FULLSTACKARENA: EVALUATING BROWSER AGENTS IN DYNAMIC AND TEMPORALLY EVOLVING FULL-STACK ENVIRONMENTS

Anonymous authors
Paper under double-blind review

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 FULLSTACKARENA[1], a novel browser-use benchmark that challenges web agents with complex, full-stack environments. FULLSTACKARENA encompasses six diverse, fully operational websites spanning online bidding, stock trading, ride-hailing, maps, social media, and video. While emulating the dynamic impact of time-sensitive events, the benchmark remains entirely self-hosted, deterministic, and reproducible. Across all 781 tasks, the four baseline models pass only 16–18% of attempts on average across our evaluation settings.

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 (OpenAI, 2025), Perplexity's Comet (Perplexity AI, 2025), and Google's Gemini-integrated Chrome (Google, 2026).

Alongside those autonomous agents, researchers have developed a slew of benchmarks to understand and evaluate their capabilities (Deng et al., 2023; Shi et al., 2017; Zhou et al., 2023; Koh et al., 2024; Jang et al., 2024; Yao et al., 2022a). Today's benchmarks capture a wide range of website formats and tasks, from testing common website user interface elements (Shi et al., 2017) to tasks that require visual information and multimodality (Koh et al., 2024; Jang et al., 2024). While some of these benchmarks involve releasing agents onto the real Internet (Deng et al., 2023), 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, request-response model of the web. The client browser sends HTTP requests to a server to fetch static HTML, CSS, and JavaScript content (often in a RESTful style), they get a response back, the browser parses/renders it locally, and the agent executes 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`

[1]Code, Docker images, and task suites: `https://anonymous.4open.science/status/fullstackarena-iclr-B1CF`.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

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 (Liu et al., 2022). 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 FULLSTACKARENA, a novel benchmark suite for browser agents. FULLSTACKARENA provides evaluations over complex web services with asynchronous state changes and temporal dependencies, in addition to the traditional request-response workloads assumed by today's agents. While emulating these complexities, it simultaneously guarantees reproducibility and determinism through a combination of pseudo-randomness and procedural generation. FULLSTACKARENA'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 10.5-24.1%. 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 FULLSTACKARENA.

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: THE DOM AND JAVASCRIPT

A webpage begins as an HTML document: a tree of nested tags that references further resources (CSS, images, JavaScript libraries) which the browser fetches while parsing. In the Web 1.0 model, that document *was* the page; it was rendered once and did not change until the user navigated away.

Modern pages are instead running programs. The browser parses the HTML into the Document Object Model (DOM), a live, in-memory tree of element, attribute, and text nodes that represents what the user can currently see. Embedded JavaScript, triggered by user input (e.g., `onclick`), browser events (e.g., page load), timers, or arriving network data, can add, remove, or modify DOM nodes at any point after the initial load. The HTML a server returns is therefore only the page's *starting* state; what an agent observes depends on which scripts have run, and when.

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

2.2 SERVERS AND CLIENT-SERVER INTERACTION

Servers have evolved in parallel. Although HTTP itself is stateless, modern backends (e.g., `node.js`) keep per-session state in cookies, caches, and databases, and their logic is increasingly time-aware:

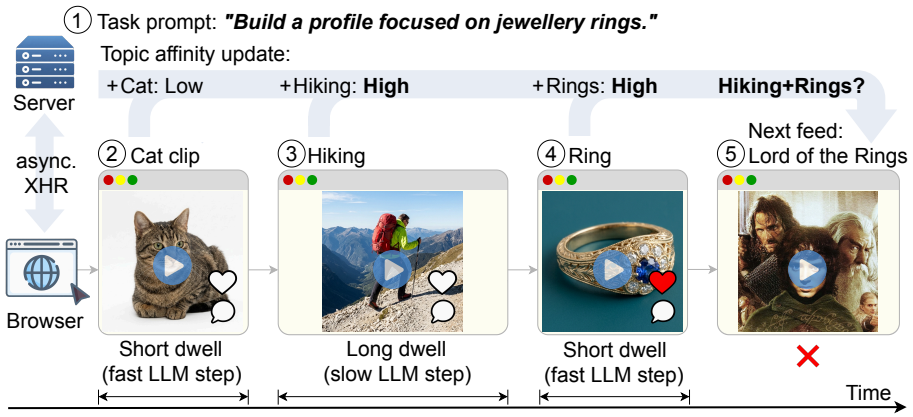


Figure 1: A browser agent interacting with a dynamic short-form video platform. Step 1 gives the task; Steps 2–4 show the server asynchronously updating topic affinity: a long hiking dwell (e.g., from a slow LLM step) raises *Hiking*, while liking the ring raises *Rings*. In Step 5, these combined signals lead to *The Lord of the Rings* rather than ring-focused content, so the task fails.

rate limits, session timeouts, expiring tokens, and time-varying prices make a response depend on *when* a request arrives, not just on what it contains.

The browser, in turn, no longer has to reload the page to communicate with the server. With `XMLHttpRequest` (XHR) and its Promise-based successor, the Fetch API, JavaScript issues asynchronous requests in the background and patches the DOM when responses arrive. Infinite scroll is the canonical example (Figure 1): while the user watches, the client prefetches the next several videos (5+ on TikTok) and the server's recommender decides what to serve, so the page never reloads but its content keeps changing. WebSockets extend XHR and Fetch with a persistent, full-duplex channel over which the server can *push* updates at any moment, and Server-Sent Events (SSE) provide a lighter one-way stream for notifications and live feeds. Newer protocols extend this even further, e.g., WebTransport (Consortium., 2023) over QUIC and peer-to-peer WebRTC (Uberti & Thatcher, 2011). The browser is, thus, only the surface of a deep, stateful, time-dependent stack: page content changes in response to the server, other users, and the clock, not only to the user's own actions.

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

The current paradigm of browser automation largely assumes the more traditional, request-response model of the web. While existing benchmarks have enabled tremendous strides in the capability of browser automation, particularly in terms of visual reasoning (Koh et al., 2024; Jang et al., 2024; Shi et al., 2017) and task-specific accuracy (Drouin et al., 2024; Yao et al., 2022a; Lù et al., 2024; Yoran et al., 2024; Wei et al., 2025), 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 dynamic system where the outputs are often a function of time as much as a function of request content.

State-of-the-art agents, such as WebVoyager (He et al., 2024), the `browser-use` library (Browser Use, 2026), and commercial solutions (e.g., Perplexity's Comet, OpenAI Atlas, and Google Chrome Auto Browser), largely inherit this assumption: each step first observes the DOM or a visual screenshot, predicts an action (e.g., a click or keystroke), and captures 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 to overestimate the user's interest in other topics (e.g., hiking and, by extension, Lord of the Rings),

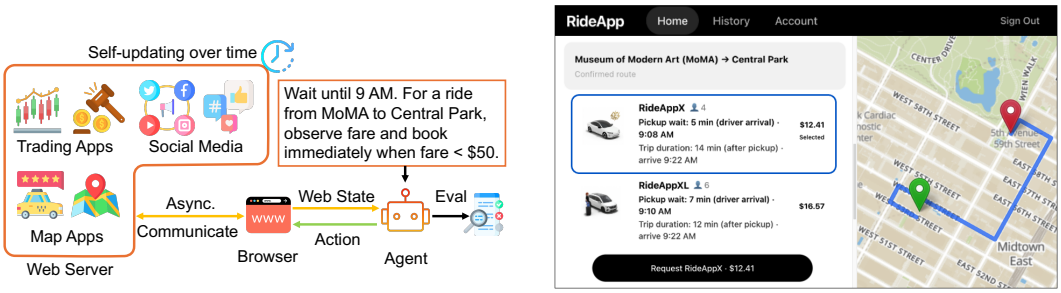


Figure 2: Left: Overview of FULLSTACKARENA, where browser agents interact with self-updating web servers through asynchronous communication. Right: Screenshot of RideApp, one of the six websites. Its backend asynchronously updates time-varying information such as ETA and fare.

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 FULLSTACKARENA: TOWARD A RICHER BROWSER AGENT ENVIRONMENT

This paper presents FULLSTACKARENA, a benchmark suite for evaluating browser agents on dynamic, asynchronous, and time-dependent web environments. Like popular existing benchmarks (Zhou et al., 2023; Koh et al., 2024; Jang et al., 2024; Drouin et al., 2024; Shi et al., 2017), FULLSTACKARENA 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 FULLSTACKARENA (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 FULLSTACKARENA'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, FULLSTACKARENA 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 (Zhou et al., 2023; Koh et al., 2024) model agent interaction as a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998). In this observe–act formulation, every transition is indexed by an agent action: the world is implicitly frozen between steps, and the time an agent spends choosing its next action is neither represented nor consequential. This abstraction breaks down on the modern web. Real web servers evolve on their own, driven by internal timers, scheduled jobs, and asynchronous data streams, and they do not pause while an LLM agent spends seconds to minutes generating its next action. Like in Figure 1, the agent's own deliberation time becomes part of the environment dynamics.

To capture this, we adopt a continuous-time, partially observable semi-Markov formulation (Sutton et al., 1999; Yu, 2006; Alt et al., 2020) that makes the delay between observing the environment and acting on it explicit. We describe the shared environment dynamics and observations by

$$\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, Z, \phi, g).$$

$\mathcal{S}$ contains the hidden browser and server state, including database contents, active sessions, pending requests, scheduled events, and the simulated environment's clock, whose reading we denote by $t \geq 0$; all times below refer to this clock, and $s(t)$ denotes the state at time t. $\mathcal{A}$ contains browser actions together with an explicit `wait(`Δt`)` action. $\mathcal{O}$ contains the observations available to the agent, such as the DOM or accessibility tree, screenshots, and current and elapsed time; $Z(o \mid s)$ specifies the observation distribution given state s.

We separate changes caused by the passage of time from changes caused by agent actions. $\phi(s, \delta)$ gives the state reached after $\delta \geq 0$ units of time with no new action, capturing background activity such as price updates, auction countdowns, and completing requests. $g(s, a)$ gives the state immediately after action a is applied; any asynchronous work it triggers then proceeds through ϕ. Both functions are deterministic given the full state (Section 3.1).

Interactions proceed as follows. At time t_k, the agent receives $o_k \sim Z(\cdot \mid s(t_k))$ and takes a *decision delay* $d_k \geq 0$, covering LLM inference and other overhead, to issue action a_k. Because the environment keeps evolving, a_k is applied to $\phi(s(t_k), d_k)$ rather than to the state the agent observed. A further *execution delay* $e_k \geq 0$, e.g., for a page load, then elapses before the next observation:

$$t_{k+1} = t_k + d_k + e_k, \qquad s(t_{k+1}) = \phi\big(g(\phi(s(t_k), d_k),\, a_k),\, e_k\big).$$

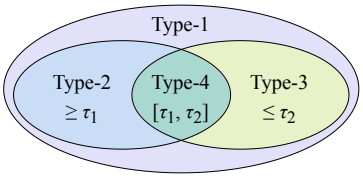
For `wait(`Δt`)`, $g(s, a) = s$ and $e_k = \Delta t$. As a result, identical action sequences can yield different outcomes: in Figure 1, LLM deliberation counts as watch time and changes the recommendations.

Given a task described as a natural-language intent i, the reward $r(\mathbf{a}_0^K, \mathbf{s}_0^K)$ scores the trajectory up to the final epoch K, where each action is paired with its application time $t_k + d_k$ and $\mathbf{s}_0^K$ are the corresponding states. It checks whether the state transitions satisfy the intent, including its timing constraints (Section 3.3); e.g., for an intent to place an order before a given time, the order must both exist and have been placed in time. It also evaluates the agent's output, such as the correctness of a predicted answer.

3.3 Task Taxonomy and Construction

We organize tasks according to the temporal constraints required for successful completion, as summarized in Figure 3. Tasks are instantiated over the same stateful web environments, but differ in whether correctness depends only on *what* the agent does or additionally on *when* it acts. We consider two elementary and independent temporal constraints: a task may require the agent not to start until a release time τ_1, or to finish by a deadline τ_2. Their combinations naturally yield four mutually exclusive task types.

Type-1 tasks impose no explicit temporal constraint and may be completed at any time, providing a control for the underlying web-interaction difficulty. Type-2 tasks impose a start-time constraint, requiring the agent to begin no earlier than the release time τ_1. Type-3 tasks impose a deadline, requiring the task to be completed no later than τ_2. Type-4 tasks combine both constraints, requiring execution within the bounded time window $[\tau_1, \tau_2]$. This taxonomy separates temporal requirements from website-specific functionality, allowing the same temporal structure to be studied across heterogeneous web applications.



Type	Time	Example
Type-1	None	Place a second order if the first fails.
Type-2	$\geq \tau_1$	Place an order after 9:00 AM.
Type-3	$\leq \tau_2$	Buy 10 shares before 9:05 AM.
Type-4	$[\tau_1, \tau_2]$	Post between 9:00–9:05 AM.

Figure 3: Temporal task taxonomy in FULLSTACKARENA. The two circles represent independent lower- and upper-time constraints. Type-1 has no time constraint, Type-2 imposes a lower bound τ_1, Type-3 imposes an upper bound τ_2, and Type-4 lies in their intersection and satisfies both.

Tasks are constructed from the functional workflows and dynamic mechanisms supported by each website. Rather than treating temporal constraints as isolated timing instructions, we embed them into multi-step browser interactions involving operations such as navigation, search, comparison, state inspection, waiting, and state-changing actions. A task may therefore require an agent to identify a target, observe or wait for the environment to evolve, re-evaluate the resulting state, and then act under the corresponding temporal constraint. Parameterized task templates are instantiated with different entities, values, and temporal conditions to produce concrete task instances.

In total, FULLSTACKARENA contains 781 tasks, instantiated from 192 templates, spanning all four types across six websites and a cross-website set (Section A.2). Representative task instances from each website are shown together with their evaluation annotations in Table 1. The website-specific mechanisms from which these tasks are constructed are described in Section A.1, and the complete per-website task distribution is reported in Section A.2.

3.4 EVALUATION ANNOTATION

Following (Zhou et al., 2023), we evaluate functional correctness rather than requiring an agent to reproduce a reference action trajectory. Each task contains an `eval` block specifying one or more conditions for successful completion; when multiple conditions are specified, all of them must pass.

We implement four primary programmatic evaluation types. `string_match` checks textual responses through exact or substring matching, exclusion rules, and numeric constraints. `url_match` verifies navigation outcomes from the resulting browser URL. `program_html` checks rendered webpage content and state through programmatic queries. `api_match` directly queries the application backend to verify persistent outcomes such as bookings, transactions, bids, and social-media interactions. For free-form answers that cannot be reliably judged by lexical matching, we additionally support an LLM-based fuzzy matcher for semantic equivalence, following the evaluation approach used in WebArena (Zhou et al., 2023). For temporally constrained tasks, evaluators also validate the relevant timing condition, so a correct action performed outside the required interval is counted as a failure.

Table 1 illustrates representative task intents and their evaluation implementations across all six websites. Additional details on the controlled clock, environment reset, and temporal interface used during evaluation are provided in Section B.

4 EXPERIMENTS

We evaluate four baseline models, GPT-5.6-Sol, Gemini-3.8-Flash, DeepSeek-V4-Pro, and Qwen3.5-122B-A10B, using a common `browser-use` v0.12.1 agent scaffold with its default reasoning-enabled system prompt and structured browser-action interface. Each task has a maximum of 30 agent steps. We report three configurations: capped waiting without automatic time information, uncapped waiting without automatic time information, and uncapped waiting with automatic time information. In the capped configuration, each `wait(seconds)` action is limited to 30 seconds; in the uncapped configurations, it has no upper bound. Automatic time information supplies the elapsed time since task start, the duration of the previous step, and the current `Website time` in each observation. The agent can call the website-time tool in every configuration. Details of the

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: Representative task intents and their evaluation implementations across the six websites. Blue indicates expected values; red indicates evaluation functions.

agent interface, temporal observations, clock configuration, and environment reset are provided in Section B.

4.1 Main Results

Table 2 shows current browser agents achieve low success rates on FullStackArena. Pooled over all 781 tasks, the strongest model, GPT-5.6-Sol, succeeds on only 24.1% of tasks with capped waiting and 26.2% with uncapped waiting, while the other three models remain below 18%. Although GPT-5.6-Sol leads on most websites, under capped waiting, Gemini-3.8-Flash and DeepSeek-V4-Pro outperform it on Stock Trading and Ride-hailing, respectively. Humans complete 83.7% of the tasks (Section 4.2), far above the best agent.

Does the model have a sense of time? Trajectory inspection shows that agents often choose the right action but fail to execute it at the right moment; timing-related violations account for 52–86% of the failures (Section C.2). We observe three recurring patterns: (1) Slow execution. A single observe–reason–act step often takes more than 10 seconds, so when a task requires acting within a short window, an otherwise correct action frequently lands too late. (2) Poor control of dwell time. On Social Media, the recommender infers interest from how long a post stays on screen, but agents move through the feed one step at a time and often exceed the permitted dwell time (e.g., 19–29 seconds when at most 18 are allowed). (3) Acting on stale information. Agents often act on page state read well before execution without re-observing it; on Auction, an agent may wait for the bidding window to open and then bid on a listing whose ranking has changed in the meantime.

Does automatic time information help? Figure 4 compares uncapped waiting with and without automatic time information. Time information helps overall, raising the pooled success rate from 16.0% to 17.7%, and most for GPT-5.6-Sol (from 26.2% to 30.7%; e.g., from 24.0% to 40.0% on Auction). However, on Map, GPT-5.6-Sol drops from 20.6% to 14.2%, and on Ride-hailing, Qwen3.5-122B-A10B drops from 19.2% to 10.8%. Trajectories suggest why: without time information, agents tend to poll the page frequently, whereas with it, some agents instead infer a single long wait, which is often miscalculated and performs worse than frequent polling. Time information is thus useful, but current agents cannot yet reliably reason over it.

Does uncapped waiting help? We compare capped and uncapped waiting without automatic time information (Table 2). We observed, for long waits, the cap forces the agent to split a single wait into many 30-second waits, each followed by a full agent step, which adds delay and requires the agent to track how long it has already waited; removing the cap avoids this. For short timing windows,

Model	Stock Trading		Auction		Ride-hailing		Map	
	C	U	C	U	C	U	C	U
GPT-5.6-Sol	14.0%	**19.9%**	**22.0%**	**24.0%**	20.8%	**32.5%**	**21.9%**	**20.6%**
Gemini-3.8-Flash	**16.9%**	14.0%	12.0%	13.0%	21.7%	24.2%	18.1%	20.0%
DeepSeek-V4-Pro	14.0%	16.2%	2.0%	2.0%	**22.5%**	21.7%	7.7%	5.2%
Qwen3.5-122B-A10B	14.0%	6.6%	0.0%	3.0%	15.8%	19.2%	12.9%	9.7%

Model	Social Media		Video		Cross-website		Total	
	C	U	C	U	C	U	C	U
GPT-5.6-Sol	**34.8%**	**37.9%**	**27.8%**	**22.2%**	**35.4%**	**27.1%**	**24.1%**	**26.2%**
Gemini-3.8-Flash	21.2%	21.2%	21.1%	18.9%	4.2%	2.1%	17.7%	17.7%
DeepSeek-V4-Pro	6.1%	5.3%	20.0%	**22.2%**	2.1%	0.0%	11.1%	10.9%
Qwen3.5-122B-A10B	6.1%	6.1%	17.8%	14.4%	0.0%	0.0%	10.5%	9.1%

Table 2: Success rate (SR) of baseline models without automatic time information. C: capped (each `wait` limited to 30 s); U: uncapped. Best result per column in **bold**. Each result cell contains 136 Stock Trading, 100 Auction, 120 Ride-hailing, 155 Map, 132 Social Media, 90 Video, or 48 Cross-website tasks; Total pools all 781 tasks.

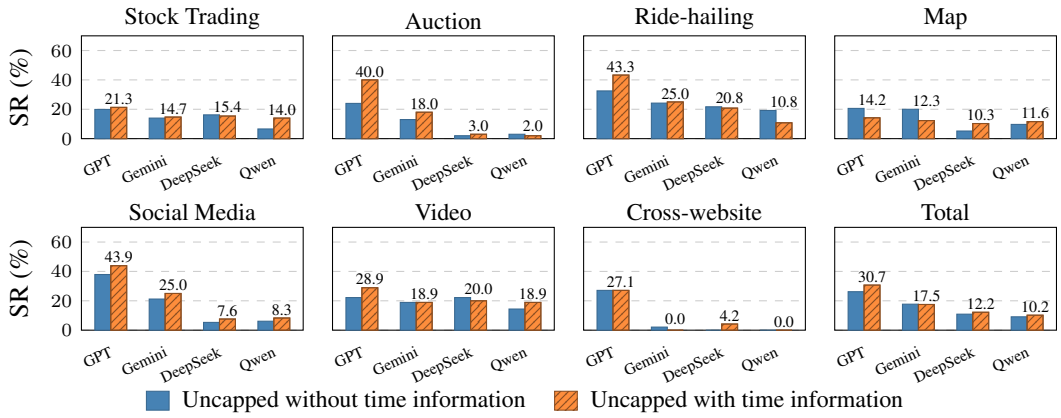


Figure 4: Success rate (SR) under uncapped waiting with and without automatic time information. The number above each pair of bars is the SR with automatic time information; the SR without it corresponds to the U columns of Table 2.

however, an uncapped agent may issue a single long wait and skip past the window. These opposing effects largely cancel out, the success rate barely changes for this (15.8% to 16.0%).

4.2 Human Baseline

To verify the low agent success rates reflect agent limitations rather than unsolvable tasks, we collected a human reference in which six annotators attempted all tasks. The success rate is 83.7% (Table 3). Inspecting the recorded trajectories, we find most human failures stem from minor slips rather than genuine difficulty: impatience (e.g., submitting an action before the required time), missed timing windows (e.g., switching to another activity while waiting and forgetting to return), incomplete answers (e.g., describing only three Shorts when the task asks for five), and misreading the instruction (e.g., operating on regular videos when the task targets Shorts).

5 Related Work

Web Agent Benchmarks. Self-hosted benchmarks such as WebArena (Zhou et al., 2023), WebShop (Yao et al., 2022a), WorkArena (Drouin et al., 2024), REAL (Garg et al., 2026), and WoB (Shi

	Stock Trading	Auction	Ride-hailing	Map	Social Media	Video	Cross-website	Total
Human	80.1%	77.0%	86.7%	88.4%	81.8%	90.0%	79.2%	83.7%
Best agent	21.3%	40.0%	43.3%	21.9%	43.9%	28.9%	35.4%	30.7%

Table 3: Success rate (SR) of humans versus the best-performing agent on each website. For each website, best agent is the highest success rate among all models and settings.

et al., 2017) provide controlled, reproducible environments; VisualWebArena and VideoWebArena extend WebArena multimodally (Koh et al., 2024; Jang et al., 2024), BrowserGym unifies many benchmarks behind a common interface (Chezelles et al., 2024), and OSWorld (Xie et al., 2024) and TheAgentCompany (Xu et al., 2026) broaden the setting to operating systems and workplaces. Other benchmarks use recorded snapshots (Deng et al., 2023; Lù et al., 2024), which freeze external dynamics, or the live web (He et al., 2024; Yoran et al., 2024; Pan et al., 2024; Xue et al., 2025; Wei et al., 2025; Gou et al., 2025), which is hard to reproduce. Recent work models time more directly through time-varying answers (Gou et al., 2025), live markets (Qian et al., 2026), client-side real-time challenges (Thomas et al., 2025), or simulated app environments that evolve independently of the agent (Froger et al., 2026). FULLSTACKARENA evaluates browser agents on reproducible full-stack web applications whose server-side state evolves continuously over time.

Language Model Based Agents. Language-model agents have progressed from browser-based question answering and reason–act loops (Nakano et al., 2021; Yao et al., 2022b) to agents operating real websites, rendered pages, or human-like browsing interfaces (Gur et al., 2024; Zheng et al., 2024; He et al., 2024; Hong et al., 2023; Yu et al., 2025). Later work improves planning and error recovery (Yao et al., 2023; Gu et al., 2025; Shinn et al., 2023), separates high-level planning from execution (Agashe et al., 2025a;b), and learns from interaction, reusable workflows, or trajectories synthesized from web tutorials (Putta et al., 2024; Wang et al., 2025; Xu et al., 2025). Most still assume a synchronous observe–think–act loop, with only limited work on time-aware agents (Ginart et al., 2024; Wen et al., 2026). FULLSTACKARENA instead stresses agents with continuously evolving, partially observable web state.

Modeling Temporal Dynamics in Interactive Environments. Interactive environments are commonly modeled as MDPs (Puterman, 2014; Sutton et al., 1998), with extensions to alternative discounting (Schultheis et al., 2022) and dynamics learned from observational data (Gasse et al., 2021; 2023), or as POMDPs (Kaelbling et al., 1998), including continuous-time variants (Alt et al., 2020). Existing web formulations remain discrete: WebArena models action-triggered deterministic transitions (Zhou et al., 2023), VisualWebArena uses a POMDP (Koh et al., 2024), and earlier work defines an MDP over DOM states (Liu et al., 2018). Semi-MDPs model variable action durations (Sutton et al., 1999; Yu, 2006), while real-time reinforcement learning lets the environment evolve during computation (Ramstedt & Pal, 2019; Xiao et al., 2020; Bouteiller et al., 2021), including recent work on LLM agents in games (Wen et al., 2026). Building on these formulations, FULLSTACKARENA models web interaction as a continuous-time POSMDP (§3.2), where hidden state includes absolute time and can evolve while the agent is idle or generating its next action.

6 CONCLUSION

In this work, we introduce FULLSTACKARENA, a benchmark for evaluating browser agents in dynamic, temporally evolving web environments. Comprising six self-hosted websites and 781 tasks, FULLSTACKARENA 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 FULLSTACKARENA will drive the development of more temporally-aware agent architectures.

ETHICS STATEMENT

FULLSTACKARENA consists of self-hosted, sandboxed web applications; no task interacts with a live third-party service, and agent actions have no effect outside the local deployment. Website content is either synthesized or derived from open-source datasets with authorization for non-commercial usage (Section A.1). The human baseline was reviewed and approved by our institution's Institutional Review Board; six participants took part voluntarily after being informed of the study's purpose; we recorded only task outcomes and interaction trajectories, collected no personal information, and report results only in aggregate (Sections B.4 and 4.2). The benchmark is intended for diagnostic research, and its sandboxed environments pose little risk of misuse.

REPRODUCIBILITY STATEMENT

Code, Docker images, and the complete task suite are available at `https://anonymous.4open.science/status/fullstackarena-iclr-B1CF`. Section 3 describes how a simulated clock and seeded procedural generation make every environment deterministic, and how each task is scored by programmatic evaluators. Sections A.1 and A.2 describe how each website is built, the data it uses, and the task suite, so that the benchmark can be rebuilt. Sections B.1 and B.2 specify how each environment and its clock are restored to a fixed initial state before every task, so that all agents start from identical conditions. Section 4 and Section B.3 specify the models, agent scaffold, step budget, and evaluation configurations needed to reproduce our experiments.

AI USE STATEMENT

We used LLM-based assistants to help with writing and with implementing the benchmark environments. They were not used to generate research ideas or to design the benchmark or experiments. All AI-assisted content was reviewed and verified by the authors.

REFERENCES

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*, 2025a. URL `https://openreview.net/forum?id=43XMKuTTK0`.

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*, 2025b. URL `https://openreview.net/forum?id=zg5is4GJ3R`.

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

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher J Pal, and Jonathan Binas. Reinforcement learning with random delays. In *ICLR*, 2021.

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

De Chezelles, Thibault Le Sellier, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F Xu, Siva Reddy, Quentin Cappart, et al. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024.

World Wide Web Consortium. Webtransport, 2023. URL `https://www.w3.org/TR/webtransport/`.

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.

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.

Romain Froger, Pierre Andrews, Matteo Bettini, et al. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, 2026.

Divyansh Garg, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, Christian Schroeder de Witt, and Sumeet Ramesh Motwani. REAL: Benchmarking autonomous agents on deterministic simulations of real websites. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=Un1sWxmZuI.

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

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.

Antonio A Ginart, Naveen Kodali, Jason Lee, Caiming Xiong, Silvio Savarese, and John Emmons. Asynchronous tool usage for real-time agents. *arXiv preprint arXiv:2410.21620*, 2024.

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.

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.

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.

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.

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 (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 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/.

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)*, pp. 14281–14290, 2023. URL https://api.semanticscholar.org/CorpusID:266210390.

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.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Seungbae Kim, Jyun-Yu Jiang, Masaki Nakada, Jinyoung Han, and Wei Wang. Multimodal post attentive profiling for influencer marketing. In *Proceedings of The Web Conference 2020*, pp. 2878–2884, 2020.

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)*, pp. 881–905, 2024.

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.

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.

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

Dennis Luxen and Christian Vetter. Real-time routing with OpenStreetMap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 513–516, 2011. doi: 10.1145/2093973.2094062.

Mapillary. Mapillary. https://www.mapillary.com.

Metropolitan Transportation Authority. MTA subway GTFS static feed. https://rrgtfsfeeds.s3.amazonaws.com/gtfs_subway.zip.

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.

NYC Taxi and Limousine Commission. TLC trip record data. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed: 2026-09-26.

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

OpenStreetMap contributors. OpenStreetMap. https://www.openstreetmap.org. Accessed: 2026-09-26.

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.

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

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

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.

Lingfei Qian, Xueqing Peng, Hanley Smith, Yi Han, Yueru He, Haohang Li, Yupeng Cao, Yangyang Yu, Guojun Xiong, Peng Lu, Yan Wang, Vincent Jim Zhang, Huan He, Alejandro Lopez-Lira, Jimin Huang, Jian-Yun Nie, and Sophia Ananiadou. When agents trade: Live multi-market trading arena for llm agents. In *Proceedings of the ACM Web Conference 2026*, WWW '26, pp. 7833–7844, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723070. doi: 10.1145/3774904.3792821. URL https://doi.org/10.1145/3774904.3792821.

Simon Ramstedt and Chris Pal. Real-time reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

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

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*, pp. 3135–3144. PMLR, 2017.

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.

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

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.

George Thomas, Filippos Christianos, Alex James Chan, Rohit Midha, Jikun Kang, Wenqi Wu, Fraser David Greenlee, Andrew Toulis, and Marvin Purtorab. Webgames: Challenging general-purpose web-browsing AI agents. In *ICML 2025 Workshop on Computer Use Agents*, 2025. URL https://openreview.net/forum?id=lEIV7ZWu3t.

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

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.

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.

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=n1AvXiU2lu.

Ted Xiao, Eric Jang, Dmitry Kalashnikov, Sergey Levine, Julian Ibarz, Karol Hausman, and Alexander Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. *arXiv preprint arXiv:2004.06089*, 2020.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Zhiruo Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Melroy Maben, Raj Mehta, Wayne Chi, Lawrence Keunho Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking LLM agents on consequential real world tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=LZnKNApvhG.

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.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=6jZi4HSs6o.

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, 2022a.

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*, 2022b.

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.

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*, pp. 8938–8968, 2024.

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

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.

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.

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 BENCHMARK DETAILS

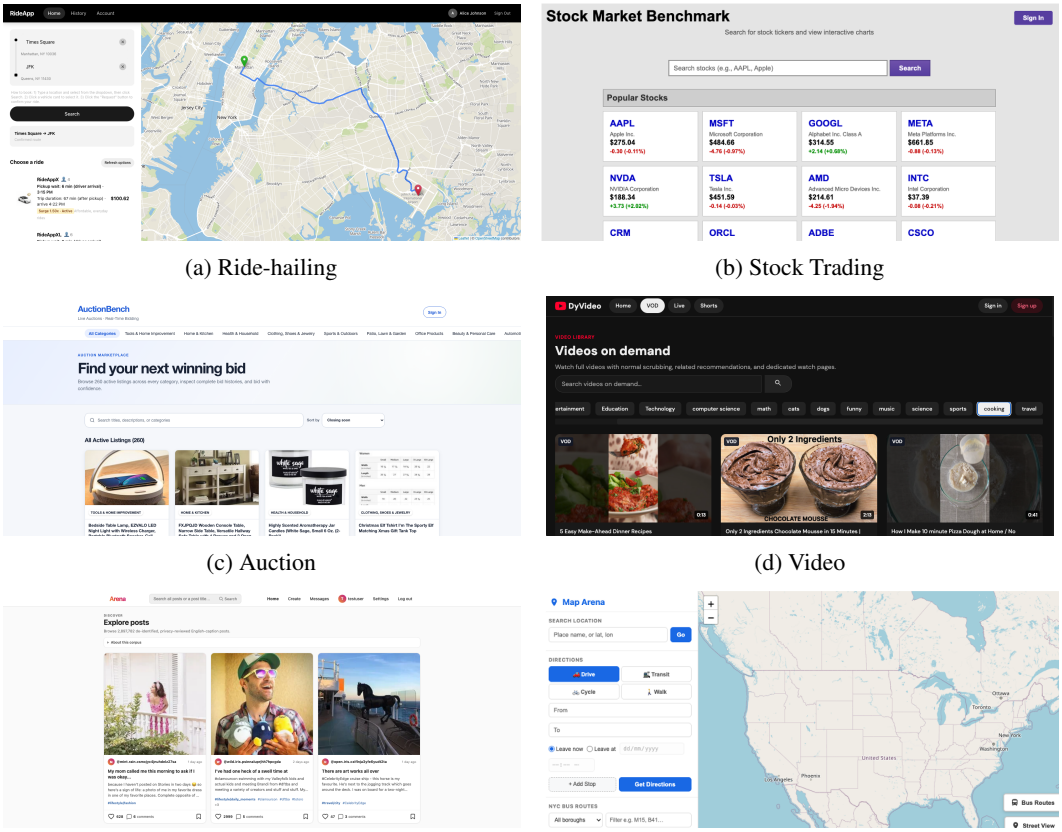
This section describes the additional details about how the six websites are constructed (Section A.1), and how tasks are distributed across websites (Section A.2).

A.1 WEBSITE CONSTRUCTION AND DATA SOURCES

FULLSTACKARENA comprises six self-hosted web applications (Figure 5), each deployed in containers with its own backend server and database. Each application reproduces the core workflows of its real-world counterpart while maintaining backend state that evolves with a simulated clock. Table 4 summarizes the functionality and scale of each website.

Website content is either derived from publicly available sources or synthesized. Ride-hailing uses travel-time, distance, and demand statistics aggregated from NYC TLC trip records (NYC Taxi and Limousine Commission). Ride-hailing and Map compute routes with OSRM (Luxen & Vetter, 2011) over OpenStreetMap data (OpenStreetMap contributors), and Map additionally uses the MTA's published subway timetable (Metropolitan Transportation Authority) and street-level imagery from Mapillary (Mapillary). Auction listings are derived from the product catalog released with WebShop (Yao et al., 2022a) and used in WebArena (Zhou et al., 2023). Arenagram posts are drawn from a research dataset of public Instagram posts (Kim et al., 2020), used with the distributor's authorization for research purposes. Stock prices combine historical base prices with procedurally generated variation, and the remaining quantities, including bid histories, engagement counts, and traffic conditions, are synthesized. All external data are used solely for non-commercial research.

Each website's dynamic behavior is driven by a simulated clock that is initialized to a task-specific time at the start of every task. All time-varying quantities are deterministic functions of the simulated



(a) Ride-hailing (b) Stock Trading

(c) Auction (d) Video

(e) Arenagram (Social Media) (f) Map

Figure 5: Screenshots of the six self-hosted websites in FULLSTACKARENA.

Website	Representative functionality	Scale
Stock Trading	Portfolio trading, watchlists, stock screening, and bid–ask quotes, with prices updated every second	30 tickers across multiple sectors
Auction	Listing search, bidding, bid histories, and staggered auction closings	260 listings in 15 categories; 1,560 seeded bids
Ride-hailing	Fare and ETA quotes for three vehicle classes, surge pricing, and booking	263 NYC taxi zones; 178,303 zone-to-zone travel-time statistics; 8,290 surge factors
Map	Driving, cycling, walking, and transit routing, departure-time ETAs, location search, and street-level imagery	OpenStreetMap road network for the northeastern US with car, bike, and foot routing profiles; NYC subway timetable; street-level panoramas
Arenagram (Social Media)	Personalized feeds, likes, saves, comments, follows, and search	Posts drawn from a research corpus of public Instagram posts
Video	On-demand, live, and short-form videos, skippable ads, and recommendations	Locally hosted video library streamed via HLS

Table 4: Overview of the six websites in FULLSTACKARENA, listing representative functionality and the scale of the underlying data.

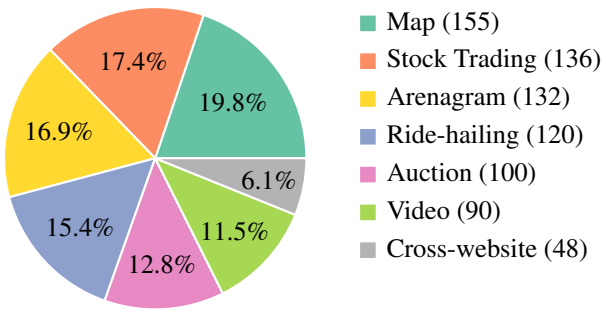


Figure 6: Distribution of 781 tasks (192 templates) in FULLSTACKARENA: six websites plus 48 cross-website tasks.

time and fixed seeds, rather than draws from a runtime random number generator: stock prices are computed each second from a seeded multi-scale noise function, ride quotes are perturbed by hash-based noise that changes every 30 seconds, traffic multipliers combine hour, road-type, area, and weekday factors, and engagement counts are generated once with a fixed seed. As a result, an agent that observes a page at the same simulated time always sees the same values, and task answers remain stable across runs and across parallel deployments.

A.2 TASK DISTRIBUTION

FULLSTACKARENA contains 781 tasks instantiated from 192 parameterized templates, as shown in Figure 6: 733 tasks (180 templates) on the six individual websites and 48 cross-website tasks (12 templates) that require coordinating actions across more than one website. Map contains 155 tasks (19.8%), Stock Trading 136 (17.4%), Arenagram 132 (16.9%), Ride-hailing 120 (15.4%), Auction 100 (12.8%), Video 90 (11.5%), and the cross-website set 48 (6.1%). Each template is instantiated with different entities, values, and temporal conditions, yielding between three and five instances per template. Together, these tasks cover the temporal-constraint types defined in the main text while spanning a diverse set of dynamic web applications.

	Stock Trading				Auction			
Task type	C	U	U+T	All	C	U	U+T	All
Type-1	96.7%	85.0%	96.7%	92.8%	56.2%	59.4%	59.4%	58.3%
Type-2	0.5%	1.8%	1.4%	1.2%	13.4%	14.3%	19.6%	15.8%
Type-3	0.7%	1.5%	4.4%	2.2%	0.0%	0.8%	8.6%	3.1%
Type-4	15.6%	15.6%	17.2%	16.1%	2.3%	4.7%	8.6%	5.2%
All types	14.7%	14.2%	16.4%	15.1%	9.0%	10.5%	15.8%	11.8%
	Ride-hailing				Map			
Task type	C	U	U+T	All	C	U	U+T	All
Type-1	91.7%	89.6%	83.3%	88.2%	40.7%	37.0%	34.3%	37.3%
Type-2	50.0%	69.8%	63.5%	61.1%	10.9%	10.9%	7.6%	9.8%
Type-3	2.1%	2.1%	1.4%	1.9%	23.1%	19.2%	14.1%	18.8%
Type-4	1.0%	2.1%	8.9%	4.0%	1.5%	2.3%	3.4%	2.4%
All types	20.2%	24.4%	25.0%	23.2%	15.2%	13.9%	12.1%	13.7%
	Social Media				Video			
Task type	C	U	U+T	All	C	U	U+T	All
Type-1	50.0%	56.2%	50.0%	52.1%	73.3%	65.0%	75.0%	71.1%
Type-2	19.4%	18.8%	24.3%	20.8%	1.4%	1.4%	1.4%	1.4%
Type-3	13.9%	16.0%	19.4%	16.4%	29.2%	27.1%	26.0%	27.4%
Type-4	9.4%	8.3%	13.0%	10.2%	3.8%	3.0%	5.3%	4.0%
All types	17.0%	17.6%	21.2%	18.6%	21.7%	19.4%	21.7%	20.9%
	Cross-website				Total			
Task type	C	U	U+T	All	C	U	U+T	All
Type-1	15.6%	12.5%	12.5%	13.5%	61.1%	57.5%	58.5%	59.0%
Type-2	18.8%	15.6%	12.5%	15.6%	14.2%	16.9%	17.3%	16.1%
Type-3	6.2%	4.7%	6.2%	5.7%	10.6%	10.1%	11.3%	10.7%
Type-4	7.8%	3.1%	4.7%	5.2%	5.2%	5.3%	8.5%	6.3%
All types	10.4%	7.3%	7.8%	8.5%	15.8%	16.0%	17.7%	16.5%

Table 5: Overview of success rate by temporal-constraint type, aggregated over four models. C: capped; U: uncapped; U+T: uncapped with automatic time information; All: pooled over the three configurations shown.

B Evaluation Setup

This section describes the configuration shared by all evaluations: how each environment and its clock are initialized before a task (Sections B.1 and B.2), how agents observe and act (Section B.3), how the human baseline is collected (Section B.4), and the compute used (Section B.5).

B.1 Environment Reset

To ensure consistent initial conditions across evaluation runs, we reset each environment to a predefined state before every task. The reset removes mutable state produced by previous interactions and restores the seeded state required by the benchmark. Depending on the application, this includes ride requests and cached estimates, portfolio positions and transaction history, auction bids and listing state, social-media interactions and recommendation preferences, and video watch history and user preferences. Predominantly read-only resources, such as routing graphs, map tiles, and panorama collections, remain unchanged. This task-level reset prevents previous agent behavior or residual backend state from affecting subsequent evaluations.

B.2 Clock Configuration

For environments with time-evolving behavior, we use controllable simulated clocks to drive temporal backend updates. At the beginning of each task, the environment clock is initialized to a predefined time anchor and subsequently advances throughout agent interaction. Different applications use clock configurations appropriate to their domains, such as market hours for stock trading and local time for ride-hailing. Dynamic quantities, including ride estimates, stock prices, auction progress, recommendation recency, and traffic-adjusted ETAs, depend on the current environment time together with deterministic or seeded environment logic. This design allows website state to continue evolving independently of agent actions while keeping temporal evolution reproducible across runs.

B.3 Agent and Temporal Interface

We use `browser-use` v0.12.1 as the common browser-agent scaffold and retain its default reasoning-enabled system prompt, adding only a single instruction that points the agent to a clock tool: "Use `get_website_time()` when the task depends on current website time." Agents use the default multimodal interface, receiving the current browser state and screenshot at each step and interacting with the website through a structured tool interface. Up to five actions may be generated per step. The system prompt otherwise includes no specialized guidance for deadlines, temporal monitoring, or the semantics of the simulated clock.

In all configurations, `get_website_time()` is registered as a tool that returns the current website time, the elapsed time since the task began, and the duration of the previous agent step. With automatic time information, the same fields are additionally included in every step's `step_info`, so the agent receives them without calling the tool; without it, the agent obtains them only by calling the tool. Agents can also invoke an explicit `wait(seconds)` action to allow time to pass intentionally. During a wait, the website and its backend state continue to evolve, and the agent receives an updated browser observation at the next step.

B.4 Human Evaluation Protocol

To assess the solvability of FullStackArena tasks and provide a human reference for agent performance, we recruited 6 participants with backgrounds in computer science or related technical fields and prior experience with modern web applications. For applications involving more specialized interaction patterns, such as stock trading and map navigation, participants were familiarized with the basic functionality of the corresponding website before evaluation. This reduces failures caused primarily by unfamiliarity with the interface or application domain rather than by the task itself.

We evaluated humans on all 781 tasks, spanning all six websites and the cross-website set. Participants received the same natural-language task instructions as the browser agents and interacted with the same self-hosted environments through a standard browser interface. Human participants could observe the website clock presented in the interface and could wait naturally when a task required monitoring a changing state or acting within a specified time window. Human trajectories were evaluated using the same programmatic task evaluators used for agent evaluation, ensuring a consistent success criterion.

B.5 Compute Resources

All FullStackArena websites and their backend services are containerized and hosted on a single machine 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 benchmark environments. Agent inference is performed through hosted model APIs and therefore does not require local accelerator resources. Across all baseline rollouts reported in this paper, including the capped/uncapped wait and temporal-information experiments, the total API usage is approximately 2×10^9 tokens.

C Additional Results

This section reports results that complement the main text: agent success broken down by temporal-constraint type (Section C.1) and a trajectory-level error analysis (Section C.2).

C.1 Success Rate by Temporal-Constraint Type

We report the same four models under three configurations. C denotes capped waits without automatic time information, U denotes uncapped waits without automatic time information, and U+T denotes uncapped waits with automatic time information; the All column pools these three configurations. Table 5 summarizes success rates by temporal-constraint type for each website.

C.2 Error Analysis

To better understand where current browser agents fail in dynamic environments, we manually inspect failed trajectories and assign each failure a primary cause. When a trajectory violates multiple requirements, we attribute it to the earliest failure that is most causally decisive for the final outcome. Rather than constructing a fine-grained taxonomy for every task template, we focus on recurring failure patterns that appear across different dynamic applications.

Timing violations. The most prominent failure mode is the inability to reliably satisfy temporal constraints. Among the three environments covered by our detailed trajectory audit, timing-related violations account for 60.9% of Auction failures, 86.4% of Ride-hailing failures, and 52.2% of Arenagram failures. These failures include missing deadlines, acting before or after a required window, and violating timing constraints within multi-stage interaction sequences. In many such cases, the agent identifies the correct target and attempts the correct type of action, but executes it at the wrong time. This pattern is particularly common in tasks that require the agent to first observe a dynamic state, wait for the environment to evolve, and then react within a limited window.

Dynamic-state tracking and re-decision. A second recurring failure mode is insufficient adaptation to state changes that occur after an initial observation. For example, in Auction, 21.7% of failures arise because the agent continues using a stale shortlist or target after bid activity or rankings have changed. The agent may correctly identify a candidate early in the trajectory but fail to recompute the decision immediately before acting. A related pattern appears in Arenagram, where 26.1% of failures involve the recommendation or exploration loop: agents terminate before the recommendation state reaches the requested target, or fail to verify the updated feed after their interactions.

Interaction and completion errors. We also observe more conventional browser-agent failures, including incomplete filtering or comparison, missing required mutations, selecting the wrong target, and returning an answer that does not match the final website state. These errors remain important, but they are less dominant in the audited time-sensitive tasks. For example, most Ride-hailing failures arise from violations in the timed interaction chain rather than from basic route or vehicle recognition, while only a small fraction of Auction failures are caused by incomplete shortlists or missing observations.