we train **OpenWebRL-4B**, which establishes a new open-source state of the art on challenging live-web benchmarks. With only 0.4K initialization trajectories and 2.2K open-ended RL training tasks, OpenWebRL-4B achieves 67.0% success on Online-Mind2Web and 64.0% on DeepShop, outperforming prior open agents of similar or larger scale and remaining competitive with proprietary systems including OpenAI CUA and Gemini CUA. Beyond strong benchmark performance, we systematically study the key design choices that make online RL effective for visual web agents, and analyze how RL improves agentic reasoning. Overall, our work offers a practical path toward building more capable, reproducible, and cost-efficient open web agents.

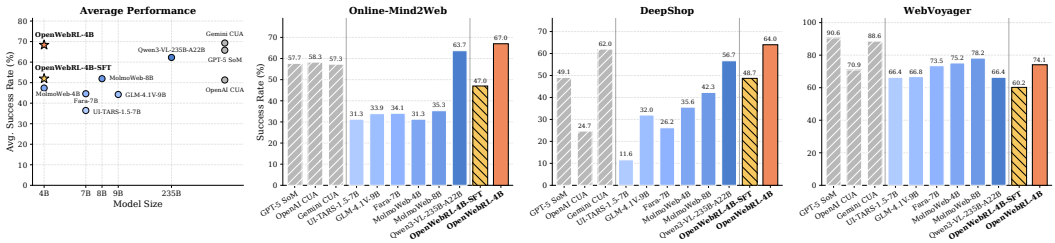



Figure 1: Performance comparison on online web benchmarks, including Online-Mind2Web (2025.04) [49], DeepShop (2025.06) [30], and WebVoyager (2024.01) [19].

*Equal contribution. ‡ Corresponding Author. † Part of the work was done during internship at Microsoft. Emails: ry21@illinois.edu, qianhuiwu@microsoft.com

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

# 1 Introduction

Web agents powered by large vision-language models (VLMs) [1, 6, 39, 38, 7] have made rapid progress on realistic browser tasks, including product search, information extraction, and multi-step navigation [13, 19, 20, 49, 61, 48, 15, 26]. However, the strongest open agents still lag behind proprietary systems in long-horizon planning and adaptation to dynamic websites. At the same time, proprietary models, closed training data and pipelines, and expensive API-based evaluation create substantial barriers for open research. As a result, there remains a widening gap between what closed systems can achieve and what the open research community can study, reproduce, and build upon.

A central bottleneck is the lack of scalable data and training frameworks. Leading open agents such as MolmoWeb [18] rely on supervised post-training over hundreds of thousands of curated trajectories, e.g., 278K trajectories, which are expensive to collect and inherently limited by static coverage [3, 18, 21]. Online reinforcement learning offers a principled alternative: instead of imitating fixed demonstrations, agents can improve by interacting with real websites and learning from task outcomes. Yet applying online RL to visual agents on the open web introduces challenges that are qualitatively different from those in controlled environments: web pages are dynamic and non-stationary, live browser interaction is slow and brittle, and open-ended tasks often lack reliable rule-based verification. Prior RL work has largely sidestepped these challenges by focusing on text-only agents in simulated or self-hosted environments [44, 59, 45]. The closest efforts on live websites, such as PAE [62] and WebGym [4], remain limited in their RL formulation and still rely on costly proprietary evaluation. More broadly, the design choices that make online RL effective for visual web agents remain underexplored.

We introduce **OpenWebRL**, a fully open framework for training visual web agents through direct interaction with live websites. Rather than treating online RL as a black-box recipe, OpenWebRL exposes and systematically studies the key factors behind effective open-web learning. The framework is built on a robust browser infrastructure for large-scale parallel rollout collection [33], and consists of three components: (1) a *supervised warm start* using only 0.4K trajectories, placing the policy in a productive exploration regime before online training; (2) an *agent harness* with multi-tool action execution, textual environment feedback, and multimodal context management to make online agentic RL more efficient and reliable; and (3) a *multimodal multi-turn GRPO objective* with trajectory-level judging, using either a GPT-based or a distilled 8B judge that matches proprietary performance while reducing evaluation cost by roughly $545.5 per experiment. Built with this framework, **OpenWebRL-4B** establishes a new state of the art among open visual web agents across three challenging live web benchmarks. With only a 4B backbone, 0.4K warm-start trajectories, and 2.2K open-web RL training tasks, OpenWebRL-4B achieves success rates of 74.1% on WebVoyager [19], 67.0% on Online-Mind2Web [49], and 64.0% on DeepShop [30]. These results substantially outperform prior open agents, including FARA-7B [3], MolmoWeb-8B [18], and even Qwen3-VL-235B-A22B-Thinking on the evaluated benchmarks, while remaining competitive with several proprietary systems such as GPT-5, OpenAI CUA, and Gemini CUA. Beyond benchmark performance, we systematically analyze the key design choices that make online RL effective for visual web agents and study how RL shapes agentic reasoning during training.

Our contributions are threefold. First, we introduce OpenWebRL, a fully open framework for end-to-end online RL of visual web agents on live websites. Second, we develop a practical multimodal multi-turn RL recipe that combines robust browser infrastructure, trajectory-level judging, and efficient context management, making open-web training effective for compact models. Third, we release a strong 4B-scale open agent together with a detailed empirical study of the ingredients that make online RL successful for visual web interaction. We hope OpenWebRL provides a practical foundation for future research on capable, reproducible, and cost-efficient open web agents.

# 2 Related Work

**LLM/VLM-based Web Agents.** Recent advances in LLM/VLM-based web agents have been driven by three complementary threads: stronger foundation models [6, 39, 11, 50, 22], agent frameworks [16, 51, 43], and web- or GUI-specific post-training [48, 35, 12, 47, 46, 7, 15, 8, 63]. More recent systems further enhance agent capabilities through curated demonstrations, synthetic data, and supervised fine-tuning tailored to realistic browser interaction [26, 3, 18, 21]. In parallel, evaluation has broadened from curated, static web benchmarks toward more diverse and realistic open-web settings [32, 19, 49, 30, 40, 23, 55, 10, 4].

Table 1: Summary of existing multi-turn web agents training pipelines. Simulated web refers to environments such as WebShop; self-hosted web refers to more realistic offline web stacks such as WebArena. Green ✓ / red ✗ indicate whether weights, data, and code were publicly released.

| Paper | Method | Observation | Train Env | Reward | Weights | Data | Code |
|---|---|---|---|---|---|---|---|
| WebRL [34] | Online curriculum RL | Text | Self-hosted web | Trained judge | ✓ | ✓ | ✓ |
| AgentRL [59] | SFT+GRPO | Text | Simulated web | Rule-based | ✗ | ✓ | ✓ |
| PAE [62] | Online Filtered BC | Multimodal | Open web | Prompted judge | ✓ | ✓ | ✓ |
| RAGEN [44] | StarPO | Text | Simulated web | Rule-based | ✗ | ✓ | ✓ |
| WebAgent-R1 [45] | SFT+Multi-turn GRPO | Text | Self-hosted web | Rule-based | ✗ | ✓ | ✓ |
| UI-TARS-2 [7] | Multi-turn PPO | Multimodal | Computer-use sandbox | Mixed judges | ✗ | ✗ | ✗ |
| WebGym [4] | Online Filtered BC | Multimodal | Open web | Prompted judge | ✗ | ✓ | ✓ |
| WebSTAR [21] | Step-filtered SFT | Multimodal | — | Prompted judge | ✗ | ✓ | ✓ |
| GUI-Libra [53] | Step-wise GRPO | Multimodal | — | Rule-based | ✓ | ✓ | ✓ |
| ScaleCUA [26] | SFT | Multimodal | — | — | ✓ | ✓ | ✓ |
| Fara-7B [3] | SFT | Multimodal | — | — | ✓ | ✗ | ✗ |
| MolmoWeb [18] | SFT | Multimodal | — | — | ✓ | ✓ | ✓ |
| OpenWebRL (Ours) | SFT + MM-GRPO | Multimodal | Open web | Mixed judges | ✓ | ✓ | ✓ |

**Reinforcement Learning for Multimodal Agents.** Deploying VLMs as agents in visually grounded environments requires moving beyond static understanding toward long-horizon decision-making[52, 42]. To bridge the gap, recent research adopts a two-stage SFT-then-RL paradigm[9, 57, 58]. The success of outcome-based RL for language reasoning [17] has spurred growing efforts to extend RLVR to VLMs and interactive agents. Early work applies this paradigm to static tasks such as perception, grounding, and visual question answering [27, 25, 37, 41, 28, 29, 53], while more recent methods target multi-turn and agentic settings [34, 62, 5, 4, 44, 59, 45, 7]. However, most of this work operates in simulated, self-hosted, or narrowly scoped environments. In contrast, we study fully open online RL for compact visual web agents trained and evaluated on live websites.

## 3 Preliminaries

**Problem Formulation.** We formulate multimodal web-agent training as a POMDP $\mathcal{M} = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. Each task $q$ specifies a start URL and instruction. At step $t$, the agent observes $o_t = (x_t, I_t)$, where $x_t$ contains textual browser information (URL, tab info, environment feedback) and $I_t$ is a screenshot. Given interaction history $h_t = (q, o_0, a_0, \ldots, a_{t-1}, o_t)$, policy $\pi_\theta$ generates a response $y_t$ containing reasoning and a structured browser action, after which the environment transitions to $s_{t+1} \sim \mathcal{T}(s_{t+1} \mid s_t, a_t)$ and yields $o_{t+1}$. An episode $\tau = \{(h_t, y_t, a_t, o_{t+1})\}_{t=0}^{T-1}$ terminates when the agent calls `done`, the step budget is exhausted, or an environment failure occurs. Task success is evaluated only after the full interaction via rule-based checks or a judge model.

**Multi-turn GRPO.** GRPO [36] is a critic-free policy optimization method that replaces a learned value function with group-relative advantages. In multi-turn settings, the sampled unit is a full trajectory rather than a single response. For each task $q$, the policy samples a group of trajectories $\{\tau_i\}_{i=1}^{G}$, where each trajectory receives a trajectory-level reward $R(\tau_i)$. The group-relative advantage is $A_i = \frac{R(\tau_i) - \text{mean}(\{R(\tau_j)\}_{j=1}^{G})}{\text{std}(\{R(\tau_j)\}_{j=1}^{G})}$. Multi-turn GRPO [45] propagates this trajectory-level signal to the action tokens generated across all turns in $\tau_i$. Denoting by $y_{i,t,k}$ the $k$-th token in the response at turn $t$ of trajectory $\tau_i$, the optimization objective becomes

$$\mathcal{L}_{\text{MT-GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=0}^{T_i-1}\frac{1}{|y_{i,t}|}\sum_{k=1}^{|y_{i,t}|}\min(\rho_{i,t,k}(\theta)A_i,\ \text{clip}(\rho_{i,t,k}(\theta), 1-\epsilon, 1+\epsilon)\, A_i)\,,$$

where $\rho_{i,t,k}(\theta)$ is the token-level importance ratio at turn $t$.

## 4 OpenWebRL: An Online RL Training Framework for Visual Web Agents

We train multimodal web agents with an end-to-end online RL pipeline on the open web, starting from a general-purpose VLM. Our goal is not only to improve visual web-agent performance, but also to systematically study the key components of online RL for open-web navigation. Since online RL is computationally expensive, we focus our main experiments on small VLMs, such as Qwen3-VL-4B [6]. However, general-purpose small VLMs often lack the web-specific knowledge needed to operate

in dynamic browser environments. We therefore warm-start the policy with supervised finetuning on a small set of trajectories collected using a stronger open-source model. Starting from the SFT policy, we iteratively collect online rollouts in live browser environments and update the model with group-relative policy optimization. In the following subsections, we describe the data preparation, rollout system, reward design, and optimization objective.

### 4.1 Agent Harness

Training and evaluating web agents on live websites introduces substantial environment noise, including dynamic page updates, pop-ups, redirects, bot detection, blocking, and transient network failures. To make open-web rollouts more reliable, we build a fault-tolerant browser environment based on Orchard Env [33] with navigation retries, timeout handling, and structured failure attribution, separating unstable website behavior from model behavior and making failures diagnosable during large-scale training. For efficiency, we run parallel browser instances that asynchronously sample independent trajectories, each maintaining its own page state and interaction history. Implementation details are in Appendix A.1.

Based on the environment infrastructure, we employ a generic multi-turn ReAct-style tool-calling agent [54] to avoid conflating scaffold-specific effects with differences in data or training recipes, and more importantly, let the same agent paradigm generalize beyond GUI navigation to any tool-using domain. Within this framework, the only browser-specific design choices are how observations are organized, what tools the agent can call, and how multi-step interaction history is maintained.

**Observation and Environment Feedback.** The environment provides a multimodal observation consisting of the current screenshot, active URL, viewport dimensions, and tab metadata. These signals allow the agent to track browser state across heterogeneous websites with diverse layouts, dynamic content, and multi-tab workflows. In addition, ***each action returns a concise textual environment-feedback message extracted from DOM-tree changes*** between consecutive interaction steps. These messages summarize the execution outcome and observable state changes, such as successful page navigation, new-tab creation, typed-text mismatches, or failed scrolling attempts. This lightweight feedback makes web interaction more observable, allowing the agent to distinguish successful actions from silent failures or unexpected browser behavior commonly encountered on live websites. More details are provided in Appendix A.2.

**Action Space and Multi-tool-call Interface.** The agent is equipped with a structured action space $\mathcal{A}$ of 13 atomic browser tools spanning pointer management (`click`, `hover`, `drag`), keyboard input (`write`, `press_keys`), page navigation (`scroll`, `goto_url`, `go_back`, `wait`), tab management (`new_tab`, `switch_tab`, `close_tab`), and termination via `done(response)`, which is the sole mechanism for successfully ending an episode and emitting the final user-facing answer. At each step, the policy response is expected to consist of one reasoning block followed by ***one or more tool call blocks***. The environment parses the tool calls and executes the corresponding browser actions sequentially before returning per-call feedback and the next screenshot. This ***multi-tool-call interface improves rollout efficiency***: short deterministic interaction chains—*e.g.*, focusing a search box, entering a query, and pressing Enter—can be completed within a single model step, eliminating unnecessary model-environment round trips that would otherwise dominate live-web rollouts. Please refer to Table 6 for more details.

**Context Management.** Long-horizon web interaction creates a fundamental context-management challenge for multimodal RL agents [24]. Each rollout step may contain a full-page screenshot, textual metadata, tool calls, environment feedback, and model reasoning traces. Retaining all screenshots quickly becomes impractical: a 30-step trajectory can exceed the context budget even for 64k-token models. However, preserving every historical screenshot is often unnecessary. Human users do not repeatedly inspect every previous browser state; instead, they rely primarily on recent visual observations together with memory of prior actions, outcomes, and task progress. We adopt the same principle by separating visual grounding from long-term memory. Recent screenshots are retained explicitly for perception, while older interaction history is compressed into textual state information, including environment feedback and the agent's own reasoning traces.

Specifically, at turn $t$, the policy receives the system instruction $s$, the task query $q$, the full sequence of previous model responses $\{y_j\}_{j<t}$, the full sequence of environment feedback strings, and the current browser observation. Each observation is represented as $o_t = (x_t, I_t)$, where $x_t$ contains lightweight textual metadata such as the active tab, URL, and previous-step feedback, while $I_t$

denotes the rendered screenshot. To bound multimodal context cost, we retain only the most recent $K$ screenshots: $\mathcal{I}_t = (I_{\max(0,t-K+1)}, \ldots, I_t)$. Empirically, retaining only the current screenshot ($K = 1$) already achieves strong performance while substantially reducing training cost (Section 5.1). Let $y_j$ denote the model response at turn $j$ which includes both reasoning content and tool calls, the resulting policy context is: $h_t = (s, q, o_0, y_0, o_1, y_1, \ldots, o_{t-1}, y_{t-1}, o_t, \mathcal{I}_t)$.

A key design choice is that ***environment feedback is always preserved***, even after its corresponding screenshot has been discarded. These feedback strings provide the only explicit execution signal for browser actions, indicating whether interactions succeeded, failed, triggered navigation, opened new tabs, or produced unexpected behavior that may not be visually obvious from the screenshot alone. This becomes particularly important under our multi-tool-call interface, where a single model step may execute several browser actions sequentially and the subsequent turn must identify which sub-actions succeeded and which require recovery or retry.

Unlike prior GUI-agent approaches that compress history into short action summaries [48, 53] or executable action code [18], we ***retain the agent's full historical reasoning traces as part of the context***. We find that these reasoning traces naturally serve as compact textual memory, capturing prior visual observations, action intent, intermediate conclusions, and task progress without requiring all past screenshots to remain in context.

### 4.2 Data Preparation and Supervised Fine-tuning

We construct the training corpus from WebGym [4] through a task-filtering, teacher-rollout, and trajectory-curation pipeline. Starting from 292K raw task instances, we remove tasks that overlap with evaluation benchmarks, subtasks decomposed from parent intents, tasks from long-tail or unstable websites, and near-duplicate intents. To identify near duplicates, we embed task intents with `Qwen3-Embedding-8B` and apply greedy similarity-based deduplication with a predefined threshold. We use a threshold of $0.99$ to construct the SFT candidate pool, resulting in 15,601 filtered seed tasks. For the RL task pool, we apply the same filtering pipeline with a stricter threshold of $0.95$ to further reduce semantic redundancy, retaining around 2.2K tasks spanning diverse real-world websites.

To obtain demonstrations for supervised warm-starting, we rollout a strong open-source teacher model, `Qwen3-VL-235B-A22B-Thinking` [6], using the agent harness described in Section 4.1. For each filtered seed task, we sample four independent teacher trajectories and use GPT-4.1 to judge task success based on the final answer, interaction history, and screenshot trajectory. This yields a pool of successful demonstrations with multiple attempts per task, providing both high-quality trajectories and useful signals about task difficulty and rollout diversity. For SFT, we intentionally curate a small, high-quality subset instead of imitating all successful teacher trajectories. The goal is to provide the small model with sufficient web-interaction competence for productive exploration, while avoiding excessive imitation that may limit the effectiveness of subsequent online RL. By default, we select successful trajectories from the PAE-WebVoyager subset, which densely covers popular real-world websites. For each task group, we retain the shortest successful trajectory. When multiple trajectories have the same length, we use the shorter total response length as a tie-breaker. We also cap the number of tasks per website to improve domain diversity. This produces our default SFT set of 412 trajectories spanning 70 websites.

We train our models with multi-turn behavioral cloning on the curated trajectories. Each assistant turn is treated as a training target conditioned on the interaction history, including screenshots, reasoning traces, tool calls, and environment feedback. We apply the loss only to the target assistant response and mask historical context and environment observations.

### 4.3 Reward Design and Judge Model

We design the reward to capture both format correctness and task success, with details provided in Appendix A.4. The reward consists of two components. ***(1) Format reward.*** First, a format reward checks whether the response follows the required browser-agent protocol: $r_{\text{fmt}}(y_t) \in \{0, 1\}$. A response receives format credit only if it contains the required thinking termination tag and a successfully parsed tool call. ***(2) Task-success reward.*** For completed trajectories, we use a VLM-as-a-judge reward to evaluate whether the task has been successfully completed. The judge takes as input the task instruction, the final response, recent screenshots, and the trajectory history, including tool calls and environment feedback. Examples of judge inputs and outputs are provided in Appendix F.2.

The final reward is assigned at the trajectory level. A trajectory receives a positive reward only when it both follows the required response format and is judged successful. It receives a negative reward if it terminates due to repeated format errors, and receives zero reward otherwise. This design encourages the agent to produce valid executable actions while optimizing for actual task completion.

We use GPT-4.1 as the default judge during training. However, relying on a proprietary judge can be costly and may limit accessibility for the research community. In our experiments, a typical training run requires 43.2K judge API calls, costing approximately $545.5. To reduce this cost and make the pipeline more accessible, we distill an 8B judge model from 12.5K diverse online rollouts and GPT-4.1 judge labels. The distilled judge is trained to predict both the judging rationale and the final success evaluation. We evaluate the effectiveness of learned judge models in Section 5.3.

### 4.4 Multimodal Multi-turn GRPO

Since rewards are assigned only at the trajectory level, we extend GRPO to optimize all assistant responses across a rollout, assigning the same group-relative advantage to every turn in the trajectory. For a trajectory $\tau_i$ with $T_i$ assistant turns, we construct $\{(h_{i,t}, y_{i,t})\}_{t=0}^{T_i-1}$, where $h_{i,t}$ is the managed multimodal context and $y_{i,t}$ is the assistant response at turn $t$. We use the same context-management procedure for rollout and optimization, and apply the loss only to assistant response tokens.

For each task $q$, we sample a group of $G$ trajectories from the current policy: $\{\tau_i\}_{i=1}^{G}$, $\tau_i \sim \pi_{\theta_{\text{old}}}$. Let $R_i = R(\tau_i)$ be the trajectory-level reward. Following group-relative policy optimization, we compute a normalized group-relative advantage: $A_i = \frac{R_i - \mu_G}{\sigma_G + \epsilon}$, where $\mu_G = \frac{1}{G}\sum_{j=1}^{G} R_j$ and $\sigma_G$ is the reward standard deviation within the group. The resulting advantage $A_i$ is assigned to all response tokens from all turns of trajectory $\tau_i$. We optimize the clipped multi-turn GRPO objective:

$$\mathcal{L}_{\text{MM-GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G}\sum_{t=0}^{T_i-1} \frac{\sum_k m_{i,t,k} \min\left(\rho_{i,t,k}(\theta)A_i, \text{clip}\left(\rho_{i,t,k}(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\right) A_i\right)}{\max(\sum_k m_{i,t,k}, 1)},$$

where $\rho_{i,t,k}(\theta) = \frac{\pi_\theta(y_{i,t,k}|h_{i,t}, y_{i,t,<k})}{\pi_{\theta_{\text{old}}}(y_{i,t,k}|h_{i,t}, y_{i,t,<k})}$ is the importance ratio for token $k$ in turn $t$ of trajectory $i$, and $m_{i,t,k}$ masks out non-assistant tokens. Note that we do not include the $1/T_i$ normalization factor at the trajectory level, as doing so would downweight longer trajectories and weaken the learning signal for harder tasks that require more interaction steps.

We use asymmetric clipping with $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$, and adopt trajectory-level dynamic sampling from [56], discarding task groups whose trajectories all receive identical rewards (*e.g.*, all-zero or all-one). This filtering removes groups dominated by environment failures or trivial tasks, while creating a natural curriculum that focuses optimization on tasks that are solvable but not yet reliably solved by the current policy.

## 5 Experiments

**Training Settings.** We use `Qwen3-VL-4B-Thinking` [6] as the main base model to keep online RL training computationally feasible. We first warm-start the model with SFT via LlamaFactory[60] for 3 epochs to obtain `OpenWebRL-4B-SFT`. Starting from this SFT checkpoint, we further apply our MM-GRPO algorithm in online web environments, using a curated 2.2K training set described in Section 4.2. The main RL training runs for 90 iterations, where each iteration consists of online rollouts followed by MM-GRPO updates, requiring about 300 B200 GPU hours in total. Across training, we collect approximately 54K online trajectories. Since 30-step rollouts are substantially slower, we train `OpenWebRL-4B` in two stages: 90 iterations with a maximum of 15 rollout steps, followed by 50 iterations with a maximum of 30 rollout steps. We also report the results of a 4B variant trained with our distilled 8B judge model and an `OpenWebRL-8B` trained with the same pipeline. More training details are provided in Appendix A.5.

**Benchmarks.** We evaluate on three challenging live web benchmarks: *WebVoyager* [19], covering open-domain navigation across popular websites (using the FARA-curated 595-task version [3]); *Online-Mind2Web* [49], comprising 300 long-horizon tasks across 136 websites; and *DeepShop* [30], targeting realistic shopping tasks with multi-constraint product selection. We follow each benchmark's standard evaluation protocol. Full details are in Appendix B.

**Baselines.** We compare our models with both proprietary and open-source baselines. The proprietary baselines include GPT-5, Gemini-3-Flash, o3, OpenAI CUA, and Gemini CUA. The open-source

Table 2: Official success rates (%) across three open-web benchmarks. * marks numbers reported in FARA [3]; † marks numbers reported in MolmoWeb [18].

| Model Name | # Steps | # Tasks | WebVoyager | Online-Mind2Web | DeepShop | Average |
|---|---|---|---|---|---|---|
| *Proprietary Models* | | | | | | |
| GPT-5 (Axtree)† | 30 | – | 70.6 | 41.9 | 40.7 | 51.1 |
| Gemini-3-flash (Axtree)† | 30 | – | 74.4 | 34.8 | 45.1 | 51.4 |
| Gemini-3-flash (Axtree)† | 100 | – | 85.6 | 44.8 | 55.3 | 61.9 |
| GPT-4o (SoM)* | 100 | – | 65.1 | 34.6 | 16.0 | 38.6 |
| o3 (SoM)* | 100 | – | 79.3 | 55.4 | 49.7 | 61.5 |
| GPT-5 (SoM)* | 100 | – | **90.6** | 57.7 | 49.1 | 65.8 |
| OpenAI computer-use-preview* | 100 | – | 70.9 | 58.3 | 24.7 | 51.3 |
| Gemini computer-use-preview† | 100 | – | <u>88.6</u> | 57.3 | 62.0 | **69.3** |
| *Open-Source Models* | | | | | | |
| Holo1-7B† | 30 | >15.6k | 55.4 | – | – | – |
| UI-TARS-1.5-7B* | 100 | – | 66.4 | 31.3 | 11.6 | 36.4 |
| GLM-4.1V-9B-Thinking* | 100 | – | 66.8 | 33.9 | 32.0 | 44.2 |
| Fara-7B* | 100 | >123.2k | 73.5 | 34.1 | 26.2 | 44.6 |
| MolmoWeb-4B† | 100 | >278.5k | 75.2 | 31.3 | 35.6 | 47.4 |
| MolmoWeb-8B† | 100 | >278.5k | 78.2 | 35.3 | 42.3 | 51.9 |
| Qwen3-VL-4B-Thinking | 30 | – | 52.6 | 32.0 | 33.3 | 39.3 |
| Qwen3-VL-8B-Thinking | 30 | – | 61.3 | 38.7 | 44.0 | 48.0 |
| Qwen3-VL-235B-A22B-Thinking | 30 | – | 66.4 | 63.7 | 56.7 | 62.3 |
| *Ours: 4B backbone* | | | | | | |
| **OpenWebRL-4B-SFT** | 30 | 0.4k | 60.2 | 47.0 | 48.7 | 52.0 |
| **OpenWebRL-4B** | 30 | 2.2k | 74.1 | 67.0 | 64.0 | 68.4 |
| **OpenWebRL-4B (3 runs)** | 30 | 2.2k | 74.1 | 67.0 | 64.0 | 68.4 |
| **OpenWebRL-4B w/ OpenWebRL-Judge-8B** | 30 | 2.2k | 68.9 | <u>67.3</u> | **68.7** | 68.3 |
| *Ours: 8B backbone* | | | | | | |
| **OpenWebRL-8B-SFT** | 30 | 0.9k | 66.2 | 54.0 | 50.0 | 56.7 |
| **OpenWebRL-8B** | 30 | 2.2k | 73.8 | 67.0 | <u>65.3</u> | 68.7 |
| **OpenWebRL-8B** | 50 | 2.2k | 74.6 | **69.7** | 63.3 | <u>69.2</u> |

baselines include Holo1-7B [2], UI-TARS-1.5-7B [35], GLM-4.1V-9B-Thinking [22], FARA-7B [3], MolmoWeb-4B/8B [18], Qwen3-VL-4B-Thinking, and Qwen3-VL-235B-A22B-Thinking [6].

**Evaluation Metrics.** Our primary metric is the *official success rate*, following the standard evaluation protocol used in prior work [3, 18]. This protocol relies on Browser-Use Stealth Browsers[2], a managed cloud-browser service that provides CAPTCHA solving and stable browser sessions, reducing failures caused by website blocking and session instability during live-web evaluation. However, Browser-Use Service introduces a paid third-party dependency, which increases evaluation cost and makes exact reproduction less accessible for the academic community. To improve transparency, we also report "*success rate w/o aborted tasks*"—the success rate excluding tasks with non-agent failures such as blocked pages or disconnected browser sessions, providing an estimate of agent performance without using Browser-Use Service.

### 5.1 Main Results

**OpenWebRL establishes a new open-source state of the art and is competitive with proprietary web agents.** Table 2 reports official success rates on three live-web benchmarks. With only a 4B backbone, a 30-step evaluation budget, and a much smaller training set than prior open-source systems, OpenWebRL-4B achieves an average success rate of $68.4\%$, substantially outperforming existing open-source agents. The gains of OpenWebRL-4B are particularly pronounced on long-horizon benchmarks, *i.e.*, Online-Mind2Web and DeepShop, as indicated by Appendix C.2. OpenWebRL-4B outperforms FARA-7B by $+32.9$ and $+37.8$ points, and surpasses MolmoWeb-8B by $+31.7$ and $+21.7$ points on the two benchmarks, respectively. Scaling the backbone to 8B further improves the average success rate to $68.7\%$ under the same 30-step evaluation budget. *Increasing the evaluation budget to 50 steps yields a slightly higher average success rate of* $69.2\%$ *for OpenWebRL-8B, suggesting that the model can generalize to longer interaction horizons* despite being trained with 30-step SFT and RL budgets. However, the 50-step setting also incurs substantially higher computation

[2] `https://browser-use.com/stealth-browsers`

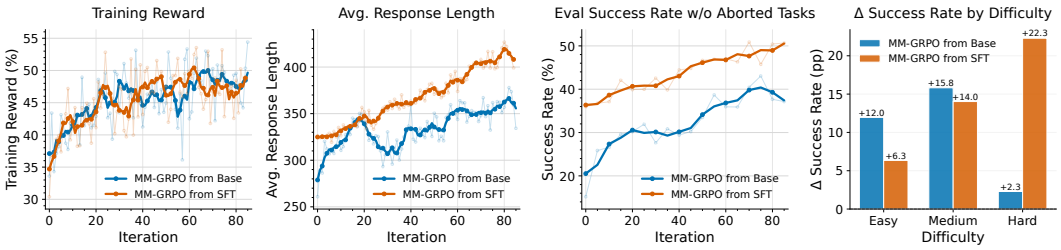



Figure 2: Comparison of MM-GRPO training from SFT and base model. (a) Average training reward and (b) average response length measured on training rollouts. (c) Evaluation success rate w/o aborted tasks and (d) success rate improvement across difficulty splits measured on Online-Mind2Web.

and wall-clock cost, so we use 30 steps as the default evaluation setting. Notably, both OpenWebRL-4B and OpenWebRL-8B are highly competitive with proprietary systems. On Online-Mind2Web and DeepShop, they surpass several closed-source agents, including GPT-5, Gemini-3-Flash, GPT-4o, o3, and the OpenAI/Gemini computer-use agents. These results show that effective online RL can substantially improve open-source VLMs for live-web interaction, enabling compact 4B–8B models to compete with much larger proprietary systems on challenging live-web tasks.

**Both SFT and MM-GRPO contribute to the final gains.** Table 2 shows that supervised fine-tuning provides a strong warm start for both model scales, while MM-GRPO contributes even larger improvements in the subsequent online RL stage. For the 4B backbone, SFT improves the average success rate from $39.3\%$ to $52.0\%$, while MM-GRPO further increases it to $68.4\%$, yielding gains of $+16.4$ points over SFT and $+29.1$ points over the base model. We observe a similar trend for the 8B backbone: SFT improves the average success rate from $48.0\%$ to $56.7\%$, and MM-GRPO further raises it to $68.7\%$. The improvements suggest that SFT provides the interaction competence needed for exploration, while online RL is the key driver of the final performance gains.

### 5.2 Learning Dynamics of MM-GRPO

**SFT initialization leads to faster and more effective RL.** Figure 2 compares MM-GRPO training initialized from the base model (`Qwen3-VL-4B-Thinking`) and from the SFT checkpoint on Online-Mind2Web benchmark. Although both runs eventually reach similar training rewards and follow a similar evaluation trend, the SFT-initialized run maintains a clear $\sim$10% advantage in evaluation success rate throughout training. This gap suggests that supervised warm-starting does more than simply accelerate optimization: it places the policy in a better region of the behavior space, enabling more effective online exploration. Overall, MM-GRPO produces stable improvements from both initializations, and the SFT-initialized policy continues to improve even after 80 iterations.

**The benefit of SFT initialization is especially pronounced on harder tasks.** Figure 2(d) further breaks down performance by task difficulty. MM-GRPO from the SFT checkpoint improves success across all difficulty splits, with the largest gain on hard tasks (+22.3 points). In contrast, RL from the base model improves only marginally on hard tasks (+2.3 points), with most of its gains concentrated on the easy and medium splits. Together, these results validate the effectiveness and necessity of warm-starting online RL from a SFT model rather than training directly from the base model.

**MM-GRPO drives targeted rather than uniform reasoning expansion.** As shown in Figure 2 (b), the model produces longer responses during MM-GRPO training. We further analyze how response length changes in Figure 3. Interestingly, **the overall trajectory length does not increase.** Instead, the average number of interaction steps decreases from 14.0 at iteration 0 to 8.9 at iteration 80, while the average and 90th-percentile trajectory lengths drop from 10.9K and 18.1K tokens to 7.9K and 15.0K tokens, respectively. To better understand the source of the increased response length, we apply lexical proxy detection to model outputs on Online-Mind2Web to identify several step-level reasoning patterns (see Appendix D for details). As shown in Figure 3(b) and (c), MM-GRPO increases both the frequency and the conditional length of several recurring reasoning patterns, including history summarization, blocker diagnosis, retry-plan reasoning, and condition-proof reasoning. For instance, the step-level presence rate rises from 14.5% to 21.4% for history summarization, from 14.2% to 23.7% for blocker diagnosis. The average response length for proxy-bearing steps also grows substantially, from 332 to 542 tokens for history summarization, from 273 to 440 tokens for

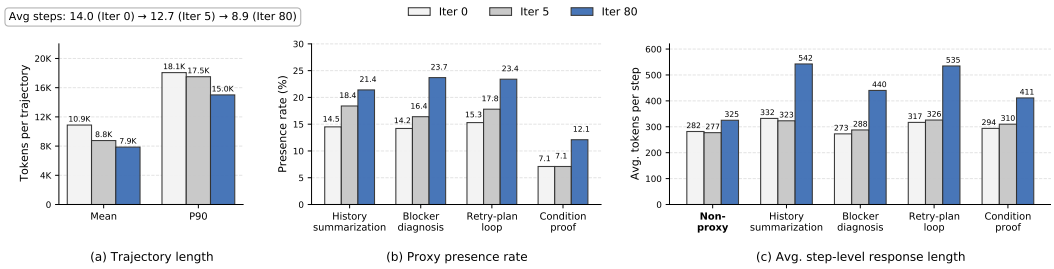



Figure 3: Diagnosing response-length growth during MM-GRPO. (a) Trajectory length, measured by the mean and P90 (90th percentile), and the average interaction steps. (b) Step-level proxy presence rates. (c) Average step-level response length conditioned on proxy type, including no-proxy steps.

Figure 4: Training and evaluation curve for RL with different judges.

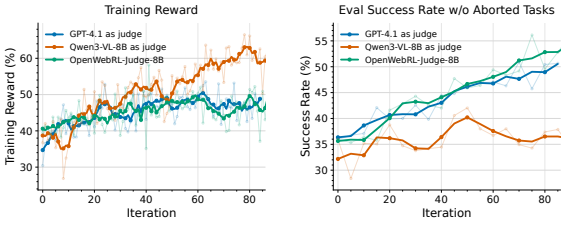



Table 3: Judge model evaluation results. Oracle is labeled by GPT-4.1.

| Model | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| o4-mini | 76.0 | **95.2** | 61.7 | 74.9 |
| gpt-4o | 85.6 | 83.6 | 93.4 | 88.3 |
| WebJudge-7B [49] | 71.6 | 67.9 | **96.9** | 79.8 |
| Qwen3-VL-4B-Instruct | 76.8 | 75.7 | 89.0 | 81.8 |
| Qwen3-VL-8B-Instruct | 80.6 | 78.6 | 91.7 | 84.7 |
| Qwen3-VL-32B-Instruct | 85.8 | 87.4 | 88.3 | 87.8 |
| OpenWebRL-Judge-4B | 85.2 | 86.1 | 89.7 | 87.8 |
| OpenWebRL-Judge-8B | **89.8** | 89.5 | 94.8 | **92.1** |

blocker diagnosis. By comparison, non-proxy steps remain much more stable, increasing only from 282 to 325 tokens on average. ***These results suggest that response-length growth during RL is not a uniform expansion across all responses; instead, the model allocates additional verbosity and reasoning patterns selectively to important steps.***

### 5.3 Evaluation of OpenWebRL-Judge

**The distilled OpenWebRL-Judge-8B provides effective training signals for stable online RL.** As shown in Table 2, *OpenWebRL-4B w/ OpenWebRL-Judge-8B* achieves strong performance on both Online-Mind2Web (67.3) and DeepShop (68.7). Its overall average score (68.3) is nearly identical to that of the GPT-4.1-Judge variant (68.4), despite relying entirely on a small reward model during RL training, substantially reducing both the cost and dependency associated with proprietary APIs. We further compare RL training dynamics in Figure 4. Both the training reward and the evaluation success rate w/o aborted tasks on Online-Mind2Web show stable optimization trends that closely match those obtained with GPT-4.1 supervision. In contrast, using *Qwen3-VL-8B as the judge model leads to clear reward hacking behavior* [14], producing higher training rewards but substantially lower evaluation success rates. These results further demonstrate that OpenWebRL-Judge-8B provides reliable and effective reward signals for stable online RL training.

**OpenWebRL-Judge-8B closely matches GPT-4.1 on held-out trajectory evaluation.** To evaluate the judge model independently from RL training, we construct a held-out set of 500 trajectory rollouts collected from different training stages and annotate them with GPT-4.1. We then compare several judge models by measuring their agreement with these GPT-4.1 annotations. As shown in Table 3, OpenWebRL-Judge-8B achieves the strongest alignment with GPT-4.1, reaching 89.8% accuracy and a 92.1% F1 score. It outperforms strong baselines, including WebJudge-7B [49], Qwen3-VL-32B, and GPT-4o. While some baselines achieve strong precision or recall individually, their overall accuracy and F1 scores remain lower, suggesting that our distilled judge captures web-task success signals more reliably than both general-purpose VLM judges and prior web-specific judge models.

### 5.4 Ablations on Rollout and Context Management

Here we ablate key design choices in OpenWebRL-4B: (i) the rollout-length curriculum used in OpenWebRL-4B, where we first run online RL with a 15-step horizon and then continue with a 30-step horizon; and (ii) the three context management strategies, including textual environment

Table 4: Ablation study on rollout length and context management strategies for online RL.

| System | WebVoyager | Online-Mind2Web | DeepShop |
|---|---|---|---|
| OpenWebRL-4B | **74.1** | **67.0** | **64.0** |
| *Rollout length ablations* | | | |
| RL w/ 30-Step Rollout Only | 66.7 (↓ -7.4) | 65.4 (↓ -1.6) | 63.3 (↓ -0.7) |
| RL w/ 15-Step Rollout Only | 70.1 (↓ -4.0) | 65.0 (↓ -2.0) | 63.3 (↓ -0.7) |
| RL w/ 10-Step Rollout Only | 70.6 (↓ -3.5) | 60.7 (↓ -6.3) | 57.3 (↓ -6.7) |
| *Context management ablations with 15-step rollouts* | | | |
| w/ Recent Two Screenshots | 68.2 (↓ -1.9) | 65.3 (↑ +0.3) | 59.3 (↓ -4.0) |
| w/o Textual Environment Feedback | 64.9 (↓ -5.2) | 57.0 (↓ -8.0) | 56.7 (↓ -6.6) |
| w/o Historical Reasoning | 55.5 (↓ -14.6) | 41.3 (↓ -23.7) | 54.7 (↓ -8.6) |

feedback, historical reasoning, and the number of recent screenshots retained in context. To ensure a controlled comparison, all variants are initialized from the same SFT checkpoint and use the identical RL training recipe, with only the ablated factor changed. For cost efficiency, the context-management ablations are trained using only 15-step rollouts and are compared against the "*RL w/ 15-Step Rollout Only*" baseline. Table 4 reports the results.

**Rollout-length curriculum.** OpenWebRL-4B outperforms all fixed-budget RL variants. Training with only 30-step rollouts leads to performance drops of $7.4$, $1.6$, and $0.7$ points on WebVoyager, Online-Mind2Web, and DeepShop, respectively. Using only 15-step rollouts performs better than starting directly with 30-step rollouts, but still underperforms the full curriculum by $4.0$, $2.0$, and $0.7$ points. Moreover, using an overly short horizon, such as 10-step rollouts, substantially hurts performance on Online-Mind2Web and DeepShop ($-6.3$ and $-6.7$ points), both of which require longer interaction horizons to complete tasks successfully. These results suggest that a medium-horizon training stage helps stabilize early exploration, while longer-horizon rollouts improve the policy's ability to handle tasks requiring extended interactions. Overall, combining 15-step and 30-step training stages is more effective than using a fixed rollout budget throughout training.

**Textual environment feedback provides lightweight but informative signals.** Removing this feedback reduces RL performance by $5.2$, $8.0$, and $6.6$ points relative to the 15-step rollout baseline. These results suggest that textual feedback complements screenshots by clarifying the environment state and supporting more informed decisions.

**Historical reasoning provides critical contextual signals.** Removing historical reasoning yields the largest degradation among all ablations, reducing performance by $14.6$, $23.7$, and $8.6$ points relative to the 15-step results. We conjecture that these reasoning traces can serve as a compact memory of the high-level plan, interpretations of past observations, attempted actions, and their outcomes. Without this information, the model must effectively replan from scratch at each step and loses access to accumulated context, which is particularly detrimental in long-horizon web navigation.

**Keeping more recent screenshots does not consistently help.** By default, the model retains only the most recent screenshot in context. Increasing this to the two most recent screenshots does not yield consistent gains, changing performance by $-1.9$, $+0.3$, and $-4.0$ points. This suggests that historical reasoning already captures most useful information from earlier visual observations, while additional screenshots mainly increase context length and add visual token overhead. It significantly increases training cost, extending runtime from approximately 240 GPU hours to 400 GPU hours.

## 6 Conclusion

We introduced **OpenWebRL**, an open framework for training visual web agents with online multi-turn RL on live websites. Built on this framework, **OpenWebRL-4B** achieves strong performance on challenging live-web benchmarks, demonstrating that effective web-agent training can emerge from relatively modest supervised initialization and scalable online interaction, rather than relying on large-scale demonstration datasets. Through systematic analysis, we identify several key ingredients that make online RL effective for visual web agents, including supervised warm start, textual environment feedback, historical reasoning for long-horizon memory, rollout-length curricula, and reliable trajectory-level success judging. We hope OpenWebRL serves as an actionable and reproducible foundation for future research on open, cost-efficient, and scalable web-agent training, and helps narrow the gap between open and proprietary systems for long-horizon interaction on the live web.

## Acknowledgments and Disclosure of Funding

This work was supported in part by NSF grants 2416897, 2505932, and 2601666 and ONR grant N000142512318. Computing resources were provided by the National Center for Supercomputing Applications (NCSA) through Delta (NSF OAC-2005572) and DeltaAI (NSF OAC-2320345), as well as Illinois Computes and the NAIRR Pilot (NAIRR250157).

## References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Mathieu Andreux, Breno Baldas Skuk, Hamza Benchekroun, Emilien Biré, Antoine Bonnet, Riaz Bordie, Nathan Bout, Matthias Brunel, Pierre-Louis Cedoz, Antoine Chassang, et al. Surfer-h meets holo1: Cost-efficient web agent powered by open weights. *arXiv preprint arXiv:2506.02865*, 2025.

[3] Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-7B: An efficient agentic model for computer use. *arXiv preprint arXiv:2511.19663*, 2025.

[4] Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. WebGym: Scaling training environments for visual web agents with realistic tasks. *arXiv preprint arXiv:2601.02439*, 2026.

[5] Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[6] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[7] ByteDance Seed. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

[8] Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, volume 2025, pages 63707–63738, 2025.

[9] Hanyang Chen, Mark Zhao, Rui Yang, Qinwei Ma, Ke Yang, Jiarui Yao, Kangrui Wang, Hao Bai, Zhenhailong Wang, Rui Pan, et al. Era: Transforming vlms into embodied agents via embodied prior learning and online reinforcement learning. *arXiv preprint arXiv:2510.12693*, 2025.

[10] Yuxi Chen, Haoyu Zhai, Chenkai Wang, Rui Yang, Lingming Zhang, Gang Wang, and Huan Zhang. Captcha solving for native gui agents: Automated reasoning-action data generation and self-corrective training. *arXiv preprint arXiv:2603.23559*, 2026.

[11] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 24185–24198, 2024.

[12] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, 2024.

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

[14] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International conference on machine learning*, pages 10835–10866. PMLR, 2023.

[15] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

[16] Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

[17] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[18] Tanmay Gupta, Piper Wolters, Zixian Ma, Peter Sushko, Rock Yuren Pang, Diego Llanes, Yue Yang, Taira Anderson, Boyuan Zheng, Zhongzheng Ren, Harsh Trivedi, Taylor Blanton, Caleb Ouellette, Winson Han, Ali Farhadi, and Ranjay Krishna. MolmoWeb: Open visual web agent and open data for the open web, 2026.

[19] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

[20] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27545–27564, 2025.

[21] Yifei He, Pranit Chawla, Yaser Souri, Subhojit Som, and Xia Song. WebSTAR: Scalable data synthesis for computer use agents with step-level filtering. *arXiv preprint arXiv:2512.10962*, 2025.

[22] Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv e-prints*, pages arXiv–2507, 2025.

[23] Yining Hong, Rui Sun, Bingxuan Li, Xingcheng Yao, Maxine Wu, Alexander Chien, Da Yin, Ying Nian Wu, Zhecan Wang, and Kai-Wei Chang. Embodied web agents: Bridging physical-digital realms for integrated agent intelligence. *Advances in Neural Information Processing Systems*, 38, 2026.

[24] Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. Rethinking memory mechanisms of foundation agents in the second half: A survey. *arXiv e-prints*, pages arXiv–2602, 2026.

[25] Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Xu Tang, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*, 2025.

[26] Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, et al. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*, 2025.

[27] Ziyu Liu, Zeyi Sun, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. Visual-rft: Visual reinforcement fine-tuning. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2034–2044. IEEE, 2025.

[28] Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Pengxiang Zhao, Guangyi Liu, et al. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 17608–17616, 2026.

[29] Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

[30] Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, Maarten de Rijke, Zhaochun Ren, and Xiuying Chen. DeepShop: A benchmark for deep research shopping agents. *arXiv preprint arXiv:2506.02839*, 2025.

[31] Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories, 2025.

[32] Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.

[33] Baolin Peng, Wenlin Yao, Qianhui Wu, Hao Cheng, Xiao Yu, Rui Yang, Tao Ge, Alessandro Sordoni, Xingdi Yuan, Yelong Shen, et al. Orchard: An open-source agentic modeling framework. *arXiv preprint arXiv:2605.15040*, 2026.

[34] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Xinyue Yang, Jiadai Sun, Yu Yang, Shuntian Yao, Tianjie Zhang, et al. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

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

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

[37] Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, et al. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*, 2025.

[38] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[39] Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, et al. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*, 2025.

[40] Brandon Trabucco, Gunnar Sigurdsson, Robinson Piramuthu, and Ruslan Salakhutdinov. InSTA: Towards internet-scale training for agents, 2025.

[41] Haozhe Wang, Chao Qu, Zuming Huang, Wei Chu, Fangzhen Lin, and Wenhu Chen. Vl-rethinker: Incentivizing self-reflection of vision-language models with reinforcement learning. *arXiv preprint arXiv:2504.08837*, 2025.

[42] Kangrui Wang, Pingyue Zhang, Zihan Wang, Yaning Gao, Linjie Li, Qineng Wang, Hanyang Chen, Yiping Lu, Zhengyuan Yang, Lijuan Wang, et al. Vagen: Reinforcing world model reasoning for multi-turn vlm agents. *Advances in Neural Information Processing Systems*, 38:172871–172933, 2026.

[43] Zhaoyang Wang, Qianhui Wu, Xuchao Zhang, Chaoyun Zhang, Wenlin Yao, Fazle Elahi Faisal, Baolin Peng, Si Qin, Suman Nath, Qingwei Lin, et al. Webxskill: Skill learning for autonomous web agents. *arXiv preprint arXiv:2604.13318*, 2026.

[44] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning, 2025.

[45] Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. WebAgent-R1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7909–7928, Suzhou, China, 2025. Association for Computational Linguistics.

[46] Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

[47] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

[48] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

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

[50] Jianwei Yang, Reuben Tan, Qianhui Wu, Ruijie Zheng, Baolin Peng, Yongyuan Liang, Yu Gu, Mu Cai, Seonghyeon Ye, Joel Jang, et al. Magma: A foundation model for multimodal ai agents. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14203–14214. IEEE, 2025.

[51] Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. In *International Conference on Learning Representations*, volume 2025, pages 97533–97565, 2025.

[52] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

[53] Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baoling Peng, Huan Zhang, Jianfeng Gao, and Tong Zhang. GUI-Libra: Training native GUI agents to reason and act with action-aware supervision and partially verifiable RL, 2026.

[54] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[55] Kuai Yu, Naicheng Yu, Han Wang, Rui Yang, and Huan Zhang. How do visual attributes influence web agents? a comprehensive evaluation of user interface design factors. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 34520–34536, 2026.

[56] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[57] Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in neural information processing systems*, 37:110935–110971, 2024.

[58] Qiusi Zhan, Hyeonjeong Ha, Rui Yang, Sirui Xu, Hanyang Chen, Liang-Yan Gui, Yu-Xiong Wang, Huan Zhang, Heng Ji, and Daniel Kang. Beat: visual backdoor attacks on vlm-based embodied agents via contrastive trigger learning. In *International Conference on Learning Representations*, volume 2026, pages 93114–93135, 2026.

[59] Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, Rui Lu, Hongning Wang, Jie Tang, and Yuxiao Dong. AgentRL: Scaling agentic reinforcement learning with a multi-turn, multi-task framework, 2025.

[60] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[61] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 414–431, 2025.

[62] Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator (PAE): Autonomous skill discovery for foundation model internet agents. In *International Conference on Machine Learning*, 2025.

[63] Yuchen Zhuang, Di Jin, Jiaao Chen, Wenqi Shi, Hanrui Wang, and Chao Zhang. Workforceagent-r1: Incentivizing reasoning capability in llm-based web agents via reinforcement learning. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 34–49, 2026.

# A Implementation Details

## A.1 Robust Web Environment Infra

To support browser-agent training on live websites, we implement a robust web environment that explicitly handles the instability of the open web. Unlike self-hosted benchmark sites, live websites exhibit frequent layout changes, asynchronous loading, pop-ups, redirects, anti-automation defenses, network errors, and non-deterministic page states.

**Sandboxed Execution.** The environment is built on Playwright with a Chromium backend and supports sandboxed execution by default. Each rollout runs inside an isolated Kubernetes sandbox [33] with its own browser environment server, preventing crashes, stale cookies, memory leaks, and site-specific side effects from contaminating other trajectories.

**Timeouts and Recovery.** The browser environment uses separate timeouts for initialization, interaction, and screenshot capture. Initial navigation is retried multiple times to handle common live-site failures such as HTTP/2 errors, connection resets, slow scripts, and transient bot-detection behavior.

**Failure Attribution and Diagnostics.** The rollout layer separates model failures from environment and infrastructure failures using explicit termination reasons, including task completion, maximum-step exhaustion, generation length limits, formatting errors, environment step errors, sandbox failures, and initialization failures. The system further records diagnostics such as server latency, memory usage, uptime, request counts, and tracebacks, enabling post-hoc analysis of whether a low reward reflects model behavior or external factors such as blocking, network instability, or server failure. Network-related initialization errors can also update a host blacklist, allowing future training to avoid websites that are repeatedly unreachable or automation-hostile.

## A.2 Textual Environment Feedback

Browser agents operate inside an instrumented web environment, which exposes state signals that are not directly visible from screenshots. These signals include the DOM element under an action coordinate, the currently focused input field, the active URL, tab state, and lightweight accessibility-tree changes. **If the agent relies only on visual input, it often needs multiple historical screenshots to infer whether the current state has changed after an action.** For example, after a `click`, a screenshot alone may not clearly indicate whether the intended element was selected, whether an input field was focused, or whether navigation was blocked. Similarly, after a `scroll`, the agent may not know whether the page actually moved or has already reached the boundary. Without such feedback, the agent can easily repeat the same failed click or continue scrolling in the current state. To provide this information more efficiently, we augment visual observations with lightweight textual environment feedback.

For each parsed tool call, the environment generates feedback by comparing browser states before and after action execution. Before executing an action, it records lightweight state variables such as the active page URL, the number of open tabs, the current scroll position, the focused element, and, when applicable, the DOM element located at the action coordinates. The environment then executes the corresponding Playwright operation and applies action-specific rule logic to convert the observed state change into a concise textual message. When an action contains multiple tool calls, the environment processes them sequentially and returns the feedback as a list, with each entry corresponding to one executed tool call.

Several representative feedback rules are summarized below and in Table 5. For `click`, the feedback uses the clicked coordinate, the DOM element returned by `document.elementFromPoint`, changes in the page URL, and changes in the number of open tabs. For `write`, the feedback reports the currently focused element and compares the intended text with the actual value in the field after typing. For `scroll`, the feedback compares pre- and post-action scroll offsets to identify successful scrolling, boundary conditions, or no-op actions. For `press_keys`, the environment reports the normalized key sequence and whether the action triggers a URL change.

Execution exceptions are also caught and converted into explicit failure messages rather than silently terminating the rollout. Successful actions return concise messages describing both execution status

Table 5: Examples of action-level environment feedback returned by the browser environment. Feedback exposes both execution status and observable browser-state changes.

| Action | Feedback signal | Example feedback |
|---|---|---|
| `click` | Target element, location, navigation or new-tab effect | `Succeed: ‘click‘ on <button> "Search" at (512, 86) executed. Page navigated to ...` |
| `click` | No-op detection | `Succeed: ‘click‘ on <div> at (211, 335) executed. Note: no visible navigation or new tab detected.` |
| `write` | Focused element and typed content | `Succeed: ‘write‘ typed "Alpine Ridge" into <input> role=combobox "Search".` |
| `write` | Actual value mismatch | `Note: the field's actual value is "New York, NY", which differs from the typed text.` |
| `scroll` | Scroll direction, amount, and boundary detection | `Succeed: ‘scroll‘ down by 50% executed. Note: page scroll position did not change and may be at a boundary.` |
| `press_keys` | Normalized keys and navigation effect | `Succeed: ‘press_keys‘ [‘Enter‘] executed. Page navigated to ...` |
| `goto_url` | Direct navigation success or failure | `Failed: ‘goto_url‘ execution failed: Page.goto: net::ERR_HTTP2_PROTOCOL_ERROR ...` |
| `tab ops` | New, switched, or closed tab state | `Succeed: ‘switch_tab‘ switched to tab 1 (https://example.com).` |
| `done` | Task termination with final response | `Succeed: task marked as done. Response: ...` |

and observable browser-state changes. The resulting feedback is appended to the next observation, giving the policy a compact signal for detecting failed clicks, blocked navigation, unintended tab changes, rejected inputs, boundary scrolling, and other common open-web interaction issues.

### A.3 Action Space

We define a compact browser action space consisting of 13 atomic tools, as summarized in Table 6. These tools cover the core operations needed for open-web interaction, including pointer control, keyboard input, page navigation, tab management, and task termination. Each action is represented as a structured tool call with explicit arguments, which makes the policy output easy to parse and execute in the browser environment.

### A.4 Reward Design

Our browser RL reward is a hybrid of deterministic rule-based verification and LLM-as-a-judge evaluation. The deterministic rules serve two purposes: they enforce the interaction protocol required by the browser environment, and they avoid unnecessary judge calls for trajectories that are clearly invalid or incomplete. **The judge model is only used for completed trajectories that pass these basic validity checks.** For each generated trajectory $\tau_i$, we compute three reward-related quantities: a format score $F(\tau_i)$, a judge score $J(\tau_i)$, and the final training reward $R(\tau_i)$.

Table 6: Browser action space: 13 atomic tools grouped by family.

| Category | Tool | Argument | Description |
|---|---|---|---|
| Pointer Mgmt. | `click` | `x, y, button, click_type` | Mouse-click at a screen pixel; supports single/double click and left/right/middle button. |
| | `hover` | `x, y` | Move the cursor to a pixel to reveal tooltips or open dropdowns. |
| | `drag` | `x1, y1, x2, y2` | Drag-and-drop from a start pixel to an end pixel. |
| Keyboard Mgmt. | `write` | `text` | Clear the focused input and type a string. |
| | `press_keys` | `keys` | Press keys sequentially or as a hotkey combo. |
| Page Nav. | `scroll` | `direction, amount` | Scroll the page or element by a viewport fraction. |
| | `goto_url` | `url` | Navigate the current tab to a given URL. |
| | `go_back` | — | Navigate back in the browser history. |
| | `wait` | `seconds` | Pause for $N$ seconds to allow the page to settle. |
| Tab Mgmt. | `new_tab` | — | Open a new blank browser tab. |
| | `switch_tab` | `index` | Switch to the tab with the given 0-based index. |
| | `close_tab` | — | Close the current tab. |
| Termination | `done` | `answer` | End the episode and emit the final answer. |

**Rule-based format verification.** Before invoking the judge model, we verify that the agent response follows the required browser-action format. In our response format, a turn is considered valid only if it contains the expected closing thinking tag and a parseable tool call. This ensures that the model emits an executable browser action rather than free-form text.

**Rule-based status filtering.** The judge model is not called for trajectories that do not reach a completed browser state, where agent outputs 'done' by itself. If the rollout terminates because of malformed tool calls, generation aborts, context-length truncation, environment errors, or exhausting the maximum number of browser steps, the judge score is set to zero by rule:

$$J(\tau_i) = 0.$$

Malformed format termination is treated as a special failure case and receives a negative final reward:

$$R(\tau_i) = -1 \quad \text{if the trajectory terminates due to repeated format errors.}$$

This penalizes trajectories that cannot produce executable browser actions.

Even for completed trajectories, the judge model is skipped if no final answer can be extracted. The final answer must come from the terminal `done` tool call. If the final answer is missing, we set $J(\tau_i) = 0$ without querying the judge.

**VLM-as-a-judge evaluation.** For completed trajectories with a valid final answer, we query a judge model to determine task success. The judge input contains the original task instruction, the agent's final answer, three recent screenshots from the trajectory, and an explicit action history summarizing the agent's tool calls and environment feedback. The judge is instructed to return a verdict of either `SUCCESS` or `NOT SUCCESS`.

The judge output is mapped back to a binary score by a deterministic parser:

$$J(\tau_i) = \begin{cases} 1, & \text{if the judge output contains } \texttt{SUCCESS}, \\ 0, & \text{if the judge output contains } \texttt{NOT SUCCESS}, \\ 0, & \text{if the judge output cannot be parsed.} \end{cases}$$

The parser checks for `NOT SUCCESS` before `SUCCESS` to avoid mistaking a negative verdict for a positive one.

**Final reward composition.** The final reward is a gated combination of the rule-based checks and the judge score:

$$R(\tau_i) = \begin{cases} -1, & \text{if the trajectory terminates due to repeated format errors,} \\ 0, & \text{if } F(\tau_i) = 0, \\ J(\tau_i), & \text{otherwise.} \end{cases}$$

Thus, a trajectory can receive reward 1 only if it both follows the required browser-action format and is judged successful.

**Invalid or unreliable samples.** If reward evaluation fails because of judge timeout or infrastructure-related rollout failures, the sample can be marked for removal from optimization. Such samples have their loss mask zeroed, so they do not contribute gradients. This keeps reward-model or environment-system failures from introducing noisy policy updates.

Table 7: Training hyperparameters for browser-agent reinforcement learning. Values summarize the Qwen3-VL 4B browser training configuration in `examples/browser`; ranges indicate script variants.

| **Hyperparameter** | **Value** |
|---|---|
| Number of training iterations | 90 |
| Max rollout steps | 15, 30 |
| Context screenshots per turn | 1 |
| Judge screenshots per trajectory | 3 |
| Judge model | GPT-4.1 or our trained 8B judge model |
| Effective rollout query number per iteration | 48 |
| Group size per prompt | 5 |
| PPO epochs | 2 |
| Maximum response length | 1024 |
| Maximum context length | 32768 |
| RL algorithm | MM-GRPO |
| KL coefficient | 0.0 |
| Entropy coefficient | 0.0 |
| PPO clipping range | low 0.2, high 0.28 |
| Global batch size for optimization | 256 |
| Optimizer | Adam |
| Learning rate | $1 \times 10^{-6}$, constant schedule |
| Weight decay | 0.1 |
| Adam betas | $(0.9, 0.98)$ |
| Training backend | Megatron |
| Tensor parallel size | 4 |
| Micro-batch size | 1 |
| Attention backend | FlashAttention |
| Browser viewport | $1280 \times 1000$, DPR 1 |
| Coordinate normalization scale | 1000 |
| Sampling temperature | 0.8 |
| Rollout engine | SGLang |
| Sandbox mode | Kubernetes sandbox |
| Sandbox CPU and memory | 1 CPU, 4 GiB |
| Max concurrent sandboxes | 80–100 during training |
| Inference step timeout | 30 s |
| Rollout task timeout | 600 s |
| Sandbox acquire timeout | 600 s |
| Browser step request timeout | 45 s |
| Browser step retries | 2 |
| Environment exit timeout | 15 s |
| Rollout abort wait timeout | 30 s |
| Rollout health check | Interval 30 s, timeout 30 s, first wait 180 s |

### A.5 Training Details

**SFT.** We train Qwen3-VL-4B-Thinking for 3 epochs with peak learning rate $10^{-5}$ under a cosine schedule with a $10\%$ linear warmup. Each optimizer step uses a per-device batch of 2 with 8-step gradient accumulation, giving a per-worker effective batch of 16 and a global batch of 128 across 8

Table 8: Evaluation hyperparameters for browser-agent evaluation in `examples/browser`.

| Hyperparameter | Value |
|---|---|
| Max evaluation rollout steps | 30 |
| Sandbox mode | Kubernetes sandbox |
| Sandbox CPU and memory | 1 CPU, 4 GiB |
| Parallel browser sandboxes | 16 |
| Context screenshots per turn | 1 |
| Judge screenshots per trajectory | 3 |
| Judge model | GPT-4.1 for our evaluation success rate w/o aborted tasks;<br>officially recommended models for reporting the official score |
| Judge timeout | 120 s |
| Generation temperature | 0.6 for the official-score evaluation and 0.0 for evaluation success rate w/o aborted tasks |
| Top-$p$ | 0.95 for the official-score evaluation and 1.0 for evaluation success rate w/o aborted tasks |
| Top-$k$ | 20 for the official-score evaluation and 1 for evaluation success rate w/o aborted tasks |
| Maximum response length | 4096 tokens |
| Maximum context length | 32768 tokens |
| Task timeout | 600 s |
| Inference step timeout | 120 s |
| Rollout engine | SGLang |
| SGLang dtype | bfloat16 |
| SGLang context length | 32768 tokens |
| SGLang tensor parallel size | 2 |
| SGLang data parallel size | 4 |

data-parallel workers. For the 8B model, we use the same training setup, with the only difference being the SFT dataset: we add $500$ trajectories from the InSTA-v3 subset, resulting in $912$ trajectories in total.

**MM-GRPO.** The policy is initialized from the SFT checkpoint. During training, each task is executed interactively in a live browser environment. Each rollout state comprises the current observation, recent screenshot context, and environment feedback in browser tool-call format. Rewards combine format validity with an LLM-as-a-judge success signal, where the judge receives the full action history and recent trajectory screenshots as input. To ensure stable training, we apply bounded rollout execution with explicit timeouts governing model generation, browser actions, sandbox acquisition, and task-level rollout completion. Training runs for 90 iterations and requires approximately 300 B200 GPU hours in total. We employ trajectory-level dynamic sampling, collecting rollouts until 48 effective groups of size 5 are obtained. Key training hyperparameters for our OpenWebRL-4B are summarized in Table 7. Regarding the 8B model, we use a lower learning rate of $5 \times 10^{-7}$.

### A.6 Evaluation Details

Each evaluation task is executed as an interactive browser trajectory with a maximum of 30 steps. We serve our model using the SGLang framework. For **official success rate evaluation**, we follow the judge protocol of prior work and use Browser-Use Stealth Browsers. Based on the finding of [18] that stochastic sampling outperforms deterministic sampling for web agents, we adopt stochastic decoding with the following hyperparameters: temperature 0.6, top-$p$ 0.95, top-$k$ 20, max response length 4096 tokens, and repetition penalty 1.0. For **success rate evaluation without aborted tasks**, we use deterministic decoding with temperature 0.0. All evaluation settings and decoding parameters are summarized in Table 8.

## B Benchmark Details

We evaluate on three online benchmarks:

- **WebVoyager** [19] is a 643-task open-domain benchmark spanning 15 popular websites. We use the version curated by FARA [3], which removes infeasible tasks and updates those

with outdated information from the original dataset, resulting in 595 tasks in total[3]. For evaluation, we follow the canonical WebVoyager protocol[4] and use GPT-4o as the judge, consistent with FARA.

- **Online-Mind2Web** [49] comprises 300 longer-horizon tasks drawn from 136 popular websites across diverse domains. Compared to WebVoyager, these tasks typically require more extended multi-step reasoning and interaction sequences. For evaluation, we adopt the OSU AgentTrek protocol[5], following the standard setup used in prior work with o4-mini as the judge.
- **DeepShop** [30] is a 150-task benchmark focused on online shopping scenarios, where agents must search, compare, and select products under realistic constraints. Task success is determined using Molmo-Web's structured-output evaluation protocol [18] with GPT-4o as the judge, which programmatically verifies the correctness of the final response[6].

# C Additional Experiment Results

Table 9: Success rates (%) across 3 repeated runs. We report the mean and standard deviation for each benchmark. Avg denotes the unweighted mean across the three benchmarks.

| System | WebVoyager | Online-Mind2Web | DeepShop | Avg |
|---|---|---|---|---|
| Qwen3-VL-4B-Thinking (base) | $52.7 \pm 0.8$ | $33.7 \pm 1.5$ | $34.4 \pm 1.0$ | 40.3 |
| OpenWebRL-4B-SFT | $59.3 \pm 1.8$ | $49.7 \pm 1.2$ | $54.7 \pm 3.3$ | 54.6 |
| OpenWebRL-4B (ours) | $\mathbf{72.1 \pm 1.8}$ | $\mathbf{67.4 \pm 0.8}$ | $\mathbf{66.0 \pm 1.8}$ | **68.5** |

## C.1 Results Across Three Repeated Runs

To assess evaluation stability, we repeat the evaluation of Qwen3-VL-4B, OpenWebRL-4B-SFT, and OpenWebRL-4B across the three benchmarks. As shown in Table 9, OpenWebRL-4B achieves an average success rate of $68.5\%$, closely matching the originally reported $68.4\%$. Its per-benchmark means differ from the original results by $-2.0$, $+0.4$, and $+2.0$ percentage points on WebVoyager, Online-Mind2Web, and DeepShop, respectively. The repeated evaluations preserve the ranking of all three models on every benchmark, with OpenWebRL-4B outperforming the base and SFT models by $28.2$ and $13.9$ points on average. These results support the consistency of the gains from RL across evaluation runs.

## C.2 Task Difficulty Understanding

To characterize task difficulty and the number of steps required for task completion, we analyze Qwen3-VL-235B-Thinking's results across three online benchmarks. As shown in Figure 5, Online-Mind2Web and DeepShop exhibit lower success rates and require more steps to complete successfully (11.5 and 14.2 on average, respectively) compared to WebVoyager (8.0). Note that the step counts reported in Figure 5 are averaged over all trajectories, including failed ones. These results confirm that Online-Mind2Web and DeepShop are more challenging benchmarks, demanding longer action sequences and yielding lower success rates. The stronger performance of OpenWebRL-4B on precisely these harder benchmarks (Table 2) therefore provides stronger evidence for the effectiveness of our training framework.

## C.3 Ablations on MM-GRPO Designs

**Online RL benefits most from a balanced SFT warm start.** We first study how supervised warm-starting affects subsequent online RL. Figure 6 compares MM-GRPO training from four initial 4B policies: the base model without SFT, a lightweight SFT model trained on 0.4K trajectories for 1

[3] https://github.com/microsoft/fara/blob/44908264c810d3806365c6aab63a43c2d52a8057/webeval/data/webvoyager/WebVoyager_data_08312025.jsonl#L4

[4] https://github.com/MinorJerry/WebVoyager/blob/main/evaluation/auto_eval.py

[5] https://github.com/OSU-NLP-Group/Online-Mind2Web/blob/main/src/methods/agenttrek_eval.py

[6] https://github.com/allenai/molmoweb/blob/main/benchmarks/judges/deepshop_judge.py

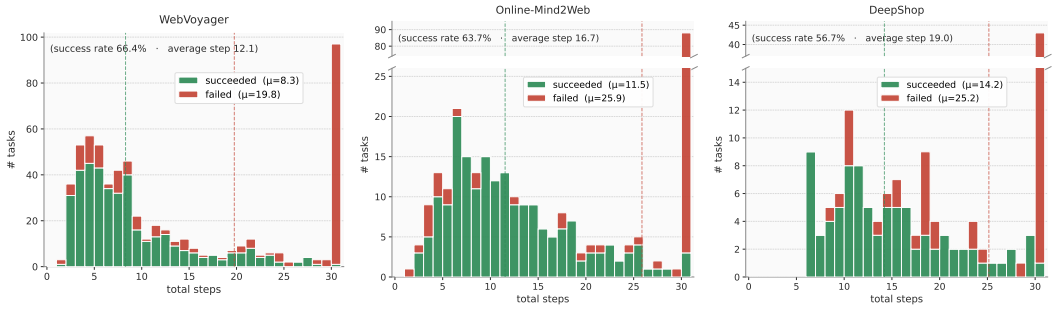



Figure 5: Trajectory length (total steps) distribution of Qwen3-VL-235B-Thinking.

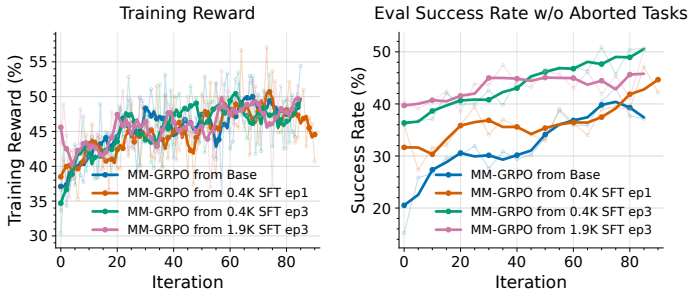



Figure 6: Comparison of MM-GRPO training under different supervised warm-start initializations.

epoch, our default SFT model trained on the same 0.4K trajectories for 3 epochs, and a larger-data SFT model trained on 1.9K trajectories for 3 epochs. The 1.9K trajectories are collected with the same teacher rollout pipeline, but retain a broader and less aggressively curated task distribution from WebGym. The results show that supervised initialization is crucial, i.e., all SFT initializations outperform the base model initialization, but *stronger warm start does not necessarily lead to better online RL*. The 1.9K/3-epoch initialization underperforms the default setting after online RL. Although it starts from a relatively strong policy, its improvement quickly saturates, and its final evaluation performance remains below that of the 0.4K/3-epoch initialization. We hypothesize that heavier imitation training, especially on a less curated task distribution, may reduce policy plasticity or bias the policy toward behaviors that are less amenable to online refinement. In contrast, the default initialization gives the policy enough competence to explore effectively while still allowing RL to further adapt it through online trial and error.

**Trajectory dynamic sampling improves training stability and optimization efficiency.** As shown in Figure 7(a) and (b), removing trajectory-level dynamic sampling leads to noticeably less stable training dynamics in both training reward and evaluation success rate. With dynamic sampling, training collects a fixed number of effective groups by filtering out zero-reward-variance groups, whereas the variant without dynamic sampling always collects a fixed number of groups regardless of reward variance. As a result, the effective batch size can vary significantly across iterations without dynamic sampling, leading to noisier policy updates and less stable optimization. In contrast, dynamic sampling stabilizes training and enables faster performance improvement in MM-GRPO.

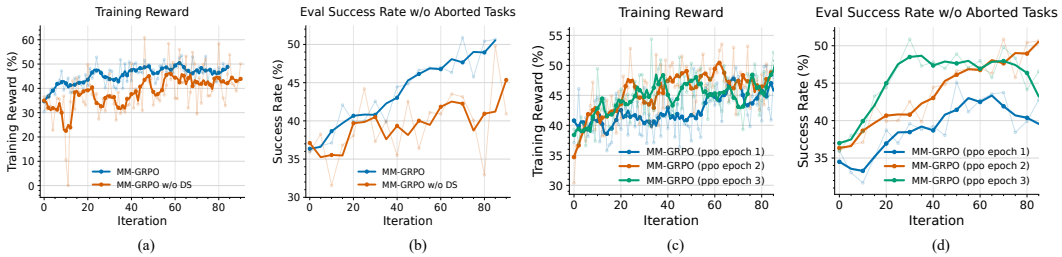



Figure 7: Ablation study on (a)(b) dynamic sampling (DS) and (c)(d) PPO epoch for MM-GRPO.

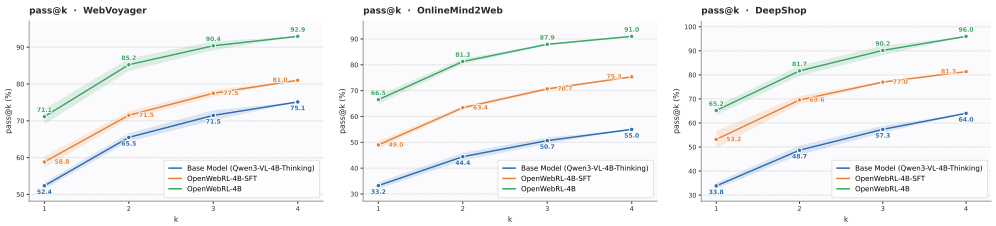



Figure 8: Pass@k performance of OpenWebRL-4B and baselines on three online benchmarks.

**Moderate PPO epochs provide a balance between data efficiency and stable optimization.** PPO epochs control how many times the policy is updated on the same rollout batch before collecting new trajectories, creating a trade-off between data efficiency and off-policyness. As shown in Figure 7(c) and (d), using only one PPO epoch leads to slower optimization and worse evaluation performance, suggesting insufficient updates for each rollout batch. Increasing the PPO epochs to 2 substantially improves both training reward and evaluation success rate, achieving the best overall performance. However, increasing the PPO epochs to 3 results in less stable evaluation behavior: although training rewards remain high, the evaluation success rate peaks early and later declines. This suggests that excessively reusing the same rollout data increases off-policyness and can lead to over-optimization, while a moderate number of PPO epochs better balances sample efficiency and stable online RL training.

### C.4 Test-time Scaling

We further study whether visual web agents can benefit from additional test-time computation. Specifically, we evaluate *pass@k*, where each task is attempted with $k$ independent rollouts and is counted as successful if at least one rollout succeeds. We use $k \in \{1, 2, 3, 4\}$ and keep the inference setup fixed across models, with a maximum rollout length of 30 steps. Figure 8 compares the base model (Qwen3-VL-4B-Thinking), OpenWebRL-4B-SFT, and OpenWebRL-4B on WebVoyager, Online-Mind2Web, and DeepShop. The results show that live-web agent performance can be scaled by sampling multiple independent attempts. However, **OpenWebRL-4B exhibits a consistently stronger pass@k curve than both the base and SFT models across all three benchmarks.** This indicates that online RL improves not only single-rollout success, but also the probability that repeated attempts discover a successful trajectory. Notably, OpenWebRL-4B achieves over $90\%$ pass@4 official success rate across the three live-web benchmarks, substantially outperforming MolmoWeb-8B [18], which reports around $60\%$ pass@4 success rate on Online-Mind2Web. The strong pass@k scaling suggests that OpenWebRL-4B learns not only more accurate individual actions, but also a richer distribution of viable interaction strategies that can be exploited through multiple test-time attempts.

### C.5 Results on AgentRewardBench

To further evaluate the generalizability and effectiveness of our judge, we report results on AgentRewardBench [31]. AgentRewardBench contains 1,302 trajectories from five widely used web-agent benchmarks: WebArena (WA), VisualWebArena (VWA), AssistantBench (AB), WorkArena (Work), and WorkArena++ (Wk++). As shown in Table 10, OpenWebRL-Judge-8B achieves the best overall F1 and Recall, while maintaining competitive Precision across benchmarks. OpenWebRL-Judge-4B also performs strongly, achieving the best Precision on WebArena and WorkArena.

It is worth noting that AgentRewardBench is built on a different agent stack, introducing distribution shift in both action representation and available environment feedback. In particular, AgentRewardBench contains action types outside our action space, making some action conversions approximate. Its trajectories also lack the per-step environment feedback used by our judges, leaving only partial signals such as URL changes, propagated errors, and accessibility-tree fragments. We believe these results should be viewed as a lower-bound estimate of our performance under this out-of-distribution setting.

Table 10: Fine-grained evaluation results on AgentRewardBench. We report overall Precision, Recall, and F1, as well as per-benchmark Precision across AssistantBench (AB), VisualWebArena (VWA), WebArena (WA), WorkArena (Work), and WorkArena++ (Wk++). Rows marked with † are taken from AgentRewardBench [31] of its simple judge with final screenshot, and rows marked with * are taken from Online-Mind2Web [49].

| Judge | Overall | | | Per-Benchmark Precision | | | | |
|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | AB | VWA | WA | Work | Wk++ |
| Claude 3.7 S.† | 69.4 | 76.3 | 72.7 | 71.4 | 64.8 | 69.3 | 85.3 | 66.7 |
| GPT-4o† | 68.1 | 80.3 | 73.7 | 77.8 | 60.7 | 69.9 | 93.8 | 59.6 |
| GPT-4o Mini† | 64.5 | 78.3 | 70.8 | 80.0 | 57.4 | 66.9 | 90.3 | 54.8 |
| WebJudge (GPT-4o)* | 73.7 | 71.2 | 72.4 | 66.7 | 69.8 | 72.6 | 92.3 | 75.0 |
| WebJudge-7B* | 75.7 | 58.0 | 65.6 | 80.0 | 66.7 | 77.5 | 100.0 | 70.0 |
| **OpenWebRL-Judge-4B** | 73.2 | 67.0 | 70.0 | 75.0 | 62.2 | 78.0 | 92.6 | 70.0 |
| **OpenWebRL-Judge-8B** | 72.8 | 72.5 | 72.7 | 80.0 | 67.0 | 73.5 | 82.9 | 74.2 |

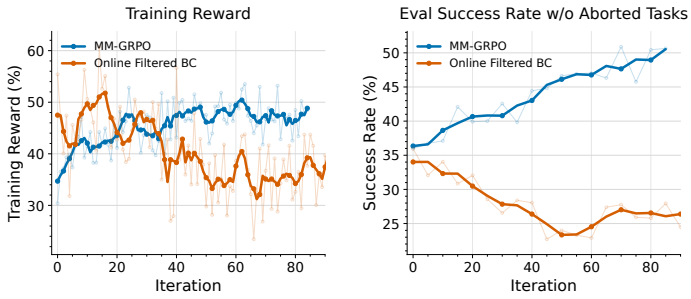



Figure 9: Comparison with online filtered behavior cloning baseline.

### C.6 Comparison with Online Filtered Behavior Cloning Baseline

Figure 9 compares MM-GRPO with an online filtered behavior cloning (BC) baseline. The BC baseline uses the same online training setup as MM-GRPO on top of the same SFT checkpoint, but only keeps successful trajectories for supervised updates, uses a fixed rollout query batch size, and does not apply dynamic sampling. Although online filtered BC initially achieves competitive training rewards, its evaluation success rate without aborted tasks steadily declines throughout training, eventually dropping below 30%. In contrast, MM-GRPO maintains stable optimization and continuously improves evaluation performance, reaching over 50% evaluation success rate, indicating that simply imitating filtered online trajectories is insufficient for robust policy improvement in open-web environments. These results highlight the importance of reinforcement learning updates, rather than pure supervised imitation, for effective online adaptation and long-horizon web interaction.

### C.7 Human Evaluation of Task Completion

To independently assess task completion and the reliability of the judge model, human experts audited 50 trajectories labeled as successful by the judge from the final policy. Of these, 41 fully satisfied the task requirements, 6 achieved partial completion but missed some requirements, and 3 executed the task incorrectly. Treating both partial completions and failures as errors yields a judge precision of $82.0\%$ $(41/50)$, with outright failures accounting for $6.0\%$ $(3/50)$ of judge-labeled successes.

To examine whether false positives become more prevalent during RL training, we also audited trajectories on the same 50 tasks using an early checkpoint at iteration 5. The judge labeled 24 trajectories as successful, of which human experts confirmed 18 as fully successful, yielding a precision of $75.0\%$ $(18/24)$. Under the same strict criterion, judge precision therefore increases from $75.0\%$ early in training to $82.0\%$ at the final checkpoint. This audit provides no evidence of declining judge precision over training.

Table 11: Lexical proxies used to analyze response-length growth. The proxies are regex-based, interpretable, and not mutually exclusive.

| Proxy | What it measures | Example phrase families | Motivation |
|---|---|---|---|
| History summarization | The agent lists previous attempts, actions, failures, or exhausted sources. | "I've tried", "we already checked", "previous attempt", "so far", "multiple approaches", "all sources failed". | Long responses often contain detailed inventories of prior failures, especially near termination. |
| Blocker diagnosis | The agent diagnoses CAPTCHA, verification, access denial, or automation-related blockers. | "CAPTCHA", "Cloudflare", "403 forbidden", "verification required", "anti-bot", "automated access", "cannot solve verification". | Many long-tail failures arise from web security or anti-automation barriers. |
| Retry-plan loop | The agent considers repeated alternatives, direct navigation, or "one more" attempts. | "one final approach", "try a different search", "alternative source", "go back", "search directly", "let me reconsider". | Captures cases where the agent continues planning retries instead of terminating or taking a concise action. |
| Condition proof | The agent checks task constraints one by one or explicitly verifies that concrete constraints are satisfied. | "meets the requirements", "all criteria met", "first condition", "under $50", "within 5 miles", "gluten-free". | Verbose successful or partially successful cases often explicitly verify each user constraint. |

### C.8 Train–Test Overlap and Cross-Website Generalization

**Task-level filtering.** Starting from 292K raw tasks, our filtering pipeline removes tasks overlapping with the evaluation benchmarks and near-duplicate intents identified through similarity matching. The resulting training data contains only 0.4K SFT tasks and 2.2K RL tasks. This filtering mitigates task-level leakage by excluding identified exact and semantic overlaps.

**Generalization to unseen websites.** We further examine whether evaluation websites appear in either the SFT or RL training data. On Online-Mind2Web, 250 of the 300 tasks span 120 websites absent from both training sets. Our model achieves success rates of $70.2\%$ on seen websites and $66.4\%$ on unseen websites, a gap of only $3.8$ percentage points. Its strong performance across 120 unseen websites supports the transferability of the learned interaction skills beyond websites encountered during training.

## D Proxy-Based Analysis of Response Length

To better understand why response length increases across training iterations, we define four lightweight lexical proxies over agent responses. These proxies are not intended to be exhaustive semantic labels; instead, they provide interpretable, automatically computable indicators of common reasoning patterns observed in long responses. We apply them to assistant turns by matching precision-oriented regular expression families in the reasoning text before the first tool call. These proxies are deliberately simple and transparent. A response may match multiple proxies, and the categories are not mutually exclusive. For example, a response can both summarize failed attempts and diagnose a CAPTCHA blocker. We therefore use these proxies as descriptive measurements rather than ground-truth semantic labels. We summarize those proxies in Table 11.

All regular expressions are evaluated case-insensitively. A step is marked as proxy-bearing if it matches any pattern from one of the four proxy families. We also separately track responses with none of the four proxies as a baseline. This no-proxy baseline shows smaller length growth than proxy-bearing responses, suggesting that these four patterns account for a meaningful portion of the observed per-step verbosity increase.

In our analysis, the proxies already appear in early iterations, indicating that **later training does not introduce entirely new reasoning modes. Instead, later iterations tend to amplify these patterns when they occur.** In particular, history summarization and retry-plan loops are associated with longer

responses and heavier response-length tails. Blocker diagnosis captures common failure states caused by CAPTCHA or access restrictions, while condition proof explains why even valid or successful responses may become more verbose: the agent increasingly justifies its answer by checking task requirements explicitly.

**Exact matching patterns.** For reproducibility, we include the exact patched proxy definitions used in the analysis below. We use a compact style to keep long regular expressions readable.

```
HISTORY_SUMMARIZATION = [
    r"\b(?:i(?:'ve| have)|we(?:'ve| have))\s+(?:already\s+)?"
    r"(?:tried|attempted|checked|searched|looked at|visited|used)\b",
    r"\b(?:i|we)\s+already\s+"
    r"(?:tried|attempted|checked|searched|looked at|visited|used)\b",
    r"\b(?:i|we)\s+had\s+(?:already\s+)?"
    r"(?:tried|attempted|checked|searched|looked at|visited|used)\b",
    r"\b(?:previous|earlier|last)\s+"
    r"(?:attempt|search|approach|step|site|page)\b",
    r"\b(?:so far|up to this point),?\s+(?:i|we)\s+(?:have|'ve|had)\b",
    r"\b(?:tried|attempted|checked|searched|looked at|visited|used)\b"
    r"[^.!?\n]{0,80}\b(?:multiple|several)\s+"
    r"(?:attempts|approaches|searches|sites|websites|sources)\b",
    r"\b(?:multiple|several)\s+"
    r"(?:attempts|approaches|searches|sites|websites|sources)\b"
    r"[^.!?\n]{0,80}\b(?:failed|blocked|led to|resulted in|did not|"
    r"didn't|couldn't|cannot)\b",
    r"\b(?:all|every)\s+(?:attempt|approach|site|source|path|option)s?\s+"
    r"(?:has|have)\s+(?:failed|led to|resulted in)\b",
    r"\b(?:exhausted|tried)\s+(?:all|every|multiple|several)\s+"
    r"(?:options|approaches|paths|sources|sites)\b",
]
```

Listing 1: History summarization proxy patterns.

```
BLOCKER_DIAGNOSIS = [
    r"\b(?:captcha|recaptcha|cloudflare|access denied|403\s+forbidden|"
    r"robot check)\b",
    r"\baccess\s+(?:is\s+)?forbidden\b",
    r"\b(?:blocked|blocking|unable to access|cannot access|can't access|"
    r"not accessible)\b",
    r"\b(?:(?:security|bot|human)\s+verification|verification\s+"
    r"(?:challenge|required|page|screen|check))\b",
    r"\b(?:security check|security measure|anti[- ]bot|anti[- ]automation)\b",
    r"\b(?:automated access|automated behavior|bot detection|"
    r"detected automated)\b",
    r"\b(?:requires?|needs?)\s+human\s+"
    r"(?:interaction|verification|input)\b",
    r"\b(?:cannot|can't|unable to)\s+(?:interact with|solve|complete)\s+"
    r"(?:the\s+)?(?:captcha|recaptcha|verification)\b",
]
```

Listing 2: Blocker diagnosis proxy patterns.

```
RETRY_PLAN_LOOP = [
    r"\b(?:one|a)\s+(?:final|last)\s+(?:approach|attempt|try|option)\b",
    r"\b(?:try|attempt|use|take)\s+(?:a\s+)?"
    r"(?:different|another|alternative|new)\s+"
    r"(?:approach|search|query|site|source|path|method)\b",
    r"\b(?:need|needs|needed|should|could|might|may|will|would|try|use|"
    r"take|switch to|look for|search for)\b[^.!?\n]{0,50}\b"
    r"(?:different|alternative|another|new)\s+"
    r"(?:approach|search|query|site|source|path|method)\b",
    r"\b(?:different|alternative|another|new)\s+"
    r"(?:approach|search|query|site|source|path|method)\b"
    r"[^.!?\n]{0,50}\b(?:is needed|may work|might work|could work|"
    r"should work|would help|to try|to search|to use)\b",
    r"\b(?:go back|return)\s+(?:and|to)\s+(?:try|search|look|navigate)\b",
    r"\b(?:navigate directly|search directly|try searching directly|"
    r"search more broadly)\b",
    r"\b(?:let me|i should)\s+"
    r"(?:reconsider|try again|try another|switch to|use a different)\b",
    r"\b(?:rather than|instead of)\s+[^.!?\n]{0,80}\b"
    r"(?:i should|i'll|i will|let me)\s+(?:try|search|navigate|use)\b",
]
```

Listing 3: Retry-plan loop proxy patterns.

```
PROOF_VERB = (
    r"\b(?:meets?|matches?|satisf(?:y|ies|ied)|fulfills?|"
    r"qualif(?:y|ies|ied)|confirms?|verif(?:y|ies|ied)|fits?|"
    r"complies with|addresses)\b"
)
REQ_WORD = (
    r"(?:criteria|requirements|conditions|constraints|criterion|"
    r"requirement|condition|constraint)"
)
REQ_MET = PROOF_VERB + r"[^.!?\n]{0,80}\b" + REQ_WORD + r"\b"
ALL_MET = (
    r"\b(?:all|each|every)\s+(?:the\s+)?"
    r"(?:criteria|requirements|conditions|constraints)\b"
    r"[^.!?\n]{0,80}\b(?:met|satisfied|fulfilled|addressed|checked)\b"
)
ORDINAL_REQ = (
    r"\b(?:(?:first|second|third|fourth|fifth|sixth|seventh|eighth|"
    r"ninth|tenth|final|last)|\d+(?:st|nd|rd|th)?)\s+\b"
    + REQ_WORD + r"\b"
)

VALUE_PATTERNS = [
    r"\b(?:under|less than|below)\s+\$?[0-9][0-9,]*(?:\.[0-9]+)?\b",
    r"\b(?:at least|at most|no more than|fewer than|greater than)\s+"
    r"[0-9][0-9,]*(?:\.[0-9]+)?\b",
    r"\bwithin\s+[0-9][0-9,]*(?:\.[0-9]+)?\s*"
    r"(?:miles?|mi|km|kilometers?)\b",
    r"\b(?:before|after)\s+[0-9]{1,2}:[0-9]{2}\b",
    r"\b(?:model years?|years?)\s+[0-9]{4}\s*(?:-|to)\s*[0-9]{4}\b",
]
DOMAIN_PATTERNS = [
    r"\bgood with kids\b",
    r"\bgood with cats\b",
    r"\bgluten[- ]free\b",
    r"\bnut[- ]free\b",
    r"\bshort[- ]haired\b",
    r"\bdepart(?:ing|s)? from\s+[A-Z][A-Za-z .-]+",
    r"\blocated near\s+[A-Z0-9][A-Za-z0-9 .-]+",
]

def has_condition_proof(step):
    if re.search(REQ_MET, step, flags=re.I):
        return True
    if re.search(ALL_MET, step, flags=re.I):
        return True
    if re.search(ORDINAL_REQ, step, flags=re.I):
        return True

    constraints = VALUE_PATTERNS + DOMAIN_PATTERNS
    for sent in re.split(r"[.!?\n]+", step):
        has_proof = re.search(PROOF_VERB, sent, flags=re.I)
        n_constraints = sum(
            bool(re.search(pat, sent, flags=re.I)) for pat in constraints
        )
        has_value = any(
            re.search(pat, sent, flags=re.I) for pat in VALUE_PATTERNS
        )
        if has_proof and has_value:
            return True
        if has_proof and n_constraints >= 2:
            return True
    return False
```

Listing 4: Condition proof proxy patterns and composite rules.

```
def proxy_labels(step):
    labels = set()
    if any(re.search(p, step, flags=re.I) for p in HISTORY_SUMMARIZATION):
        labels.add("history_summarization")
    if any(re.search(p, step, flags=re.I) for p in BLOCKER_DIAGNOSIS):
        labels.add("blocker_diagnosis")
    if any(re.search(p, step, flags=re.I) for p in RETRY_PLAN_LOOP):
        labels.add("retry_plan_loop")
    if has_condition_proof(step):
        labels.add("condition_proof")
    return labels

def is_non_proxy(step):
    return len(proxy_labels(step)) == 0
```

Listing 5: Final proxy and non-proxy labels.

**Limitations.** Because the proxies are regex-based, they may miss paraphrases or include mild false positives. The retry-plan proxy is intentionally broad enough to recover context-anchored alternative-approach language, but this also means that some benign planning statements may be counted. The condition-proof proxy is more conservative: generic mentions of constraints are not sufficient unless they match explicit requirement-satisfaction patterns or co-occur with concrete constraints and a proof verb in the same sentence. Nevertheless, the proxy definitions are interpretable, easy to reproduce, and useful for identifying high-level mechanisms behind response-length growth.

## E Error Analysis

We manually inspect 100 failed trajectories sampled from evaluation runs of multiple OpenWebRL checkpoints on Online-Mind2Web without the Browser-Use Stealth Browser service, and categorize them into four failure types. Figure 10 summarizes the distribution.

**Access and environment issues (51%).** The largest category stems from live-web instability, including page loading failures, access restrictions, and CAPTCHA blockers. In these cases, the agent typically follows a reasonable strategy, but environmental obstacles prevent the trajectory from reaching a verifiable terminal state.

**Reasoning and knowledge limitations (27%).** The second major category reflects model-side deficiencies in planning and constraint tracking. In long-horizon tasks—particularly shopping or search tasks involving multiple simultaneous requirements such as price, color, rating, size, and product type—the agent may satisfy most constraints while overlooking one or more. A smaller share of failures arises when the task presupposes background or domain-specific knowledge that the model does not reliably possess.

**Visual grounding and interaction errors (13%).** Another common failure mode is inaccurate interaction with the page. The agent may click a nearby but incorrect element, miss a small dropdown or pagination control, or fail to notice that a filter has not been applied. These errors are more likely when multiple actions are executed before the next observation, making it harder for the agent to diagnose intermediate failures.

**Task definition and evaluation issues (9%).** The remaining failures stem from ambiguous or underspecified task instructions, with a small fraction attributed to judge errors—cases where the agent completes the task correctly but the automated judge incorrectly marks it as incomplete.

Overall, this error analysis indicates that further progress requires greater robustness to open-web instability, more reliable long-horizon constraint maintenance, improved intermediate-state monitoring, and stronger recovery mechanisms for failed interactions.




Figure 10: Distribution of failure modes based on a manual inspection of 100 failed trajectories.

# F Prompt Templates

This section presents prompt examples used in our pipeline. For rollout examples, we show one user turn and the corresponding assistant turn in the exact serialized conversation format consumed by the model. For judge examples, we provide the full template together with one concrete example constructed from a real trajectory.

## F.1 Agent Input and Output

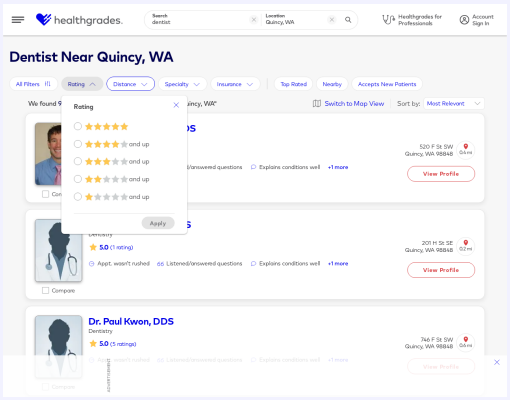
**Example Rollout Input: System Prompt**

```
<|im_start|>system
You are a GUI agent designed to operate in an iterative loop to automate browser
tasks.

# GUI Agent Policy

As an autonomous GUI agent operating on the Web Browser platform, your primary
function is to analyze screen captures and perform appropriate UI actions to
complete assigned tasks.

## Core Responsibilities

You can perform web browser interactions including mouse interactions, keyboard
interactions , navigation, tab management, task completion, and waiting.

## Input Information

At each step, you will receive the following information:
1.  Action History
2.  User Request
3.  Observation, including tab info, screenshot, and optional A11y Tree

## Output Requirements
```

```
Your output must include one <think> block and one or more <tool_call> blocks.
The
response must follow this exact structure:
<think> ... </think>
<tool_call> {"name": ..., "arguments": ...} </tool_call>

## Guidelines

Analyze the current browser state, reflect on prior actions, assess progress
toward the goal , plan the next step, and emit only valid executable tool calls.
When appropriate, multiple sequential tool calls may be emitted in one response.
The final done.response must be plain text only, with no LaTeX-style markup.

# Tools
...
<|im_end|>
```

**Example Rollout Input: User Prompt**

```
<|im_start|>user
<tool_response>
Succeed: `click` on <rect> at (237, 310) executed. Note: no visible
navigation or new
tab detected.
Succeed: `click` on <button> type=button "Apply" at (393, 557) executed. Page
navigated to https://www.healthgrades.com/usearch.

<observation>
screen size: 1280 x 1000
Current tab: 0
Available tabs:
- Tab 0 (active): https://www.healthgrades.com - Healthgrades > Universal
Search > Find a Doctor
screenshot:
<|vision_start|><|image_pad|><|vision_end|>
</observation>
</tool_response><|im_end|>
```




Screenshot attached in the raw rollout input above

**Example Rollout Output**

```
<|im_start|>assistant
<think>
The search has successfully found dentists with 5-star ratings near Quincy, WA
(zip code
98011 area).  There are exactly 2 results shown, both with 5.0 ratings.  The
first dentist, Dr.  Jay Magnuson, is located at a distance of 0.2 miles, while
the second, Dr.  Paul Kwon, is 0.6 miles away.

The user asked for the closest 5-star rated dentist to zip code 98011.  Based
on the search results, Dr.  Jay Magnuson is the closest at 0.2 miles.  This
completes the task of finding the closest 5-star rated dentist.

I should now provide the final answer with the name and location of the closest
5-star rated dentist.

</think>
<tool_call>
{"name":  "done", "arguments":  {"response":  "The closest 5-star rated dentist
to zip code 98011 is Dr.  Jay Magnuson, DDS, located at 201 H St SE, Quincy, WA
98848, which is 0.2
miles away.  He has a 5.0 rating based on 1 review."}}
</tool_call><|im_end|>
```

### F.2 OpenWebRL-Judge Input and Output

**Prompt Template for OpenWebRL-Judge**

As an evaluator, you will be presented with four primary components to assist you in your role:

1. **Web Task Instruction**: This is a clear and specific directive provided in natural language, detailing the online activity to be carried out.
2. **Agent Action History**: This is a chronological summary of the agent's observed actions across steps. Use it to understand what the agent tried to do, but do not treat it as ground truth if it conflicts with the screenshots.
3. **Result Screenshots**: This is a visual representation of the screen showing the result or intermediate state of performing a web task. Each screenshot will be annotated with an inferred step index in text.
4. **Result Response**: This is a textual response obtained after the execution of the web task.

**### TASK: {}**
**### Agent Action History:** {}
**### Result Response: {}**
**### {num} screenshots from the trajectory are attached below with inferred step indices.**

**## Guidelines**

- You DO NOT NEED to interact with web pages or perform actions.
- You SHOULD use the screenshots as the strongest evidence about the actual page state.
- You SHOULD use the action history to judge whether the agent followed the instruction and whether the final response is supported by what happened on screen.
- If the action history conflicts with screenshots, trust the screenshots.
- NOTE that the instruction may involve more than one task. Failing to complete either task should be considered unsuccessful.

- NOTE that the final response may contradict the screenshots; in that case the screenshots prevail. If the final response contains information not visible in the screenshots, you may still consider it only if it is consistent with the screenshots and action history.
- You should first explain your reasoning with explicit reference to the instruction, action history, screenshots, and final response. Then provide a definitive verdict as either `SUCCESS` or `NOT SUCCESS`.

**## Response Format:**
Return the reasoning first, then the verdict.

`Reasoning:`
Explain how you arrived at your final evaluation with explicit reference to the instruction, action history, screenshots, and final response.

`Verdict:`
`SUCCESS` or `NOT SUCCESS`

**Your verdict:**

**OpenWebRL-Judge Example Input**

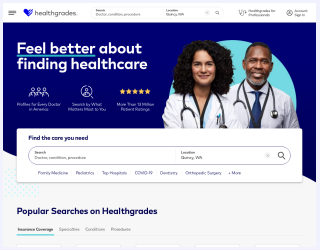
As an evaluator, you will be presented with four primary components to assist you in your role.

**### TASK:** Find the closest 5-star rated dentist to zip code 98011.

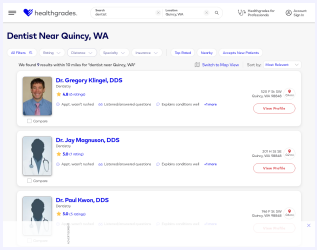
**### Agent Action History:**
Step 1: action=`[click({"point_2d": [307, 630]}); write({"message": "dentist"}); press_keys({"keys": ["enter"]})]` environment feedback: click: Succeed: `click` on search input executed; write: typed `"dentist"`; `press_keys` navigated to Healthgrades search results for dentists in Quincy, WA.
Step 2: action=`[click({"point_2d": [150, 207]})]` environment feedback: clicked the `Rating` filter with no navigation.
Step 3: action=`[click({"point_2d": [185, 310]}); click({"point_2d": [307, 557]})]` environment feedback: selected the 5-star filter and clicked `Apply`, navigating to filtered 5-star dentist results in Quincy, WA.
Step 4: action=`[done({"response": "The closest 5-star rated dentist to zip code 98011 is Dr. Jay Magnuson, DDS, located at 201 H St SE, Quincy, WA 98848, which is 0.2 miles away. He has a 5.0 rating based on 1 review."})]` environment feedback: task marked as done with the final response above.

**### Result Response:** The closest 5-star rated dentist to zip code 98011 is Dr. Jay Magnuson, DDS, located at 201 H St SE, Quincy, WA 98848, which is 0.2 miles away. He has a 5.0 rating based on 1 review.

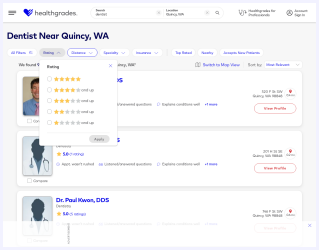
**### 3 screenshots from the trajectory are attached below with inferred step indices.**




Screenshot 1




Screenshot 2




Screenshot 3

**## Guidelines** ...

**## Response Format** ...

**Your verdict:**

**OpenWebRL-Judge Example Output**

**Reasoning:** The task instruction is to find the closest 5-star rated dentist to zip code 98011. The action history shows that the agent searched for `dentist`, opened the `Rating` filter, selected 5 stars, and returned Dr. Jay Magnuson, DDS as the answer. However, the screenshots show that the search location remained Quincy, WA rather than zip code 98011. Screenshot 2 shows dentists near Quincy, WA after the initial search, and Screenshot 3 confirms that the final 5-star results are also in Quincy, WA, with Dr. Jay Magnuson at 0.2 miles and Dr. Paul Kwon at 0.6 miles from that Quincy search center. The final response repeats Quincy, WA and does not provide evidence that the result is near zip code 98011. Since the instruction explicitly requires a dentist near 98011 and the observed trajectory instead stays in Quincy, WA, the task was not completed successfully.

**Verdict:** `NOT SUCCESS`

## G Broader Impacts

OpenWebRL releases an open framework, training data, and a 4B-parameter model for training visual web agents with online multi-turn RL. We discuss the main societal considerations below.

**Positive impacts.** The strongest visual web agents today are proprietary, with training recipes and weights closed. By releasing our full pipeline, training tasks, and OpenWebRL-4B, we lower the barrier for academic and independent researchers to study and build on online RL for visual agents. Showing that competitive performance is achievable with modest initialization data also reduces reliance on expensive human demonstrations, making this line of research more accessible to resource-constrained groups. Capable open web agents further serve as building blocks for assistive applications that help users who struggle with conventional web interfaces.

**Potential negative impacts.** More capable visual web agents lower the technical barrier for automated scraping in violation of site terms of service, circumvention of bot-detection systems, and large-scale generation of inauthentic accounts or engagement. Multi-turn agents can also compound errors over long horizons, which is particularly concerning when deployed in settings involving purchases, form submissions, or other state-changing actions. Online RL on live websites additionally places traffic load on third-party servers.

**Mitigations.** We trained on benchmark environments and task distributions designed for agent research, and rate-limited interactions during data collection. OpenWebRL-4B is released at a scale where capabilities remain bounded relative to frontier proprietary systems, limiting the marginal uplift

our release provides to a sophisticated misuse actor. We provide model and dataset cards documenting intended research use and encourage downstream users to respect site terms of service and applicable laws, and to apply human-in-the-loop confirmation or domain restrictions when deploying agents in user-facing settings. We believe open release with clear documentation better supports community work on agent safety, bot-detection defenses, and misuse evaluation than a closed release would.

## H Additional Case Studies

Figure 11 and Figure 12 show two representative long-horizon shopping tasks in which the agent must satisfy multiple user constraints through extended web interaction. In Figure 11, the agent searches for a top-selling men's Birkenstock clog in brown and size 10–10.5. In Figure 12, it searches for the cheapest black leather sofa with at least three seats and a rating of at least four stars.

These trajectories highlight several agent capabilities. First, the agent decomposes the high-level request into concrete subgoals, including navigating to the relevant product category, applying filters, inspecting candidate products, selecting product options, and locating the final purchase control. Second, the agent exhibits reflective behavior by monitoring whether an action achieves the intended state and revising its strategy when it does not; for example, it switches from ineffective filtering or sorting operations to direct product inspection, and returns to the results page after identifying an unsuitable candidate. Third, the agent maintains task-relevant information across many steps, preserving constraints such as color, size, material, rating, seat count, and price while evaluating later pages. This persistent state tracking allows the agent to reject partially matching products and complete the task only after verifying that all required attributes are satisfied.

## I Broader Impacts

OpenWebRL releases an open framework, training data, and a 4B-parameter model for training visual web agents with online multi-turn RL. We discuss the main societal considerations below.

**Positive impacts.** The strongest visual web agents today are proprietary, with training recipes and weights closed. By releasing our full pipeline, training tasks, and OpenWebRL-4B, we lower the barrier for academic and independent researchers to study and build on online RL for visual agents. Showing that competitive performance is achievable with modest initialization data also reduces reliance on expensive human demonstrations, making this line of research more accessible to resource-constrained groups. Capable open web agents further serve as building blocks for assistive applications that help users who struggle with conventional web interfaces.

**Potential negative impacts.** More capable visual web agents lower the technical barrier for automated scraping in violation of site terms of service, circumvention of bot-detection systems, and large-scale generation of inauthentic accounts or engagement. Multi-turn agents can also compound errors over long horizons, which is particularly concerning when deployed in settings involving purchases, form submissions, or other state-changing actions. Online RL on live websites additionally places traffic load on third-party servers.

**Mitigations.** We trained on benchmark environments and task distributions designed for agent research, and rate-limited interactions during data collection. OpenWebRL-4B is released at a scale where capabilities remain bounded relative to frontier proprietary systems, limiting the marginal uplift our release provides to a sophisticated misuse actor. We provide model and dataset cards documenting intended research use and encourage downstream users to respect site terms of service and applicable laws, and to apply human-in-the-loop confirmation or domain restrictions when deploying agents in user-facing settings. We believe open release with clear documentation better supports community work on agent safety, bot-detection defenses, and misuse evaluation than a closed release would.




Figure 11: Trajectory Example: Add the most top-selling Birkenstock's men clogs in brown, size 10-10.5 to my cart.

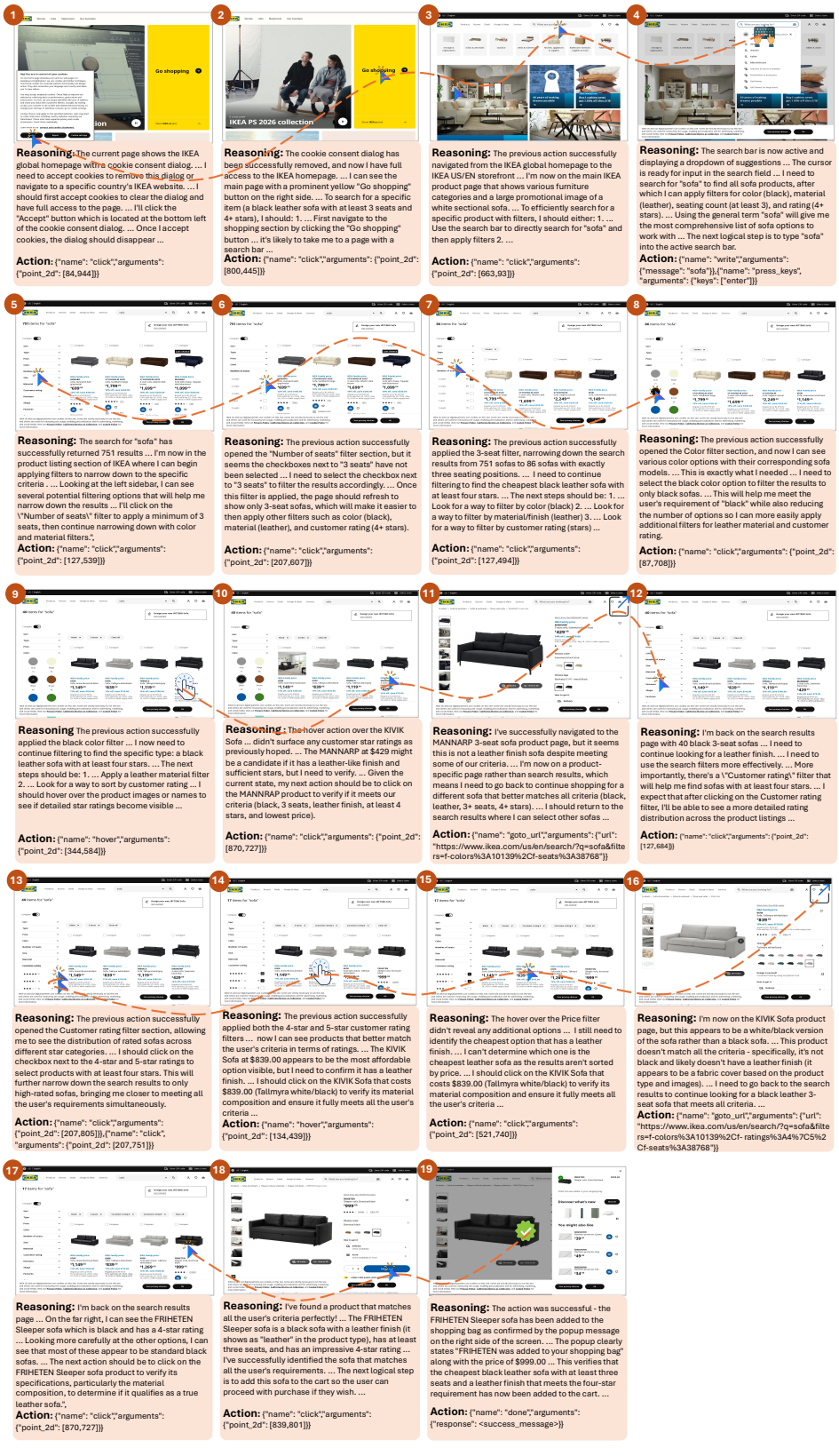



Figure 12: Trajectory Example: Add the cheapest black sofa with at least three seats, a leather finish, and at least four stars to my cart.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction introduces our framework, model, training scale, evaluated benchmarks, and empirical claims. These claims are supported by the method introduction in Section 4, the experiments in Section 5, and the main results in Table 2.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses live-web instability, CAPTCHA/blocking failures, and browser-service dependencies in Section 5 and Appendix E. In particular, Appendix E reports a manual failure analysis covering environment issues, reasoning limitations, visual-grounding errors, ambiguous tasks, and judge errors.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical results or formal theorems. The mathematical content is limited to the problem formulation, reward definition, and MM-GRPO optimization objective.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 5 describe the agent harness, data preparation, reward design, optimization objective, benchmarks, baselines, metrics, and main training configuration. Appendix A and Table 7 provide implementation details, hyperparameters, environment settings, and evaluation settings needed to reproduce or verify the main experiments.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide an anonymized link https://anonymous.4open.science/status/OpenWebRL-48CF containing the training and evaluation code for reproducing the main results. Processed task/data artifacts, model checkpoints, and judge checkpoints will be released after paper acceptance.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 specifies the base model, SFT and RL stages, training task source, evaluation benchmarks, baselines, metrics, and rollout budget. Appendix A, Table 7 and Table 8 list optimizer, learning rate, batch size, clipping parameters, context length, rollout settings, sandbox settings, and evaluation hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Due to the high cost and non-determinism of repeated live-web evaluation, we only include statistical results in Appendix C.1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 reports the approximate B200 GPU hours for the main RL run and the number of online trajectories. Table 7 and Table 8 specify GPU/backend settings, sandbox CPU and memory, concurrency, timeouts, and evaluation resources.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   The work studies training and evaluation of web agents using existing benchmarks, open-web tasks, automated model rollouts, and model-based judging, and does not involve human-subject experiments or sensitive personal data collection.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss broader impacts in Appendix I, covering both positive societal impacts (democratizing visual web agent research through open release of code, data, and models; reducing reliance on expensive human demonstrations; enabling assistive applications) and potential negative impacts (misuse for automated scraping or ToS violations, circumvention of bot-detection systems, inauthentic engagement at scale, compounding errors in state-changing actions, and server load during online training). We also describe the mitigations adopted in this release, including training only on benchmark environments designed for agent research, rate-limiting interactions during data collection, bounded model scale (4B parameters), and model and dataset cards documenting intended research use.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our release will be framework, web-agent checkpoint, judge checkpoint, and documented artifacts for research use. High-risk safeguard category in this checklist item is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use existing assets including LlamaFactory and Slime (cited in Section 4, [Apache 2.0]) as our training framework, and WebGym Tasks (cited in Section 4, [Community Data License Agreement – Permissive, Version 2.0]) for training and evaluation. We additionally build on Qwen3-VL-4B-Thinking ([Apache 2.0]). All assets are used in accordance with their respective licenses and terms of use, and original creators are properly credited via citation in Section 4.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces new model, judge, training-data, and code assets, and we will document these assets through the anonymized release link with documentation, licenses and reproduction instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or human-subject studies. Training trajectories, reward labels, and evaluations are produced by automated model rollouts and model-based judges.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human-subject research, crowdsourced participants, or participant compensation. Therefore IRB or equivalent human-subjects review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs and VLMs are central to the method and are described throughout the paper: the policy is initialized from Qwen3-VL-4B-Thinking, teacher trajectories use Qwen3-VL-235B-A22B-Thinking, GPT-4.1 is used as a training judge, and an OpenWebRL-Judge model is distilled from GPT-4.1 labels. See Section 5 for more details.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.