DynamicWebArena: Evaluating Browser Agents in Evolving Full-Stack Environments

Jiaheng Lu
University of Pennsylvania

Xian Wang
University of Pennsylvania

Shrishti Roy
University of Pennsylvania

Colin Zhao
University of Pennsylvania

Sidharth Sankhe
University of Pennsylvania

Chuyue Wang
University of Pennsylvania

Xuting Liu
University of Pennsylvania

Spyros Pavlatos
University of Pennsylvania

Linh Phan
University of Pennsylvania

Vincent Liu
University of Pennsylvania

Abstract

Existing browser-use benchmarks largely assume a classical, static web architecture. In contrast, today's web relies heavily on dynamic, stateful, and temporal behaviors driven by asynchronous requests, persistent server communications, and real-time user interactions. Consequently, current agents struggle to handle the non-deterministic, partially observable nature of modern web applications. This exposes fundamental flaws not just in the models themselves, but in current agent architectures and prompting strategies.

To address these deficiencies, we introduce DynamicWebArena[1], a novel browser-use benchmark that challenges web agents with complex, full-stack environments. DynamicWebArena encompasses diverse, fully operational applications such as online bidding, stock trading, and algorithmically driven social media. While emulating the dynamic impact of external, time-sensitive events, the benchmark remains entirely self-hosted, deterministic, and reproducible. Our experiments demonstrate a significant performance gap, with current agents GPT-5.4 achieving only an 6.8% end-to-end task success rate.

1 Introduction

The web is the primary front-end into much of our daily lives: we use it to find restaurants, to get to those restaurants, to connect to and ask friends to join us, and to trade stocks to fund all of these endeavors. To enable automated interaction with this infrastructure, researchers have integrated Large Language Models (LLM) into agentic browser workflows like OpenAI's Atlas [23], Perplexity's Comet [25], and Google's Gemini-integrated Chrome [11].

Alongside those autonomous agents, researchers have developed a slew of benchmarks to understand and evaluate their capabilities [7, 17, 18, 29, 37, 44]. Today's benchmarks capture a wide range of website formats and tasks, from testing common website user interface elements [29] to tasks that require visual information and multimodality [17, 18]. While some of these benchmarks involve

[1]Code, Docker images, and task suites: `https://anonymous.4open.science/r/fullstackarena_submission-126B/`.

Preprint.

releasing agents onto the real Internet [7], for practical reasons (e.g., the need for reproducibility, verifiable task solutions, and avoidance of CAPTCHAs), self-hosted benchmarks, typically comprising a handful of websites and associated tasks packaged in a Docker image, remain an important staple of modern browser-use evaluation.

Unfortunately, while today's benchmarks are thorough and diverse in their application domains and tasks, they tend to assume a classical model of the web—one where clients send HTTP requests to a server to fetch static HTML, CSS, and JavaScript content (often in a RESTful style), which the browser then parses/renders locally. Current browser-use agents assume this architecture implicitly, fetching the web page, analyzing it, and executing an action in response.

The modern web is substantially more complex. Today's web servers no longer serve solely static content; they are now general-purpose programs that exhibit dynamic, stateful, and time-dependent behavior. Likewise, communication between the browser and server is no longer limited to one-off GET and POST HTTP requests; it can also occur via asynchronous HTTP requests (e.g., `XMLHttpRequest` and `fetch`) and through persistent communication channels like WebSockets. Through these mechanisms, the content and behavior of modern websites are not just a function of individual users' actions, but also of external events, other users, and backend updates.

TikTok offers a concrete example of this complexity. A user visiting `tiktok.com` will issue a series of HTTP requests to fetch the main page and all of the content necessary to render the first video and at least four subsequent videos—as the user scrolls, the website's JavaScript will issue asynchronous prefetches of new videos using `XMLHttpRequest` and `fetch` operations. TikTok's web servers, in addition to serving the above requests, collect statistics on all user interactions (even mouse clicks on 'empty' areas of the web page) and measure dwell time on all videos. If a user shows interest in a topic, their recommendation embedding will update within 1 minute [20]. In essence, *when* an action is taken—and *how long* a user remains on a page—can directly influence future actions.

Existing browser-use agents can struggle to handle dynamic, time-dependent, and partially observable behavior. The problem is exacerbated by the common practice of stripping JavaScript and CSS—which comprise the overwhelming majority of website size—out of the model inputs. In most cases, the agents are unaware of the rich set of interactions that are taking place in the background. Thus, the issues with existing agents are not limited to the models themselves, but *the underlying architecture of the agents and how/when we prompt them*.

To quantify these deficiencies and facilitate the development of autonomous web agents, we present DynamicWebArena, a novel benchmark suite for browser agents. DynamicWebArena provides evaluations over complex web services with asynchronous state changes and temporal dependencies, in addition to the request-response workloads assumed by today's agents. Despite emulating these complexities, it simultaneously guarantees reproducibility and determinism through a combination of pseudo-randomness and procedural generation. DynamicWebArena's applications are realistic and broad, encompassing six fully operational applications, each representing a distinct domain prevalent on the Internet: online bidding, stock trading, street maps, ride-hailing, a social media and video platform enhanced with recommendation systems.

Experiment results show that existing browser agents perform relatively poorly on our benchmark, with an end-to-end task success rate of only 6.8%. We attribute the limited performance of current LLMs to their inability to actively explore and recover from failures when they try to complete complex tasks. These outcomes underscore the necessity for further development towards robust and effective agents in DynamicWebArena.

2 The Modern Web

We begin with a primer on the structure and most relevant components of the modern web, before providing an overview of how existing browser agents interact with it.

2.1 Clients: HTML, the DOM, and JavaScript

At the core of every webpage lies an HTML (HyperText Markup Language) document that defines the webpage's initial structure, content, and semantics. The HTML language itself is composed of a system of nested tags (e.g., `<div>`, `<h1>`, and `<p>`) that define (a) the elements of the page,

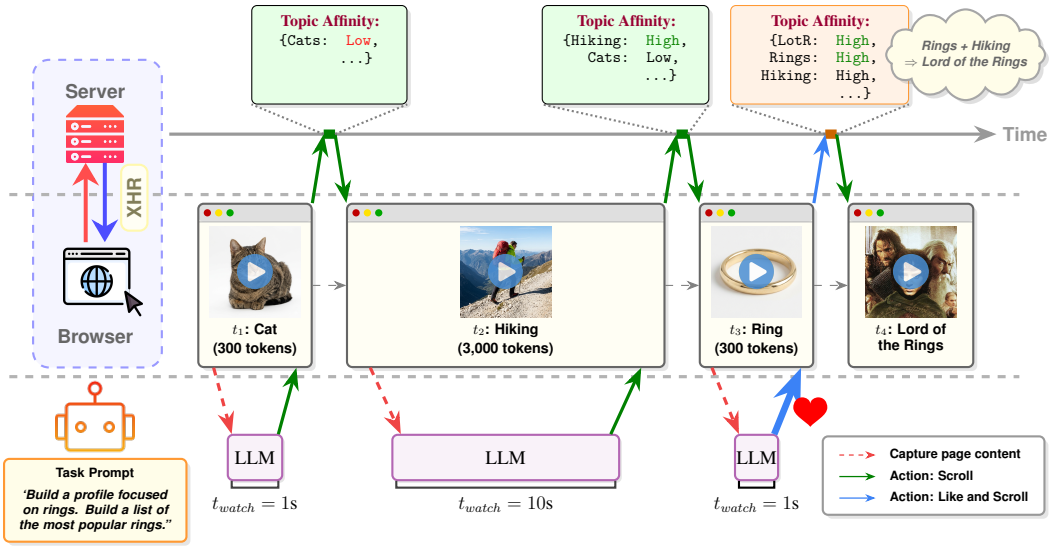


Figure 1: A Web Agent interaction with a short-form video platform. The browser communicates asynchronously with a backend server whose recommendation system estimates the user's topic affinity from a like-weighted watch time, normalizing after each scroll. Because the agent's watch time per video is dictated by how long the LLM takes to process the page and emit an action, generation latency—which scales with the number of processed tokens—becomes the de facto engagement signal. This induces a perverse coupling in which the LLM's deliberation cost is conflated with user affinity: a *hiking* video that requires extensive reasoning (3,000 tokens, $t_{watch} = 10$s) drives $p(\text{HIKING})$ very high. When the agent finally encounters a *ring* video, the unintended affinity for *hiking* may lead to unexpected side effects and a lack of sufficient *ring* videos to complete the task.

(b) their content, and (c) any associated attributes. Each page's HTML document can recursively import/reference other resources (e.g., images, CSS, and JavaScript libraries) that the browser will automatically fetch as it parses the downloaded HTML.

In the early days of the web (now commonly referred to as Web 1.0), websites were entirely static resources: browsers would fetch the page and any embedded content using the HTTP protocol, parse the contents, and render them to users. The modern web has since become much more dynamic, enabled in large part by two browser features that have made web clients into complex, fully fledged applications.

One such technology is the DOM API. The DOM is an object-oriented representation of a web page that serves as a 'live' version of the HTML. It organizes every element, attribute, and piece of text into a tree-like structure of nodes. While HTML is a static file, the DOM is a high-performance data structure living in the browser's memory.

Interacting with the DOM is embedded JavaScript code. JavaScript can be triggered by user interactions (e.g., `onclick` HTML events) or by browser events (e.g., page load completion or window resize). JavaScript programs can manipulate the DOM tree in real-time by adding, removing, or modifying nodes after the initial HTML has been loaded and parsed.

These mechanisms, together, allow web developers to control the content displayed in the browser.

2.2 Servers and Client-server Interaction

While users can typically only view the client-side code of the web, many of its more powerful features rely on server-side support and on granular interactions between servers and the browser.

These servers are, in many ways, even more complex than their browser counterparts. While HTTP is technically a stateless protocol, today's servers are built on programmable frameworks like `node.js` and maintain deep session state via cookies, server-side caches, and backend services/storage systems. Their logic is also increasingly 'time-aware.' Rate limiting, session timeouts, and time-varying content (such as dynamic pricing or expiring tokens) introduce a temporal dimension to their control

flow. Servers are thus no longer a set of predictable RESTful endpoints but rather a temporal, state-dependent system.

Communication between the browser and server is similarly flexible, with many modern features implemented using asynchronous communication protocols that allow the browser to update state without a full-page reload. One of the earliest mechanisms for implementing this type of communication is JavaScript's XMLHttpRequest (XHR) API. XHR allows JavaScript to send asynchronous requests to the web server and receive data, all without navigating the browser away from the current page. Rather, responses are handled via a callback mechanism that processes the returned data, which can then be used to dynamically update the DOM.

A classic use case of XHR is infinite scroll (e.g., on a short-form video platform like the one in Figure 1). When users scroll down after watching a video, the browser sends an asynchronous XHR request to retrieve new content—the base page is never reloaded. A seamless scrolling experience is achieved through extensive prefetching (5+ videos on TikTok). In parallel, the server-side uses its internal recommendation system to decide which video to show the user next.

While XHR is still widely used in modern web applications, the scale and complexity of today's applications have led to the rise of several other abstractions for client-server interaction.

- Fetch: JavaScript's Fetch API is often touted as the modern successor to XHR. In contrast to XHR's callbacks and event listeners (e.g., `onreadystatechange()`), the Fetch API leverages JavaScript Promises and integrates natively with the `async`/`await` design patterns.
- WebSockets: While XHR and Fetch are restricted to a classic request-response pattern initiated by the browser, WebSockets enable full-duplex, bidirectional communication over a single, long-lived TCP connection. This eliminates the overhead of HTTP headers per message and allows the server to push data to the client in real time without prompting, enabling a wide range of client-server interactions.
- Server-Sent Events (SSE): SSE is a simpler, more constrained alternative to WebSockets that sets up a unidirectional stream from the server to the client over HTTP for features like 'push' notifications and live news feeds.

These are in addition to other communication protocols with less universal support, such as WebTransport [6], a potential successor to WebSockets based on QUIC, and WebRTC [33], which allows browsers to directly exchange real-time media with other browsers in a peer-to-peer fashion.

Based on the above discussion, it is evident that the modern web has evolved from a collection of interconnected documents into a high-concurrency distributed system, where the browser is merely the surface of a deep, stateful, and time-dependent stack.

2.3 The Gap in Today's AI Browser Benchmarks and Agents

The current paradigm of browser automation largely assumes the more classic model of the web. While existing benchmarks have enabled tremendous strides in the capability of browser automation, particularly in terms of visual reasoning [17, 18, 29] and task-specific accuracy [8, 21, 35, 37, 40], they predominantly operate under the assumption of a quasi-static environment where state transitions are triggered exclusively by agent actions, and ignore the fact that the modern web is a living distributed system where the outputs are often a function of time as much as a function of input.

State-of-the-art agents, such as WebVoyager [15], the `browser-use` library [5], and various commercial solutions (e.g., Perplexity's Comet, OpenAI Atlas, and Google Chrome Auto Browser), inherit this assumption: each step involves first observing the DOM or a visual screenshot, predicting an action (e.g., a click or keystroke), and observing the resulting change in state.

The extent of temporal awareness in contemporary agents is generally limited to a primitive `wait` action. This mechanism is primarily utilized as a heuristic to ensure page resources have finished loading or to bypass simple UI transitions. These waits are typically hard-coded (e.g., 5 s) or bounded by a small maximum duration, intended to serve more as a 'pause' button than a strategic tool.

As a concrete example of the impact of a simplistic model of web interaction, consider Figure 1, where the agent is instructed to build a ring-focused profile to more efficiently research rings. The lack of precise control over decoding and dwell time can cause the server's recommendation system

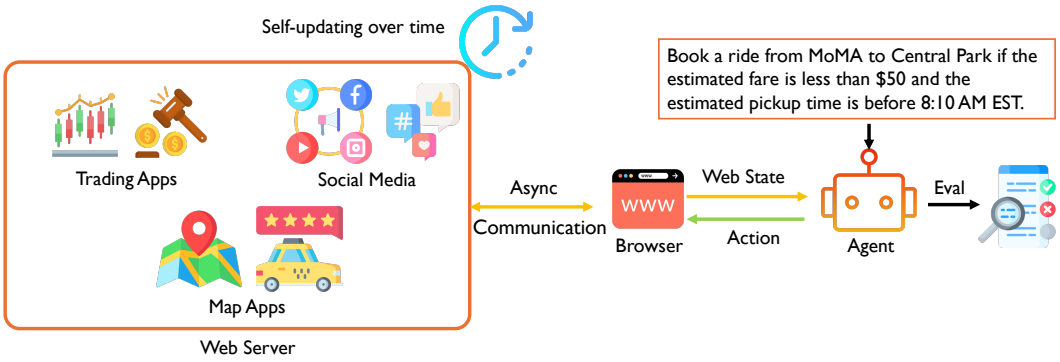


Figure 2: Architectural overview of DynamicWebArena.

to overestimate the user's interest in other topics (e.g., hiking and, by extension, Lord of the Rings), leading it to serve a different feed than the user intended. More generally, on modern platforms, delayed action or repeated attempts can lead to missing windows for critical interactions (e.g., stock trading interfaces, flash sales, or multi-factor authentication loops). Because these agents lack a model of temporal state, they cannot distinguish between a server that is busy and a server that has permanently transitioned to a new state due to a timeout.

Crucially, this gap is due to more than just limitations in the models' reasoning capabilities or context windows—the underlying architecture of today's browser agents is ill-suited to the complexities of the modern web.

3 DynamicWebArena: Toward a Richer Browser Agent Environment

This paper presents DynamicWebArena, a benchmark suite for evaluating browser agents on dynamic, asynchronous, and time-dependent web environments. Like popular existing benchmarks [8, 17, 18, 29, 44], DynamicWebArena is a standalone environment that provides realistic, reproducible, and deterministic tasks, along with evaluation tests and captured trajectories. We select auction, stock trading, video, social media, mapping, and ride-hailing websites due to their prevalence in everyday life. The overall architecture of DynamicWebArena (see Figure 2) also mirrors existing benchmark suites, with the major addition being the inclusion of complex, asynchronous server-side updates and communication. This shift, while seemingly small, introduces several important challenges and implications for DynamicWebArena's design, modeling, and task/evaluation construction, detailed below.

3.1 Ensuring Reproducibility via Procedural Determinism

To ensure consistent benchmarking and fair longitudinal comparisons, the environment must remain stable across repeated evaluations. This goal stands in direct conflict with the non-determinism and time-sensitivity we seek to emulate from the modern web—in a 'full stack' environment, reproducibility must extend beyond the initial DOM state to include the scheduled logic of the backend and user interactions.

Our solution relies on procedural generation and synchronized pseudorandomness. Rather than relying on live, volatile external APIs, DynamicWebArena simulates backend data streams—such as fluctuating stock prices, ride-hailing availability, or rush-hour congestion—using a combination of historical traces and stochastic models seeded with task-specific identifiers. For instance, in a stock-trading task, prices of securities are defined by a mixture of historical market traces and pseudorandom procedurally generated prices tied to a global simulation clock. This ensures that two different agents, when starting at the same relative timestamp, will observe the exact same environmental state transitions, regardless of the physical wall-clock time at which the test is executed or its duration.

3.2 Environment Definition

Existing browser benchmarks [18, 44] typically model agent interaction as a Partially Observable Markov Decision Process (POMDP). Unfortunately, standard POMDP fails to include any notions of time-dependence and is further built on the assumption that the environment state transitions only in response to an agent's action. In contrast, modern web servers evolve independently of the client, driven by internal timers, scheduled tasks, and asynchronous data streams.

Instead, we formalize the DynamicWebArena environment as a Continuous-Time, Partially Observable Semi-Markov Decision Process (CT-POSMDP) [32, 41]. A POSMDP generalizes a POMDP by allowing actions to take variable amounts of time, during which the underlying environment may continue to evolve independently of the agent. Formally, we define the environment as:

$$\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{P})$$

- $\mathcal{S}$ is the set of hidden environment states, including the internal server-side variables (e.g., database state, active session tokens, etc.) as well as absolute time.
- $\mathcal{A}$ is the action space. This includes standard browser-level interactions as well as the explicit `wait(`Δt`)` command.
- $\mathcal{O}$ is the observation space, which, mirroring a realistic web browsing experience, can include the current DOM tree, accessibility tree, visual screenshots rendered by the browser, and information about current and elapsed time.
- $\mathcal{T} \subset \mathbb{R}_{\geq 0}$ is the set of possible continuous time delays (sojourn times) between decision epochs.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{T} \times \mathcal{S} \to [0, 1]$ is the joint state-time transition probability function. The term $\mathcal{P}(s', \tau \mid s, a)$ defines the probability that taking action a in state s causes the system to transition to state s' after a time interval τ.

Crucially, in our formulation, the environment state s_t can change even when a is a null action, representing the *background evolution* of the server logic. Otherwise, at each step t, the agent selects an action $a_t \in \mathcal{A}$ based on its observation $o_t \in \mathcal{O}$, and the environment evolves according to the transition function $\mathcal{P}$.

Given a task specified as a natural-language intent i, we define a trajectory-level reward function

$$r(a_{1:T}, s_{1:T}),$$

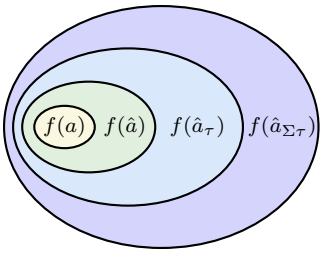
which evaluates whether the sequence of actions and resulting states satisfies the task objective. For example, in transactional tasks, the reward verifies whether the final state reflects a successful completion (e.g., placing an order), while for information-seeking tasks, it evaluates the correctness of the agent's output.

3.3 Task Creation

We taxonomize tasks by considering their necessary set of actions. These sets form a nested hierarchy, where each level strictly subsumes the previous one. The base set $f(a)$ captures tasks composed of a single webpage action. This is extended by $f(\hat{a})$ to multi-step compositions, further generalized by $f(\hat{a}_\tau)$ to include relative temporal dependencies, and finally by $f(\hat{a}_{\Sigma\tau})$ to incorporate absolute time. By spanning these taxonomies, DynamicWebArena covers a wide range of realistic behaviors that are largely absent from existing web-agent benchmarks. In total, we curate 862 tasks across six websites (see Figure 5 in the Appendix for the full distribution).

Evaluation Annotation. Each task configuration includes an `eval` block that specifies evaluation annotations—metadata defining how agent performance is automatically assessed. The annotation structure supports multiple evaluation types that can be composed together, with individual scores multiplied to produce a final result (all conditions must pass for success).

We implement four primary evaluation types. `string_match` verifies textual answers through exact matching, substring inclusion (`must_include`), exclusion checks (`must_exclude`), and numeric comparisons with inequality constraints. This type supports extracting numeric values from natural



	Actions	Time	Example
$f(a)$	Single	None	Place an order
$f(\hat{a})$	Multiple	None	If one order fails, place a second
$f(\hat{a}_\tau)$	Multiple	Rel.	Dwell 12s on post 1, 3s on post 2
$f(\hat{a}_{\Sigma\tau})$	Multiple	Abs.	Place orders at 9:00 AM and 9:05 AM

Figure 3: Task categories based on action primitives. Concentric ovals illustrate increasing complexity.

Website	Intent	Eval Implementation
Social Media	Build a profile which loves cats via any engagement method.	`top_subcategory`([pet,cat]); `weight_pct`(cat, 30)
Stock Trading	Buy 1 share AAPL at 9:30–9:31, then 2 shares at 9:31–9:32, then 3 shares at 9:32–9:33.	`verify_timed_buys`(AAPL, [(1,9:30), (2,9:31), (3,9:32)])
Ride-hailing	Book RideAppX from Empire State Building to Central Park only if fare is $25–$50 AND pickup time is 8:00–9:00 AM.	`db_check`(fare=[25,50], time=[8:00,9:00])
Auction	Find Electronics item with most bids. Wait 5 min. If price increased, outbid by $1; else bid at current price.	`bid_handled`(Electronics, wait=5min)
Video	Open video, wait until ad can be skipped, skip it promptly.	`ad_skip`(min=3s, max=6s)
Map	Get directions Empire State Building to Yonkers. Compare ETA at 6AM vs 8PM, report if increased or decreased.	`string_match`(must_include= [decreased, 30 min, 25 min])

Table 1: Example task intents and their evaluation implementations showcasing temporal constraints. Blue indicates expected values; red indicates evaluation functions.

language responses and validating multiple numbers via labeled parameters or positional indices. `url_match` confirms navigation outcomes by checking whether the final browser URL matches a reference pattern, supporting both exact matching and substring containment. `program_html` extracts webpage content via JavaScript locator expressions (e.g., `document.querySelector`) and validates the extracted text against expected patterns. This enables checking dynamic UI state that emerges from agent interactions. `api_match` queries backend systems directly to confirm that the agent's actions produced the intended state changes. We provide domain-specific helper functions for each application—such as verifying social media interactions, checking watchlist entries, or validating booking records—that execute against application databases and return structured results for comparison. For tasks involving temporal reasoning or stochastic elements, API checks can include tolerance bands (e.g., $\pm 5\%$ price widening) to account for dynamic pricing or time-sensitive conditions. The evaluator router inspects the `eval_types` list in each task configuration and instantiates the corresponding evaluator objects, which are combined in a composition class that executes all checks against the same agent trajectory. Table 1 provides representative examples of task intents and their corresponding evaluation implementations across different websites.

4 Baselines

Our main baseline results are summarized in Table 2. We evaluate all baseline agents under both capped and uncapped execution settings. The capped setting, used in prior work such as WebVoyager [15], limits each `wait` action to a maximum of 30 seconds to accommodate webpage loading. For long-running tasks, this constraint requires stronger temporal planning and may prevent the LLM from issuing overly long or incorrect wait instructions.

In the uncapped setting, there is no upper bound on the wait duration. This allows us to study the trade-off between task success and interaction efficiency in temporally evolving web environments.

Table 2: Success rate (SR) comparison of baseline models across websites.

Website	Model	SR Capped (30s)	SR Uncapped
Stock Trading	DeepSeek-V4-Pro	3.9%	5.9%
	Gemini-2.5-Pro	14.7%	33.3%
	Qwen3-VL-235B-A22B	14.7%	15.7%
	Grok-4.3	17.6%	17.6%
	GPT-5.4	22.5%	39.2%
Auction	DeepSeek-V4-Pro	0.0%	0.0%
	Gemini-2.5-Pro	12.0%	18.0%
	Qwen3-VL-235B-A22B	7.0%	14.0%
	Grok-4.3	5.0%	6.0%
	GPT-5.4	22.0%	33.0%
Ride-hailing	DeepSeek-V4-Pro	0.0%	5.2%
	Gemini-2.5-Pro	1.3%	14.3%
	Qwen3-VL-235B-A22B	0.0%	6.5%
	Grok-4.3	1.3%	5.2%
	GPT-5.4	3.9%	26.0%
Map	DeepSeek-V4-Pro	0.0%	2.3%
	Gemini-2.5-Pro	5.3%	6.3%
	Qwen3-VL-235B-A22B	2.1%	3.8%
	Grok-4.3	6.0%	14.3%
	GPT-5.4	6.8%	16.7%
Social Media	DeepSeek-V4-Pro	4.2%	2.8%
	Gemini-2.5-Pro	32.4%	25.4%
	Qwen3-VL-235B-A22B	29.6%	26.8%
	Grok-4.3	2.8%	18.3%
	GPT-5.4	23.9%	15.4%
Video	DeepSeek-V4-Pro	11.1%	11.1%
	Gemini-2.5-Pro	23.3%	23.3%
	Qwen3-VL-235B-A22B	10.0%	10.0%
	Grok-4.3	15.6%	12.2%
	GPT-5.4	17.8%	16.7%

Additionally, we compare text-only agents (DeepSeek V4) with multimodal agents, including Gemini-2.5-Pro, Qwen3-VL-235B-A22B-Instruct, Grok-4.3, and GPT-5.4.

4.1 Analysis

Does the model have a sense of the time? In our trajectory-level error analysis, we find that LLMs lack an accurate sense of time: they cannot reliably track elapsed time, current time, or the time spent during decoding. In capped mode, agents often overestimate wait duration (e.g., assuming more than 30 seconds have passed) and fail to plan temporally by issuing appropriate follow-up waits. To validate this, we conduct an ablation that removes the wait cap and allows arbitrary delays. This consistently improves performance—for example, Gemini-2.5-Pro gains 18.6% on Stock Trading and GPT-5.4 gains 22.1% on Ride-hailing. In some cases, the model does not even realize a wait is required (e.g., when an action must occur minutes later) and fails to invoke external tools to obtain timing information.

Would the external time information tool help? We conduct the time-information ablation exclusively on the ride-hailing website because it is the only site that both (1) lacks an explicit clock displayed in the UI and (2) includes $f(\hat{a}_{\Delta\tau})$ tasks requiring temporal reasoning. For reproducibility and to support temporal reasoning, we embed visible clocks in other websites (auction, stock trading, social media, map) so agents can always observe the current simulated time. This design choice means ablating time information on those sites would require removing the clock from the UI entirely, confounding the experimental setup.

5 Related Work

5.1 Web Agent Benchmarks

A wide range of benchmarks has been proposed to evaluate language agents in web environments. One major line provides controlled and reproducible environments, including self-hosted benchmarks such as WebArena [44] and VisualWebArena [18], which provide Dockerized websites for browser interaction tasks, as well as domain-specific platforms like WorkArena [8] for enterprise knowledge

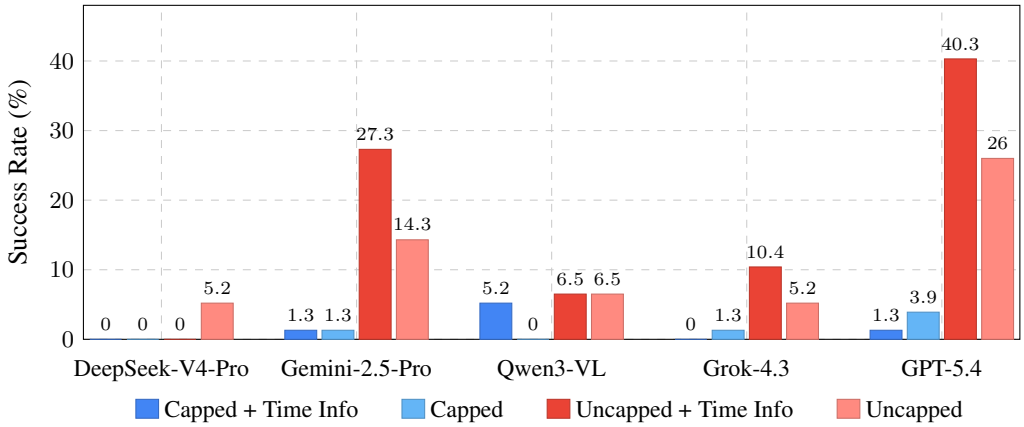


Figure 4: Ride-hailing performance across all evaluation modes. Dark colors indicate time information provided; light colors indicate no time information. Blue = capped (30s); red = uncapped.

work. Similar efforts include WebShop [37], VideoWebArena [17], and WoB [29], which introduce richer modalities. Another line evaluates agents in open-web environments, including Mind2Web 2 [12], WebLINX [21], AssistantBench [40], BrowseComp [35], and WebCanvas [24], which emphasize complex real-world tasks. Overall, while these efforts improve realism and standardization, they largely focus on static or partial aspects of the web stack, leaving the full-stack dynamic and self-updating nature of modern web applications underexplored.

5.2 Language Model Based Agents

Language-model-based agents have progressed from browser-assisted question answering [22] to a reasoning-acting interleaved framework [38], and program synthesis for task execution [14]. Leveraging visual understanding, several works [15, 16, 43] develop agents that predict interactions from rendered screenshots rather than parsed DOM trees. Complex tasks necessitate structured planning through tree search [39], self-correction [30], and model-based action simulation [13]. Recent efforts introduce hierarchical and compositional architectures for autonomous computer use [2, 3], while others incorporate reinforcement learning [27], workflow memory [34], trajectory synthesis [36], and human-inspired browsing actions [42] to improve agent learning. The temporal dynamics and partial observability of modern web applications in DynamicWebArena present unique challenges for rigorously testing and advancing these methods.

5.3 Modeling Temporal Dynamics in Interactive Environments

Interactive environments like web environments evolve over time and can be naturally modeled as Markov processes or MDPs [26, 31]. Recently, MDPs have been widely applied and extended in reinforcement learning and agent-related domains, e.g., improving time discounting functions in MDP-based RL frameworks to enhance decisions [28] and improving state transition modeling in MDPs [9, 10]. Notably, partially observable Markov decision processes (POMDPs) [1, 4] introduce observation functions to model perceptual noise and state ambiguity, preventing agents from directly accessing the true state and requiring decision-making in belief space; this framework is close to web browser interaction processes, which is also adopted by VisualWebArena [18]. More general examples include POMDP-based Mind2Web [7], WebArena [44], which applied MDP with deterministic transitions, and earlier work that constructs MDPs over DOM state spaces and optimizes policies using sparse rewards [19]. Building on these approaches, our method explicitly introduce time-aware observations into the MDP, better capturing the temporal dynamics of full-stack web environments.

6 Conclusion

In this work, we introduce DynamicWebArena, a benchmark for evaluating browser agents in dynamic, temporally evolving web environments. Comprising six self-hosted websites and 714 tasks,

DynamicWebArena captures the complexity of modern full-stack web interactions while remaining fully deterministic and reproducible.

Our evaluation reveals that current LLMs fundamentally lack an accurate sense of time—they cannot reliably track elapsed duration or coordinate with clocks. The dominant failure modes, including timing precision errors and wall-clock confusion, point to limitations in current agent architectures that treat web interaction as stateless observe-act loops. We hope DynamicWebArena will drive the development of more temporally-aware agent architectures.

References

[1] Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2011.

[2] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. In *Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*, 2025. URL https://openreview.net/forum?id=43XMKuTTK0.

[3] Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=zg5is4GJ3R.

[4] Bastian Alt, Matthias Schultheis, and Heinz Koeppl. Pomdps in continuous time and discrete spaces. *Advances in Neural Information Processing Systems*, 33:13151–13162, 2020.

[5] Browser Use. Browser use: The way AI uses the internet. https://browser-use.com/, 2026. Accessed: 2026-03-30.

[6] World Wide Web Consortium. Webtransport. URL https://www.w3.org/TR/webtransport/.

[7] 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.

[8] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

[9] Maxime Gasse, Damien Grasset, Guillaume Gaudron, and Pierre-Yves Oudeyer. Causal reinforcement learning using observational and interventional data. *arXiv preprint arXiv:2106.14421*, 2021.

[10] Maxime Gasse, Damien Grasset, Guillaume Gaudron, and Pierre-Yves Oudeyer. Using confounded data in latent model-based reinforcement learning. *Transactions on Machine Learning Research*, 2023.

[11] Google. Chrome gets new gemini 3 features, including auto browse. https://blog.google/products-and-platforms/products/chrome/gemini-3-auto-browse/, 2026. Accessed: 2026-02-23.

[12] Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jiménez Gutiérrez, Yiheng Shu, et al. Mind2web 2: Evaluating agentic search with agent-as-a-judge. *arXiv preprint arXiv:2506.21506*, 2025.

[13] Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your llm secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*, 2025.

[14] 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*, 2024. URL https://openreview.net/forum?id=9JQtrumvg8.

[15] 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. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371. URL https://aclanthology.org/2024.acl-long.371/.

[16] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14281–14290, 2023. URL https://api.semanticscholar.org/CorpusID:266210390.

[17] Lawrence Jang, Yinheng Li, Dan Zhao, Charles Ding, Justin Lin, Paul Pu Liang, Rogerio Bonatti, and Kazuhito Koishida. Videowebarena: Evaluating long context multimodal agents with video understanding web tasks. *arXiv preprint arXiv:2410.19100*, 2024.

[18] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, 2024.

[19] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

[20] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. Monolith: Real time recommendation system with collisionless embedding table, 2022. URL https://arxiv.org/abs/2209.07663.

[21] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024.

[22] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[23] OpenAI. Introducing chatgpt atlas: The browser with chatgpt built in. https://openai.com/index/introducing-chatgpt-atlas/, 2025. Accessed: 2026-02-23.

[24] 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.

[25] Perplexity AI. Comet: a personal ai assistant. https://www.perplexity.ai/comet/, 2025. Accessed: 2026-02-23.

[26] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[27] Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents, 2024. URL https://arxiv.org/abs/2408.07199.

[28] Matthias Schultheis, Constantin A Rothkopf, and Heinz Koeppl. Reinforcement learning with non-exponential discounting. *Advances in neural information processing systems*, 35: 3649–3662, 2022.

[29] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

[30] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

[31] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[32] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[33] Justin Uberti and Peter Thatcher. Webrtc 1.0: Real-time communication between browsers. W3C Recommendation, 2011. URL https://www.w3.org/TR/webrtc/.

[34] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

[35] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[36] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EEgYUccwsV.

[37] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

[38] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[39] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

[40] Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8968, 2024.

[41] Huizhen Yu. *Approximate solution methods for partially observable Markov and semi-Markov decision processes*. PhD thesis, Massachusetts Institute of Technology, 2006.

[42] Tao Yu, Zhengbo Zhang, Zhiheng Lyu, Junhao Gong, Hongzhu Yi, Xinming Wang, Yuxuan Zhou, Jiabing Yang, Ping Nie, Yan Huang, et al. Browseragent: Building web agents with human-inspired web browsing actions. *arXiv preprint arXiv:2510.10666*, 2025.

[43] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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

A Appendix

A.1 Website Implementations

Given the selected websites described in 3, we make the best attempt to reproduce the functionality of commonly used sites in a reproducible way. To achieve this, we utilized open-source frameworks for the development of the websites across various categories and imported data from their real-world counterparts. The implementation details are introduced below.

We implement the ride-hailing application through sourcing trip data from the public NYC TLC trip records, which are aggregated over the 263 official NYC taxi zone shapefiles into per-(pickup, dropoff, hour, day-of-week) travel-time and demand-derived surge tables totaling over 100k entries. We also seed 562 landmarks, transit hubs, hospitals, and airports for address autocomplete, and use OpenStreetMap for offline routing and tiles. To avoid ETAs and fares from being deterministic or memoizable for the agent, the same route returns different estimates at different NY-local times via stochastic noise on the aggregated travel-time variance and a citywide temporal demand multiplier, and a background simulator advances each booked ride through a realistic requested $\rightarrow$ assigned $\rightarrow$ arriving $\rightarrow$ in_progress $\rightarrow$ completed lifecycle.

The stock trading application uses React for frontend and Express/SQLite for backend. The site has the following features: portfolio (buy/sell shares), watchlist management, stock screener with sector/metric filters, bid-ask spread display. We source 48 real stock tickers across 12 sectors; static fundamentals (i.e. PE ratio, EPS) and previous years' prices were sourced from historical data. Time-based data is programmatically using a hash function generated to prevent data leakage. A simulated clock starts at market open (9:30 AM) and ticks in real-time.

The auction website is implemented via using React and Express/SQLite. Product listings were scraped from WebArena's shopping site. We sampled 200 listings across 9 categories. We simulated auction dynamics in the following manner: 1149 seeded bids (avg 5.7 bids/listing) get frontloaded to simulate existing bids, roughly 40 "bidding wars" (rapid $ 1 increments between 2-3 bidders), and staggered auction close times. Our simulated clock starts at 7:00 AM and ticks in real time. We use SQLite snapshots to reset database state between evaluation runs.

The video application implementation uses React/Vite on the frontend and Flask/SQLite on the backend. Video players for on-demand, live and shorts are implemented natively using HTTP Live Streaming (HLS). Video manifests and HLS segments were created via scraping public videos from YouTube and converting them to segments with the `ffmpeg` tool. The application has a database of 500 videos on demand, 30 live videos and 1000 short-form videos.

The Arenagram application uses Flask with vanilla JS for the frontend and SQLAlchemy/SQLite for the backend. The site has the following features: a personalized infinite-scroll image feed, like/save/comment/follow interactions, unified search across users, categories, and captions, profile pages with Posts/Liked/Saved/Commented tabs, and view-tracking via dwell time on each post. The SQLite database keeps track of users, posts, likes, saves, comments, follows, dwell time, and user category weights. Every action the user perform writes a row to one of these tables. The recommendation algorithm scores each post as category weight $\times$ subcategory weight $\times$ recency, where recency follows a roughly 10-hour half-life and the per-user category weights are bumped up whenever the user likes, saves, comments on, or follows posts in that category. Dwelling refers to passive view-tracking: a browser-side observer measures how long each post stays visible in the viewport and reports that duration back, which adds a smaller weight bump per few seconds of attention. This means the feed personalizes itself to a user's interests through both explicit interactions and implicit watch time. We source 1,000 real images partitioned across 7 categories and 21 subcategories like food/dessert, pet/cat, and sports/water sports. Time-based data for every post, like, comment, follow, and view is generated programmatically through a mockable clock. A simulated UTC clock is rendered live in the navbar and supports real, ticking-from-anchor, and frozen states.

The Map Arena application uses Leaflet and Pannellum with vanilla JS for the frontend and Flask backed by local Nominatim, OSRM, and a tileserver for the backend. The site has the following features: location search, $A \rightarrow B$ routing with arbitrary waypoints and alternate routes, traffic adjusted ETA cards with Light/Moderate/Heavy/Severe badges, "Leave now" and "Leave at" date and time pickers, turn-by-turn directions, and a Street View panel with forward and backward

mobility. We source 533 real equirectangular panoramas across 16 sequences mostly in Manhattan, downloaded via the Mapillary Graph API across 10 hand-picked neighborhood bounding boxes. Tiles, geocoding, and base driving durations come from local OSM-stack services. Traffic data is simulated by multiplying OSRM's base duration by a deterministic time-of-day curve that peaks at 8 AM and 5 PM, plus smaller modifiers for road type, area density, weekday vs. weekend, and a per-route variation factor. A simulated clock starts at Friday 9:00 AM ET and ticks forward in real time.

A.2 Environment reset

To ensure reproducible and fair evaluation, each environment supports a deterministic reset procedure that restores the system to a well-defined initial state. This is particularly important in our setting, where applications maintain complex, stateful, and time-dependent behaviors. Without such resets, agent performance would be confounded by residual state and temporal drift across runs.

For the ride-hailing app, state lives across three layers that reset must restore atomically: (i) a PostgreSQL/PostGIS store, (ii) an in-process simulated clock, and (iii) a Redis cache for ETA and zone snapshots. At ETL time we materialise immutable seed snapshots (`_seed_users`, `_seed_drivers`, `_seed_rides`) parallel to every mutable table, produced by a fixed-seed RNG so two ETL runs yield the same world. A single `POST /reset` endpoint then (1) truncates each mutable table and re-inserts from its `_seed_*` snapshot, (2) resets primary-key sequences so new entities start at predictable IDs, (3) rewinds the simulated clock to 8:00 AM NY-local, and (4) flushes the Redis cache to drop stale ETA rolls (a per-call `random.gauss(1.0, 0.03)` multiplier under a 30 s TTL) and cached zone listings. Every reset thus restores a byte-identical world state, giving evaluators a stable ground truth regardless of the agent's prior actions.

The finance environment is reset via a POST /api/eval/reset call issued before each task. The reset executes a sequence of SQL statements that (1) deletes all portfolio positions, transactions, and watchlist entries accumulated during the prior run, (2) restores the user's cash balance to its initial $100,000, and (3) re-inserts the fixed seed portfolio of 3 shares of AAPL purchased at $184.19. This returns the environment to a deterministic baseline in which the user's holdings, transaction history, and available capital are identical at the start of every task.

The auction environment is reset also via a POST /api/benchmark/reset call issued before each task batch. The reset executes a pre-generated SQL script that (1) deletes all bids placed during the prior run, (2) restores each listing's current price to its starting bid, and (3) re-inserts the fixed set of 1,560 seeded bids. This returns the environment to a deterministic baseline in which bid histories, current prices, and bid counts are identical at the start of every task.

The stock trading environment is reset via...

For the Arenagram app, state lives across two layers that reset must restore deterministically: (i) a SQLite store holding the mutable tables (users, posts, likes comments, saves, follows, views, ranking weights) on a Docker volume, and (ii) an in-process virtual clock whose offset/freeze state lives in module globals. The static image corpus and seed-time JSON fixtures are baked into the image and consumed by a fixed-seed RNG, so two reseeds tield the same users, posts, and interaction graph. Per task, the harness dispatches on `reset_scope`: a global reset rewinds the clock, drops and recreates every table, and reruns the deterministic seeder; a per-user soft reset wipes on only the target account's likes, saves, comments, follows, and views and rebases its interests weights to the uniform prior, leaving every other user and the seeded corpus untouched. After the wipe, an optional `clock_init` block pins the virtual clock to a task-specified timestamp so timing-sensitive ranking signals are stable across runs. Every reset thus restores an identical state — same IDs, same seeded interactions, same ranking prior, same anchored clock — regardless of the agent's prior actions.

For the Map Arena app, the only mutable server state is an in-process virtual clock anchored by default at Friday at 9:00 ET that ticks forward in real time from that anchor. The routing graph is supplied by external read-only services (OSRM, Nominatim, a tileserver) and the panorama SQLite database is bundled into the image and never written at runtime. Every per-route traffic multiplier the UI shows is a pure deterministic function of the virtual clock, route seed, and road/area modifiers, so the same (clock, seed) always yields the same ETA.

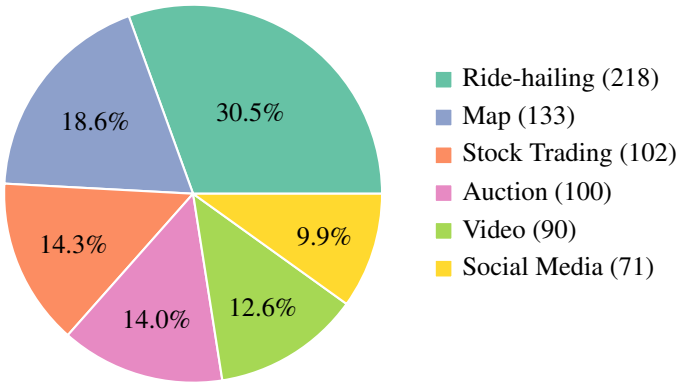


Figure 5: Distribution of 714 tasks across the six websites in DynamicWebArena. Ride-hailing and Map together comprise nearly half of all tasks due to their complex temporal and spatial reasoning requirements.

For the DyVideo app, all benchmark state is specific to a user: likes, watch times and watch history. This state evolves with website interactions, and is queried through the admin API. A reset therefore wipes the state clean for a particular user, starting with 0 watch history, neutral preferences for all shorts categories and no likes/dislikes. The shorts feed starts out with an exploratory phase of $\tilde{2}0$ shorts, before switching to a Personalized feed. The feed is set back to Exploration on reset.

A.3 Task Distribution

DynamicWebArena contains a total of 714 tasks distributed across six websites, as shown in fig. 5. Ride-hailing (218 tasks, 30.5%) and Map (133 tasks, 18.6%) together comprise nearly half of all tasks due to their complex temporal and spatial reasoning requirements. Stock Trading (102 tasks, 14.3%), Auction (100 tasks, 14.0%), and Video (90 tasks, 12.6%) each contribute roughly equal shares, while Social Media (71 tasks, 9.9%) has the smallest set focused on feed personalization and timing-sensitive interactions.

A.4 Failure reason decompositions

We categorize the task families and their dominant failure modes for each website below.

A.4.1 Ride-hailing

Family	Description
F1	Wait for stochastic price to drop below a specified cap, then book
F2	Within 3 min of login, book at the lowest fare observed across repeated price refreshes
F3	Book a ride, wait for state to flip to in_progress, then report trip duration
F4	Make two consecutive bookings separated by an N-minute wait
F5	Wait until the simulated NY wall-clock reaches a target HH:MM, then book
F6	Make three bookings; both inter-booking gaps must each be $\geq 0.8 \times N$ min
F7	Wait N minutes (no clock anchor) and then book
F8	Book such that the projected arrival lands inside a target time window

Table 3: Ride-hailing task families.

Across the five baselines on ride-hailing, the dominant gap is that none of the agents reliably executes the temporal structure each task requires (tables 3 and 4). Under the 30 s wait cap every model scores at 5.2% or below, because F6–F10 require 60 s–10 min explicit pauses that the cap structurally forbids. Removing the cap lifts GPT-5.4 to 40.3% with time information, yet even then the remaining failures expose two recurring issues: (1) wall-clock vs. simulated-clock confusion, where agents make one-shot timestamp readings and issue wait calls that land bookings before the deadline (fig. 4),

Family	Dominant failure mode
F1	Agent fails to scroll to required vehicle type (XL/Black), fabricates or misreads fare
F2	Agent books wrong ride or fails to re-check for lower fare after refresh
F3	Booking zone IDs from autocomplete don't match postcondition query
F4	Agent hallucinates confirmation numbers not visible in UI
F5	Agent books the ride too early before the target wall-clock time
F6	Request button label not updated when clicking vehicle card, or wait() fires shorter than required
F7	Long waits ($\geq$7–8 min) cause timeout or JSON validation errors
F8	App has no "schedule for later" control for target-arrival constraints

Table 4: Ride-hailing per-family dominant failure modes.

and (2) UI navigation failures, where agents fail to scroll to the required vehicle type and either fabricate fares or book the wrong ride class.

A.4.2 Arenagram

On the social media site, agents must navigate an infinite-scroll feed, interact with posts through likes, saves, comments, and follows, and manipulate a live recommendation engine that adapts to user behavior (tables 5 and 6). The dominant failure modes fall into two categories: (1) imprecise selection, where agents miscount posts or select adjacent items when targeting specific content, and (2) timing violations, where agents act outside designated time windows or miss deadlines due to the latency of scrolling and searching. Tasks requiring recommendation manipulation (F3, F6) expose a subtler issue: agents often spread engagement across multiple categories rather than concentrating on the target, preventing any single subcategory from reaching dominance.

Family	Description
F1	Scroll the feed or visit a profile, pick out a specific post/user/caption, optionally follow/-like/comment, and report a fact
F2	Scroll the feed, find the first N posts whose category tag matches, like/comment on each, and report the authors in order
F3	Dwell on posts or like enough posts until the "Your Interests" sidebar crosses a specified weight percentage, then report the weight/count/subcategory
F4	Wait for the clock to enter a single timed window and perform one or multiple actions for a specific post or user inside it
F5	Several ordered timing windows with dead zones, rate-limits, alternation, escalating actions, observe-then-act, or cross-user timed actions
F6	Use any combination of likes/saves/comments/dwells to make your dominant subcategory in the recommendation engine

Table 5: Arenagram task families.

Family	Dominant failure mode
F1	The agent was off by one when selecting posts post/user selection and reported a plausible-sounding fact without verifying it in the UI
F2	Skips a category post (takes the 2nd, 4th, 5th instead of 1st or 3rd) or double clicks a like, leaving one action missing
F3	Scrolls past the target post before the dwell timer accumulates, so the weight never crosses the threshold.
F4	Has a delayed response and acts outside of the designated $[t, t_\Delta]$ time window
F5	Loses track of which phase is active and performs an action inside a dead zone, or misses a window entirely because scrolling/searching overran it
F6	Builds up multiple categories in parallel instead of concentrating on the target, so the target subcategory never dominates

Table 6: Arenagram per-family dominant failure modes.

A.4.3 DyVideo

Family	Description
F1	Search or play videos, visit the user profile and interact with the video catalog
F2	Open a video, look at the ad countdown timer, wait for the timer to expire and skip the ad promptly
F3	Open a video with an ad with an undetermined countdown, wait for the ad to become skippable, and skip it promptly
F4	On-demand video watch timing in precise windows and recommendation algorithm training through watch times
F5	From a fresh shorts feed, steer the algorithm to make one category the dominant preference through interactions
F6	First build a preference in one category, then flip that preference to another category
F7	Deliberately suppress a category with negative interactions, especially watch time
F8	Create neutral non-target history, and then recover by focusing on another category
F9	Select an upcoming live broadcast at specific times in the future and wait until the playback starts

Table 7: DyVideo task families.

For the video site, agents perform various actions on shorts, live and on-demand feeds (tables 7 and 8). While current agents are good at navigation, parsing and control tasks, they exhibit far lower performance on temporal tasks, with the dominant failure modes being: (1) not considering their processing time or inter-step latency in timing sensitive tasks and (2) a lack of patience when considering tasks complete. For example, ad skip tasks involve the agent skipping an ad as soon as it becomes skippable. Agents routinely miss this window due to large internal processing time and show a near 0% success rate. A similar failure mode is visible in shorts, where the recommendation system state doesn't get the correct signals due to the agent spending longer on a particular short than it thinks. Lack of patience manifests in claiming shorts recommenders have been trained after just seeing one more short from a target category, which is way too early. We notice that, counterintuitively, adding a capped wait sometimes helps agents, by forcing repeated observations of state (e.g. for GPT 5.4 and Grok 4.3).

Family	Dominant failure mode
F1	No failure mode
F2	Agents did not skip the ad on time due to a lack of awareness of internal time taken. While the agents waited for the countdown timer to elapse, ads were either not skipped at all, or skipped very late
F3	Ads were not skipped at all, or skipped very late due to the observe-think-act loop being too slow to react to the ad becoming skippable
F4	Agents assumed they could act on the observed timestamp directly, but that wasn't usually the case
F5	Agents often used a simplistic "like" target, "dislike" everything else without considering watch times
F6	While the agent considered watch times, it terminated too quickly without training the flipped category enough
F7	Registering negative signals involves skipping the video with less than 10% of watch time. Failures occurred due to mismatch between observation, loop latency, and actual recorded skip time
F8	Failures occur due to imprecision, non-target shorts are too short or too long to be considered neutral watches
F9	Agents rouninely observe partial state, think its complete and declare the task as incorrectly done

Table 8: DyVideo per-family dominant failure modes.

A.4.4 Auction

On the auction site, agents must navigate category listings, track bid activity over time, and place bids at precise moments or under specific conditions (tables 9 and 10). The dominant failure modes

center on two issues: (1) timing precision errors, where agents check prices or place bids seconds off from the required wall-clock time, and (2) data extraction failures, where agents struggle to reliably scrape listing information across paginated results and either fabricate values or bid on rank-adjacent items. Tasks requiring sitewide comparisons (F5) reveal that most agents rely on visual scrolling that truncates results, while only GPT-5.4 consistently uses JavaScript DOM queries to extract complete data.

Family	Description
F1	Single-category item selection with timed or conditional bid (e.g., find item in Electronics, wait 5 min, bid if condition met)
F2	Monitor bid activity and bid on item with most growth (e.g., watch items, wait 5 min, bid on one with most new bids)
F3	User tracking with conditional bid (e.g., find items Jane Smith bid on, wait, bid based on her activity)
F4	Bid ratio computation with timed bid (e.g., compute current/starting bid ratios, find item closest to $2\times$, bid at 7:09 AM)
F5	Sitewide ranking with timed bid (e.g., rank all items by current bid across all categories, bid on 4th highest)
F6	Cross-category comparison with conditional bid (e.g., compare highest bid in Electronics vs Home & Kitchen, bid on winner)

Table 9: Auction task families.

Family	Dominant failure mode
F1	Timing precision errors — agent checked at wrong time (e.g., 7:05:07 instead of exactly 7:05) or claimed success without actually clicking Place Bid
F2	Extraction retry loops — agents attempt dozens of strategies (extract, find_elements, evaluate) without obtaining complete listing data; when successful, often bids on rank-adjacent items
F3	Data fabrication — agents report items with prices that don't match page state, or fail to navigate to the correct user's bid history
F4	Ratio miscalculation or timeout — agents compute ratios incorrectly or spend too long on extraction, missing the timed bid window
F5	Incomplete sitewide extraction — agents rely on visual scrolling that truncates results; only GPT-5.4 uses JavaScript DOM queries to extract all 260 listings in one step
F6	Category navigation errors — agents click wrong categories or miss one category entirely when comparing across multiple

Table 10: Auction per-family dominant failure modes.

A.4.5 Map

Our results on the Map benchmark show that, with the exception of DeepSeek V4, the other four models exhibit basic but very limited visual analysis capabilities in the complex map environment, with Grok 4.3 and GPT-5.4 standing out relatively. The two tables present our family taxonomy (Table 11) and the dominant failure modes (Table 12).

A.4.6 Stock Trading

On the stock trading site, agents must monitor real-time price movements, compare metrics across stocks and sectors, and execute trades at precise wall-clock times (tables 13 and 14). The dominant failure modes center on (1) timing drift, where cumulative inter-step latency causes agents to execute trades outside the valid time window, and (2) incomplete data extraction, where agents scan only a subset of stocks in a sector and miss the actual largest mover. Tasks requiring bid-ask spread checks (F5) reveal that most agents conflate the spread with individual bid or ask prices, leading to incorrect conditional decisions.

Family	Description
F1	Set the mock clock to different times, use "Leave now" to read clock-state-dependent ETAs, and compare across clock states
F2	Scan multiple departure hours within a day and report the min/max ETA or whether they cross a threshold
F3	Compare route orderings across departure times and detect the moment when the fastest route changes
F4	Cross weekday $\times$ hour interactions: compare ETAs between weekdays and weekends across multiple hours of the day
F5	Repeatedly advance the clock (via API or ticking) and analyze trends, jumps, and peaks in the resulting ETA sequence
F6	Add waypoints and compare how departure timing affects multi-leg trips differently across segments
F7	Navigate Street View panoramas and perform visual reasoning over signs, storefronts, and scene elements
F8	Watch ETAs change as the clock ticks and report the moment a travel time crosses a given threshold

Table 11: Map task families.

Family	Dominant failure mode
F1	Models lose track over long sequences, skipping a clock advance shifts every later ETA
F2	After many date/hour switches, models often pick the wrong hour even when the ETAs are read correctly
F3	Models read the highlighted route card instead of taking the minimum across all cards, mismatching the meaning of "best route"
F4	Models struggle to maintain consistent accuracy across many weekday/weekend ETA reads; state drift and accumulated reading errors corrupt the final gap comparison
F5	Multiple clock advances strain state tracking, and the added trend/peak math leaves no room for arithmetic slips
F6	Same multi-card confusion as F3, plus unstable waypoint interactions; any single wrong segment breaks the multi-leg comparison
F7	Visual perception bottleneck: fine details get misread, and non-visual models cannot attempt these tasks at all
F8	Long advance chains drop steps and corrupt every downstream threshold check

Table 12: Map per-family dominant failure modes.

A.5 Limitations and Future Work

While DynamicWebArena captures dynamic, stateful, and temporal behaviors absent from prior browser-use benchmarks, there are several limitations and work left for future.

Approximation of real-world applications. Our six applications are faithful reimplementations of widely used web services, but they cannot fully reproduce the scale, feature breadth, or proprietary backend logic of their commercial counterparts (e.g., TikTok's recommendation embeddings or Uber's surge pricing model). We trade some realism for reproducibility.

Deterministic dynamics. To enable controlled evaluation, all temporal signals, including surge multipliers, market ticks, ad countdowns, traffic curves, and recommendation scores are produced by deterministic functions of a simulated clock and seeded RNGs. Real systems are additionally driven by external events (other live users, market data feeds, breaking news) that we cannot replicate without sacrificing reset-and-replay. Agents that succeed on DynamicWebArena may still encounter behaviors outside our distribution when deployed on the live web.

Human baseline. We do not currently report a human baseline. We plan to pursue an IRB-approved human study to establish this baseline and contextualize the gap between current agents and human performance.

Family	Description
F1	Conditional buy based on comparison (e.g., compare volume/price of multiple stocks, buy the winner)
F2	Monitor and report metrics (e.g., watch AAPL and MSFT for 5 min, report which had higher intraday high)
F3	Timed trading at specific wall-clock time (e.g., wait until 9:37 AM, then execute trade)
F4	Sector-wide comparison (e.g., find which stock in Technology sector had largest % drop in intraday low)
F5	Bid-ask spread check with conditional buy (e.g., wait until 9:38 AM, check spread, buy if under $1.00)
F6	Conditional watchlist management (e.g., monitor stocks 5 min, add to watchlist only those with price increase)

Table 13: Stock Trading task families.

Family	Dominant failure mode
F1	Comparison errors — agent misreads price/volume data or compares wrong time windows; occasionally buys both stocks instead of the winner
F2	Metric extraction failures — agent reports plausible but incorrect values, or confuses intraday high with closing price
F3	Wall-clock timing drift — agent's cumulative wait undershoots or overshoots target time due to inter-step latency; executes trade outside the valid window
F4	Incomplete sector scan — agent checks subset of sector stocks and misses the actual largest mover; often reports second-largest instead
F5	Spread misinterpretation — agent reads bid or ask price instead of spread, or fails to wait until the exact specified time before checking
F6	Premature watchlist action — agent adds stocks before the observation period ends, missing price changes that would have disqualified them

Table 14: Stock Trading per-family dominant failure modes.

A.6 Compute Resources

Web environment host. All DynamicWebArena websites and their backend services are containerized and run on a single host with an Intel Xeon Platinum 8253 CPU (2.20 GHz), 1.5 TB of RAM, and 2 TB of local storage. No GPU is required to host the environments.

Agent inference. All five evaluated baselines (DeepSeek-V4-Pro, Gemini-2.5-Pro, Qwen3-VL-235B-A22B, Grok-4.3, GPT-5.4) are accessed via their respective hosted inference APIs. Aggregate API token consumption across all baseline rollouts reported in this paper, including the capped/uncapped and time-information ablations, is approximately 2×10^9 tokens.

A.7 Licenses and Terms of Use

table 15 lists each external asset, its source, and its license used in DynamicWebArena. We use each asset under terms compatible with non-commercial research and the released benchmark distribution.

Asset	Source	License / Terms
NYC TLC trip records	NYC Taxi & Limousine Commission	NYC Open Data (public, "as is")
NYC taxi zone shapefiles	NYC TLC	NYC Open Data
OpenStreetMap (tiles, geocoding, routing)	openstreetmap.org	ODbL 1.0
Mapillary panoramas (Graph API)	mapillary.com	CC-BY-SA 4.0 + Mapillary API ToS
WebArena product listings	Zhou et al. [44]	Apache-2.0
Historical stock prices and fundamentals	Open data	NA
Arenagram images (1,000 images, 7 categories)	Open data	NA
Public YouTube videos (HLS segments via `ffmpeg`)	youtube.com	NA

Table 15: Licenses and terms of use for external datasets used in DynamicWebArena.

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 state our contributions: a self-hosted, deterministic, time-aware browser-agent benchmark and a measured agent-vs-human gap, which match the implementations and results in section 3 and appendix A.

   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: We discussed the limitations and future works in appendix A.5.

   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 contains no formal theorems or proofs; the POMDP formulation is descriptive.

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: crefsec:solution formalizes the environment, observation/action spaces, and evaluation rubrics. appendix A documents each website's stack, data sources, simulator dynamics, deterministic reset, and task families.

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 release code, Docker images, and task suites at `https://anonymous.4open.science/r/fullstackarena_submission-126B/`.

   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: appendix A (Baselines) lists the five evaluated LLMs (DeepSeek-V4-Pro, Gemini-2.5-Pro, Qwen3-VL-235B-A22B, Grok-4.3, GPT-5.4), interaction budgets, and the temporal-information ablation. Per-website task families and task counts are documented in appendix A.

   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: [No]

   Justification: Success rates are reported as point estimates. Multi-seed runs were not feasible given LLM inference API cost and rate-limit constraints.

   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: Host machine spec total agent token usage are reported in appendix A.6

   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]

   Justification: This research conforms with the NeurIPS Code of Ethics in every respect.

   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: The benchmark focuses on evaluating agent capabilities in controlled, self-hosted environments. While the tasks involve domains like bidding and trading, the environments are sandboxed simulations that do not interact with real systems. The primary societal benefit is enabling safer development of web agents by exposing temporal reasoning failures before deployment.

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: We release self-hosted environments and tasks only—no pretrained models or scraped personal data with notable misuse risk.

    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: Data sources are credited in appendix A (Website implementations), and appendix A.7 (table 15) lists licenses and terms of use.

    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 appendix documents each website, the deterministic reset procedure, task families, and evaluation schema for the released benchmark.

    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: This research does not involve crowdsourcing nor research with human subjects; this paper does not include a human baseline.

    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: This research does not involve crowdsourcing nor research with human subjects.

    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: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    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.