WEBSERV: A Browser-Server Environment for Efficient Training of Reinforcement Learning-based Web Agents at Scale

Yuxuan Lu Northeastern University *
Jing Huang Amazon
Hui Liu Amazon
Jiri gesi Amazon
Yan Han Amazon
Shihan Fu Northeastern University *
Tianqi Zheng Amazon
Dakuo Wang Northeastern University *

Abstract

Training and evaluation of Reinforcement Learning (RL) web agents have gained increasing attention, yet a scalable and efficient environment that couples realistic and robust browser-side interaction with controllable server-side state at scale is still missing. Existing environments tend to have one or more of the following issues: they overwhelm policy models with noisy context; they perform actions non-deterministically without waiting for the UI to stabilize; or they cannot scale isolated client-server containers effectively for parallel RL rollouts. We propose WEBSERV, an environment that includes 1) a compact, site-agnostic browser environment that balances context and action complexity, and 2) a scalable RL environment via efficient launching and resetting web-servers to enable scalable RL training and evaluation. We evaluate WEBSERV on the shopping CMS and Gitlab tasks in WebArena, **achieving state-of-the-art single-prompt success rates** while cutting launch latency by ∼**5**× and storage need by ∼**240**×, with a comparable memory footprint, enabling **200+** concurrent containers on a single host. [2]

1 Introduction

Web agents are autonomous systems that observe browser-rendered page state and execute the same primitive user interactions (click, type, hover, scroll, navigate) to accomplish tasks on the web. These agents are increasingly studied for applications in automated UI/UX testing [6], question answering [9], and privacy-preserving browsing [1]. Prior work has largely centered on prompting with or without demonstrations [11, 8] and fine-tuning [5], while recent studies, inspired by advances in reasoning-enabled agents in other domains [2], have begun to explore training web agents with Reinforcement Learning (RL) [12]. However, progress in RL

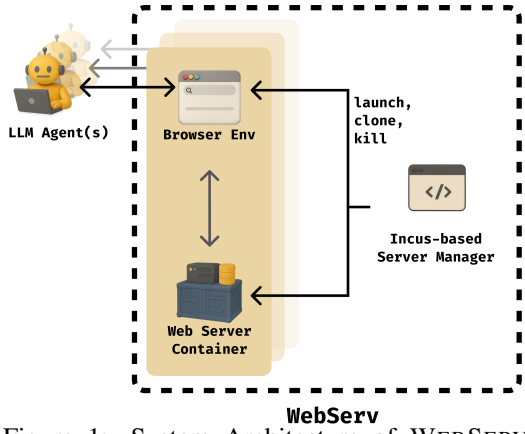

Figure 1: System Architecture of WEBSERV. Each LLM Agent interact with an isolated pair of Browser Env and Web Server Container.

*Work is done while interning or visiting at Amazon
[2]Our code is available at https://github.com/neuhai/WebServ/

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Multi-Turn Interactions in Large Language Models.

is constrained by the lack of a reproducible full-stack environment that couples robust client-side interaction with isolated server-side state.

Existing web-agent environments fail in the scenario of RL for four common reasons. First, some systems are not scalable because they require manual, per-site observation engineering or they depend on target-site features such as accessibility trees, which are often missing or inconsistent across websites [14, 13]. Second, some systems make the interface unnecessarily complex: they pass raw HTML as observations and allow arbitrary Python code for the action space, which inflates input length and makes action selection difficult (the model also needs programming ability beyond web understanding) [3]. In addition, these systems often wait for the page to fully load or wait for a specific period of time before making the next observation, which may lead to the web page being partially observed with modern single-page applications. Third, many setups do not expose simple visual hints [14, 7] that people use to judge interactivity (for example, pointer cursor on links and buttons, and text cursor on inputs). Without these cues, models often click non-interactive elements, overlook hover-only controls, and spend extra steps recovering from errors [5]. Additionally, existing environments such as WebArena [14] use Docker containers for server-side reproducibility; however, these containers are resource-intensive (launching a single shopping environment container requires approximately 6 GB of storage and about one minute to start [14]), which limits large-scale parallel evaluation and massive RL rollouts. Taken together, these limitations impede progress toward building generalizable and robust web agents.

To tackle these limitations and support future research on web agents, we propose WEBSERV for scalable and efficient evaluation and RL. First, we design and implement a fully automated parser that converts the Document Object Model (DOM) into a compact observation by removing invisible and unnecessary nodes, thereby minimizing context length. Key visual cues available to human users, for example, cursor style indicating clickability, are preserved and made available to the agent. Each interactive element is assigned a unique and semantically meaningful identifier that the agent uses to reference targets in the action space. Second, WEBSERV offers robust action execution that works with modern single-page applications by intercepting network events in the page JavaScript runtime, and returning the next observation only after the page has settled. Finally, to enable efficient large-scale experimentation, WEBSERV includes a server manager that can use any existing Docker images as web servers and elastically launch, clone, and reset isolated containers with sub-second startup and minimal resource overhead.

To measure the effectiveness of our context and action space design, we evaluate prompting-based agents on WebArena-Lite, where WEBSERV paired with Claude 4.5 achieves a **46.7%** success rate on the Shopping task, **34.3%** on the CMS task, and **40.0%** on the Gitlab task, establishing a new state-of-the-art among single-prompt agents. Furthermore, to demonstrate scalability and resource efficiency, we show that WEBSERV reduces launch latency by $\sim$**5**$\times$ and storage by $\sim$**240**$\times$ while maintaining a comparable memory footprint, enabling **200+** concurrent containers on a single host.

To summarize, our contributions are as follows:

- **Balanced web environment for text-only LLMs.** A site-agnostic interface that shortens the page description by removing invisible or irrelevant parts and exposes a small, well defined set of actions with simple parameters, while remaining compatible with arbitrary real websites or dockerized web applications.
- **Robust action execution.** An executor that finds elements by meaning, scrolls them into view, and returns the next observation only after the page has finished updating (for example, after network requests finish). It uses limited retries and clear error messages, and its semantic targeting generalizes across diverse sites.
- **Scalable environment for RL.** An *Incus* based manager that can start, clone, and reset a paired browser and web server quickly (sub-second startup), so many runs can proceed in parallel with low resource use.

2 Related Works

2.1 Web Environment Design

A key challenge in designing browser-based environments is striking a balance between the complexity of the observation space and the flexibility of the action space. For instance, WebShop [13] introduces

a simulated shopping environment where the observation space is task-specific. It consists of the current page (homepage, search results page, or product detail page) along with structured information such as the list of products. The action space is similarly constrained to a set of task-specific operations (e.g., search, purchase). While this design keeps the action space minimal yet semantically meaningful, it suffers from two major limitations: (1) the model requires a long and complex prompt to encode both the observation and action space definitions, and (2) The environment lacks generalizability across tasks and domains.

In contrast, WebAgent [3] represents the observation space using raw HTML and defines the action space as Python code that directly operates the browser. This design removes the need for manually specifying the action space and allows the model to leverage its inherent understanding of HTML elements (for example, recognizing an input field `<input type="text"/>` without requiring explicit prompt engineering). However, this approach introduces new challenges: the model must generate arbitrary code as actions, which requires not only familiarity with external libraries (such as Playwright) but also the ability to produce syntactically correct and semantically valid code.

To strike a middle ground between raw HTML and task-specific environments, WebArena [14] introduces an accessibility-tree representation that extracts only the information necessary for completing web tasks, thereby reducing the noise of raw HTML. It also defines a custom action space consisting of operations such as clicks and inputs. This approach maintains a compact context while avoiding an overly complex action space. However, it still requires website-specific adaptations (e.g., accessibility support), limiting its generalizability across domains.

From an implementation perspective, many existing frameworks re-parse the webpage after a page-load event following each action. This assumption often fails in real-world websites, where only parts of the page are refreshed, leading to incomplete or outdated observations. In addition, human users rely heavily on visual cues while browsing the web, such as cursor styles, to infer interactivity. For example, humans can easily distinguish between selectable text (indicated by "I") and clickable elements (indicated by "👆"), whereas agents frequently attempt to click plain text or other non-interactive elements. Such discrepancies highlight the need for web environments that: (1) balance the complexity of context and action spaces, (2) ensure robust action execution on dynamic, real-world websites with complex network conditions, and (3) provide essential visual cues to better align agent perception with human browsing behavior.

2.2 Web Agents

Recent advances in LLM-based Web Agents have significantly enhanced the ability of LLMs to assist with web-based tasks. Early work like WebGPT [9] enabled GPT models to interact with search engines, significantly improving question-answering performance. Subsequent systems, including WebVoyager [4], LASER [8], and WebAgent [3], extended LLM capabilities to multimodal interaction, complex state space navigation, and long-horizon planning. Claude's Computer-use API[3] further demonstrated precise, general-purpose control over user interfaces beyond web browsers.

Recently, researchers have also begun to explore fine-tuning and reinforcement learning (RL) approaches for developing web agents. **(author?)** [5] propose fine-tuning LLM-based web agents on large-scale online shopping behavioral data to improve their ability to simulate human users. Building on recent progress in enhancing reasoning capabilities of LLMs through RL, WebAgent-R1 introduces an online training paradigm for web agents with reinforcement learning. However, in their setup, multiple agents interact with a shared web server instance during the rollout phase, which can lead to unstable training. For example, one agent may add an item to the shared shopping cart, while another agent unexpectedly observes a product appearing in the cart, resulting in inconsistent training signals. The RL community has highlighted the need for scalable and efficient solutions to manage server containers for large-scale RL training.

[3] `https://www.anthropic.com/news/3-5-models-and-computer-use`

3 WEBSERV

3.1 System Overview

Existing web-agent environments either overwhelm models with excessive and noisy HTML context, rely on handcrafted task-specific interfaces that do not generalize across domains, or execute actions nondeterministically without accounting for dynamic page updates and network latency. Moreover, containerized environments such as WebArena provide reproducibility but remain resource-intensive and slow to launch, making them unsuitable for large-scale RL rollouts. To address these challenges, we propose WEBSERV, a full-stack framework that integrates: (1) a compact yet fully automated context and action space derived from the browser DOM, (2) a robust action execution backend that synchronizes observations with UI and network quiescence, and (3) an efficient and scalable manager of server containers capable of supporting massive parallel RL training.

3.2 Environment

3.2.1 Observation Space

A central challenge identified in prior work is that raw HTML observations overwhelm models with excessive and noisy content, while handcrafted task-specific abstractions (for example, WebShop) lack generality across domains.

To address these challenges, WEBSERV employs a DOM parser that automatically reduces the page to the elements that are visible and meaningful to human users. Elements that cannot be perceived by humans, such as `<script>`, `<style>`, and media tags, as well as nodes hidden through CSS (`display:none`, `visibility:hidden`, zero opacity), zero-size boxes, and non-scrollable off-screen nodes are removed. To further reduce redundancy, nested non-semantic containers such as chains of `<div>` are flattened, and empty tags are pruned unless they correspond to interactive controls like `<input>`, `<select>`, `<button>`, or `<textarea>`.

Filtering. The parser excludes entire classes of tags that are not directly useful for task execution, including `<script>`, `<style>`, `<link>`, `<meta>`, `<noscript>`, `<template>`, `<iframe>`, and media elements such as `<video>`, `<audio>`, and `<canvas>`. In addition, elements that are invisible due to CSS properties (`display:none`, `visibility:hidden`, `opacity:0`), that render with zero width and height, or that are completely off-screen and non-scrollable are removed. Only a restricted set of safe attributes is preserved, consisting of a whitelist of HTML attributes (e.g., `id`, `name`, `value`, `placeholder`, `role`, `tabindex`) as well as any `aria-*` or `data-*` attributes.

Flattening and pruning. To reduce structural redundancy, the parser collapses trivial nesting of non-semantic containers, such as chains of `<div>` wrappers, and prunes empty elements. Exceptions are made for controls that may legitimately appear empty (e.g., `<input>`, `<select>`, `<textarea>`, `<button>`, `<img>`, `<head>`, `<title>`). Inline text nodes containing non-whitespace characters are preserved in order to retain meaningful labels.

Interactivity detection. The parser heuristically determines whether elements are interactive. Nodes are marked as clickable if they are native controls (`<button>`, `<input>`, `<select>`, `<summary>`, `<area>`), anchors with `href`, elements with explicit `onclick` handlers, or elements with ARIA roles (`button`, `link`). In addition, elements with computed cursor style `pointer` are considered clickable. Elements that are disabled (via HTML attributes or CSS `pointer-events:none`) are excluded. To identify hover-sensitive targets, we monkey-patch `addEventListener` such that whenever a node registers a hover event listener, the element is annotated with `data-maybe-hoverable=true`.

Semantic identifiers. Each interactive element is assigned a stable, human-readable semantic identifier. Base names are derived from visible text, placeholders, or tag names, normalized to short slugs. Identifiers are scoped hierarchically by parent names, and uniqueness is enforced globally by appending numeric suffixes as needed. Both the cloned stripped node and the original DOM element receive the same `data-semantic-id`, and clickable elements are additionally labeled with `data-clickable=true`.

Javascript state capture. The parser augments the stripped DOM with fine-grained state information. For text inputs, textareas, and contenteditable regions, it records the current value, whether the control can be edited, numeric values (for `type=number`), text selection ranges, and focus state. For select elements, it records the current value, the selected index, whether multiple selection is enabled, and the set of selected options. Each option is cloned with its text and value, marked as selected when appropriate, and assigned its own semantic identifier namespaced under the parent select.

Output schema. The final observation is returned as a JSON object with five components: (1) a stripped and annotated HTML snapshot, (2) a list of clickable elements identified by semantic ID, (3) a list of hoverable elements, (4) a list of input elements with their state (identifier, type, value, editability, focus), and (5) a list of select elements with their selection state and per-option identifiers. This representation removes hidden or redundant markup while retaining the cues that humans rely on for interactivity, yielding a compact, semantically aligned observation space.

3.2.2 Action Space

Another challenge is that prior systems define action spaces at extremes: WebAgent relies on arbitrary code generation in Python, which requires library-specific knowledge and produces unstable executions, while WebShop restricts the agent to narrow, task-specific actions that do not generalize beyond shopping.

We design the action space so that agents can perform the same primitive interactions as human users (click, type, hover, select, navigate, manage tabs). To reduce latency and eliminate unnecessary branching, we remove *scroll* action (the environment automatically scrolls the target element into view before execution). Targets are referenced by the stable semantic identifiers described in the observation construction (`data-semantic-id`), which makes actions robust to minor DOM changes. After each action, the executor waits for network and UI quiescence (for example, no active requests for a short idle window) before returning the next observation. This design addresses prior challenges of noisy action choices, arbitrary code generation, and nondeterministic post-action states by keeping the action set compact, interpretable, and synchronized with the rendered page.

Action definitions. All actions that operate on elements identify their targets using semantic IDs that are guaranteed unique within the current observation. Before execution, the environment validates that the target remains interactable, scrolls it into view automatically (therefore no explicit scroll action is necessary), performs the operation, and then waits for an idle period before emitting the next observation. We group the available actions into four categories:

- **A. Element-level interactions**
  - Click element: perform a click action on the target element such as a button, link, or control.
  - Hover element: move the cursor over the target element to trigger tooltips or dropdown menus.
  - Key press: send a keyboard event such as Enter, Escape, Tab, or arrow keys, optionally focusing an element first.
- **B. Form and text input**
  - Type text: enter text into an input field or editable region, optionally pressing Enter afterwards.
  - Clear input: remove all content from an input field or editable region.
  - Select option: choose a specific option from a dropdown or select menu.
- **C. Navigation and page control**
  - Navigate to URL: load a new address in the current tab.
  - Back: navigate one step backward in the browser history.
  - Forward: navigate one step forward in the browser history.
  - Refresh: reload the current page.
- **D. Tab management and task control**
  - New tab: open a new browser tab, optionally with a specified URL.
  - Switch tab: change focus to another tab by index.
  - Close tab: close an existing browser tab.
  - Terminate task: shut down the browser session and optionally submit a final answer.

3.3 Robust Action Execution

Prior environments typically enforce a fixed waiting time after each action or assume that a new page will trigger a complete reload event. Both strategies are brittle. Static sleep intervals either waste time or fail to capture late-arriving updates, while relying on full page loads is incompatible with modern single-page applications (SPAs) where only parts of the DOM refresh in response to actions. As a result, agents may observe incomplete or inconsistent states, leading to unstable behavior.

To address this issue, WEBSERV introduces a network-aware synchronization mechanism that hooks into the browser's JavaScript runtime. We intercept both `XMLHttpRequest` and `fetch` APIs to track active requests and update a global activity counter whenever a request is initiated or completed. This instrumentation enables us to detect fine-grained network activity across the page, regardless of whether the site uses traditional reloads or partial updates.

After each agent action, the environment delays the next observation until it has observed a configurable idle period (e.g., 500 ms) with no outstanding network requests. This guarantees that dynamically loaded content (such as search results, shopping carts, or dropdowns populated asynchronously) is fully rendered before being exposed to the agent. If the page fails to reach an idle state within a timeout window, the environment returns control with an explicit error state, allowing the agent or training loop to handle the failure deterministically.

By synchronizing observations with actual network quiescence rather than fixed delays or reload events, WEBSERV provides robust action execution on real-world websites that depend heavily on asynchronous content loading.

3.4 Scalable and Efficient Web Server Manager

To enhance reproducibility, recent works adopt containerized web servers. For example, WebArena distributes its environments as Docker images to ensure consistent server-side state. However, directly using Docker to manage server containers in reinforcement learning presents practical challenges. Docker is designed primarily for long-lived services, and its launch speed and resource usage are not optimized for the repeated resets required in RL. For instance, launching a single shopping environment container in WebArena requires several gigabytes of storage and tens of seconds of startup time, making large-scale rollouts infeasible.

To overcome these limitations, we implement a scalable and efficient web server manager based on *Incus*, a modern container runtime that extends Linux Containers (LXC). Incus provides lightweight system containers with advanced filesystem integration, enabling fast cloning and snapshotting. Compared to Docker's layered filesystem, which copies entire modified files to the upper layer upon each launch, Incus leverages modern filesystems (e.g., ZFS, Btrfs) that support block-level copy-on-write. This distinction is crucial for web environments such as WebArena's shopping site, where a single multi-gigabyte file is touched on each reset. With block-level CoW, only the modified blocks are duplicated, dramatically reducing launch latency and storage overhead. This design offers two main benefits: (1) sub-second container startup, which enables high-throughput RL rollouts, and (2) efficient memory caching, which minimizes redundant disk usage across parallel environments.

In addition to performance gains, Incus provides two further advantages. First, it retains compatibility with Docker images through OCI support, allowing existing Docker-based web environments to run unmodified. This ensures comparability with prior implementations while improving runtime efficiency. Second, Incus supports cloning of running containers, which makes it straightforward to checkpoint server state and resume from consistent snapshots. This feature facilitates reproducible experiments and efficient training pipelines, especially in scenarios where environments must be rolled back or branched during RL exploration.

Together, these capabilities enable WEBSERV to scale reproducible web environments to reinforcement learning workloads that demand thousands of parallel container rollouts.

3.5 Post-Training Transfer and Human-Centered Inspection

Although WEBSERV is designed primarily for scalable training and evaluation, we also consider the downstream transfer of trained agents to real-world browsing contexts. To bridge the gap between

Model	Type	Shopping	CMS	Gitlab
Qwen2.5-3B [12]	Single-Prompt	4.4	5.7	13.3
Llama3.1-8B [12]	Single-Prompt	8.9	5.7	10.0
Qwen2.5-32B [12]	Single-Prompt	17.8	20	20.0
GPT-4o [12]	Single-Prompt	11.1	20	10.0
GPT-4o-Turbo [12]	Single-Prompt	13.3	14.3	16.7
OpenAI-o3 [12]	Reasoning Model	33.3	45.7	46.7
Step [11]	Multi-Prompt	37.0	24.0	30.0
Llama3.1-8B + BC [12]	Fine-tuning	17.8	20	6.7
Llama3.1-8B + WebAgent-R1 [12]	Reinforcement Learning	**44.4**	**57.1**	**56.7**
Claude 3.5 sonnet + WEBSERV	Single-Prompt	24.4	22.9	30.0
Claude 3.7 sonnet + WEBSERV	Single-Prompt	37.7	28.5	40.0
Claude 4 sonnet + WEBSERV	Single-Prompt	28.9	28.6	33.3
Claude 4.5 sonnet + WEBSERV	Single-Prompt	**46.7**	**34.3**	**40.0**
DeepSeek-R1 + WEBSERV	Reasoning Model	15.5	25.7	23.3

Table 1: Comparison of models across Shopping and CMS tasks in WebArena-Lite [10].

controlled environments and deployment settings, we develop human-centered inspection tools that make model behavior transparent and verifiable.

Trajectory inspection via replayer. We provide a web-based replayer that steps through an agent's action history. Model action can be robustly reproduced on the same webpage. Researchers and practitioners can quickly identify reasoning failures, misaligned actions, or unexpected navigation patterns without manually reproducing episodes.

Real-time action preview for human oversight. To support safe transfer to real interfaces, we implement a real-time execution mode in which model actions are previewed on the webpage before being carried out. When the agent selects an element to click, type into, or hover over, the target is temporarily highlighted in the rendered page before being executed. This mechanism prevents erroneous or unsafe interactions and also allows practitioners to collect targeted feedback signals that can be used for post-training refinement.

Together, these tools enable trained agents to move beyond offline simulation and be examined, audited, and adapted for practical integration. They also facilitate the collection of additional user feedback and correction signals that can guide further fine-tuning or reinforcement learning updates.

4 Evaluation

4.1 Performance Benchmark on WebArena

As a case study, we evaluate the robustness of our browser environment on the *shopping*, *CMS* and *GitLab* tasks from WebArena [14]. Table 1 summarizes performance across prompting-based agents, reasoning models, and reinforcement learning methods. While large reasoning models such as OpenAI-o3 or RL-based agents such as WebAgent-R1 achieve strong results, they require multiple prompts or specialized training to do so. In contrast, our system WEBSERV, when paired with Claude 4 and Claude 4.5, achieves competitive performance under the strict *single-prompt* setting, significantly outperforming other single-prompt baselines. This demonstrates that the clean and semantically enriched context provided by our environment enables LLMs to perform complex tasks more effectively without relying on multi-round reasoning or task-specific fine-tuning.

4.2 System Speed and Resource Usage

We benchmark the efficiency of our container manager against a naïve Docker-based setup. All experiments are conducted on an AWS EC2 `r6id.metal` instance with 128 vCPUs and 1024 GiB

Metric	WEBSERV (Incus)	Naïve Docker
Launch speed	1.781 s	8.963 s
Storage	28.01 MiB	6.78 GiB
Memory	1.74 GiB	1.63 GiB

Table 2: Comparison of system efficiency between WEBSERV and Docker.

of memory. To warm up the disk cache, we sequentially launch 10 containers and measure the launch speed of the last 8. Table 2 reports average launch latency, storage footprint, and memory consumption when instantiating the WebArena shopping environment.

WEBSERV achieves an average launch time of 1.78,s, compared to nearly 9,s with Docker, and reduces persistent storage from 6.78,GB to only 28,MiB through block-level copy-on-write. Memory usage remains comparable across both systems, with WEBSERV showing a modest increase due to its built-in virtualization and namespace isolation mechanisms. These improvements enable a single host to run more than 200 concurrent containers, supporting high-throughput RL rollouts while drastically lowering both storage overhead and startup latency.

The slightly higher memory usage in Incus primarily results from its architecture, which maintains a lightweight virtualized environment for each container to ensure deterministic isolation between rollouts. Unlike Docker, which shares certain kernel and daemon-level resources across containers, Incus allocates separate namespaces and minimal management daemons to guarantee reproducible execution and strict process boundaries. This design choice introduces a small, predictable memory overhead per container (typically within 100–200,MiB), but the benefit is a more stable and isolated environment that eliminates cross-container interference—a crucial requirement for RL training.

In practice, the limiting factor for scaling up concurrent containers is not memory but disk throughput. Each container launch involves reading image layers, initializing filesystems, and performing copy-on-write operations, which can easily saturate I/O bandwidth when using traditional Docker layer storage. In Docker's overlay-based storage model, every new container instantiates a fresh upper layer and writes several gigabytes of duplicated filesystem content to disk (in our measurement, approximately 6,GiB per launch), resulting in high latency and poor parallelism when multiple containers are started concurrently. These redundant writes not only prolong startup time but also create contention across concurrent rollouts, since all containers compete for limited write bandwidth.

WEBSERV mitigates this bottleneck by leveraging ZFS-backed block-level snapshots and incremental cloning, which allow new containers to be created almost instantaneously without duplicating data. In our Incus-based implementation, only around 28,MiB of new data is written during each launch, primarily for container-specific metadata and ephemeral state, while the majority of disk operations are read-only. Because multiple containers share the same base image, read operations are heavily cached by the host kernel and ZFS buffer pool, further improving launch efficiency under parallel workloads. This design minimizes write amplification, avoids redundant disk I/O, and scales efficiently across hundreds of simultaneous rollouts.

As a result, container startup time becomes decoupled from disk speed and dominated instead by lightweight metadata transactions. Even under heavy parallel initialization, WEBSERV maintains near-constant launch latency and consistent runtime performance, effectively transforming the primary scalability bottleneck from a disk-bound operation into a memory-cached snapshot lookup.

5 Discussion

5.1 Supporting RL training at scale

WEBSERV couples realistic browser side interaction with controllable server side state at scale. Concurrency is constrained mainly by disk throughput (not memory). Using ZFS based block level snapshots and copy on write, WEBSERV achieves near constant time launches and reproducible resets while persisting only 28,MiB per container and reaching an average launch time of 1.78,s versus about 9,s for Docker, enabling more than 200 concurrent containers on a single host. The

modest extra memory in Incus is acceptable because post launch access is read dominant and is cached by the kernel and the ZFS buffer pool. On the browser side we expose a compact site agnostic observation and action interface that flattens trivial containers and prunes empty nodes while preserving informative text, and we execute actions with a deterministic protocol that waits for visual and network quiescence, with one isolated application server per rollout for reproducible trajectories.

5.2 Richer operations for RL algorithms

Beyond baseline Docker semantics, WEBSERV exposes operations that are directly useful for learning algorithms. A running container can be snapshotted and cloned with block-level efficiency, which allows branching from an identical environment state without reinitializing the stack. This makes sub rollout sampling practical at scale (for example, branching at a decision point to explore multiple action proposals in parallel), supports speculative execution and top k expansions, and enables repeated what-if counterfactual trials from a common checkpoint. Fast restore to a named snapshot provides deterministic retries and fair comparisons across policies, while lightweight resets avoid the work that causes cold start effects. Together, these operations reduce variance from environment drift and make evaluation and ablation studies more reliable.

6 Conclusion

We introduced WEBSERV, a full-stack browser server environment designed for scalable reinforcement learning with web agents [4] Our system tackles three persistent challenges in prior environments (overly noisy or task-specific observation spaces, nondeterministic action execution under dynamic web conditions, and inefficient server container management). By combining a DOM parser that produces compact and semantically aligned observations, a network-aware executor that synchronizes with UI and network quiescence, and an Incus-based container manager that enables sub-second launches and block-level copy on write, WEBSERV provides both robustness and scalability. In addition, WEBSERV can target arbitrary websites (users can plug in any real online site or a Dockerized site) while isolating each rollout in its own server context and preserving realistic client behavior.

Empirical results on WebArena show that WEBSERV supports state-of-the-art single prompt performance while reducing storage requirements by more than two orders of magnitude and enabling 200+ concurrent containers on a single host. These advances make large-scale RL rollouts with realistic web interfaces both practical and reproducible.

Looking forward, we hope WEBSERV will serve as a foundation for research on web agents that goes beyond prompting, enabling reproducible evaluation and efficient training for increasingly capable RL based systems.

Limitations

While WEBSERV provides a scalable and efficient environment for training and evaluating RL-based web agents, it has several limitations that open avenues for future work.

First, our current design assumes text-only agents that operate on a stripped DOM representation. Visual signals are limited to metadata such as cursor styles, but agents cannot directly perceive the rendered page. As a result, tasks that require reasoning about spatial layout, color, images, or other visual modalities remain unsupported. Extending the environment with pixel-based or multimodal observations would allow agents to leverage visual context alongside structured DOM information. Second, although our parser produces a compact and semantically enriched HTML observation, it does not retain visual layout cues. For example, when items are arranged in a grid, the observation exposes them as a flat list without indicating how many items appear per row or whether line wraps occur. This omission simplifies the representation but removes structural signals that humans rely on to reason about grouping, alignment, and spatial organization.

[4] We used AI technologies to help with code writing and paper writing. For paper writing, the usage of AI was solely for writing assistants.

References

[1] Chaoran Chen, Weijun Li, Wenxin Song, Yanfang Ye, Yaxing Yao, and Toby Jia-Jun Li. An Empathy-Based Sandbox Approach to Bridge the Privacy Gap among Attitudes, Goals, Knowledge, and Behaviors. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pages 1–28, New York, NY, USA, May 2024. Association for Computing Machinery.

[2] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, January 2025.

[3] Izzeddin Gur, Hiroki Furuta, Austin V. Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. In *The Twelfth International Conference on Learning Representations*, October 2023.

[4] 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, June 2024.

[5] Yuxuan Lu, Jing Huang, Yan Han, Bingsheng Yao, Sisong Bei, Jiri Gesi, Yaochen Xie, Zheshen, Wang, Qi He, and Dakuo Wang. Prompting is Not All You Need! Evaluating LLM Agent Simulation Methodologies with Real-World Online Customer Behavior Data, June 2025.

[6] Yuxuan Lu, Bingsheng Yao, Hansu Gu, Jing Huang, Jessie Wang, Laurence Li, Jiri Gesi, Qi He, Toby Jia-Jun Li, and Dakuo Wang. UXAgent: An LLM Agent-Based Usability Testing Framework for Web Design, February 2025.

[7] Michael Lutz, Arth Bohra, Manvel Saroyan, Artem Harutyunyan, and Giovanni Campagna. WILBUR: Adaptive In-Context Learning for Robust and Accurate Web Agents, April 2024.

[8] Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, Wenhao Yu, and Dong Yu. LASER: LLM Agent with State-Space Exploration for Web Navigation, February 2024.

[9] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-assisted question-answering with human feedback, June 2022.

[10] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning, January 2025.

[11] Paloma Sodhi, S. R. K. Branavan, Yoav Artzi, and Ryan McDonald. SteP: Stacked LLM Policies for Web Actions, April 2024.

[12] 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, May 2025.

[13] Shunyu Yao, Howard Chen, John Yang, and Karthik R. Narasimhan. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In *Advances in Neural Information Processing Systems*, October 2022.

[14] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents, April 2024.