TRACE: Trajectory Recording and Capture Environments

Abstract

Language-model agents for web and computer use are starting to automate realistic browser tasks, but the environments needed to train and evaluate them are still costly to build and fragile to maintain, often requiring teams to handcraft websites and workflows as in recent web-agent benchmarks. This paper introduces **TRACE (Trajectory Recording and Capture Environments)**, a pipeline for capturing, post-processing, and replaying browser environments and trajectories from expert demonstrations on real websites. TRACE uses an instrumented browser to record every interaction, page state, network request, and visual change while an expert completes a task, and then a post-processing pipeline turns these captures into fully reproducible environments by extracting high-level actions, removing credentials, selecting key checkpoints, and trimming non-essential traffic for replay. With a single expert demonstration, TRACE can produce a fully fledged, self-contained environment for a live website that can be replayed offline, creating a reusable substrate not only for evaluation but also for reinforcement learning on realistic trajectories. We also release an initial demonstration dataset containing 6 human trajectories, each paired with its captured environment, screenshots, page snapshots, videos, and HTTP logs. Overall, TRACE is designed as an extensible collection and replay framework rather than a closed benchmark, making it easy to turn real browsing sessions into reusable environments for future agent training and evaluation, and it includes a minimal example evaluation pipeline for browser-use agents as a reference implementation.

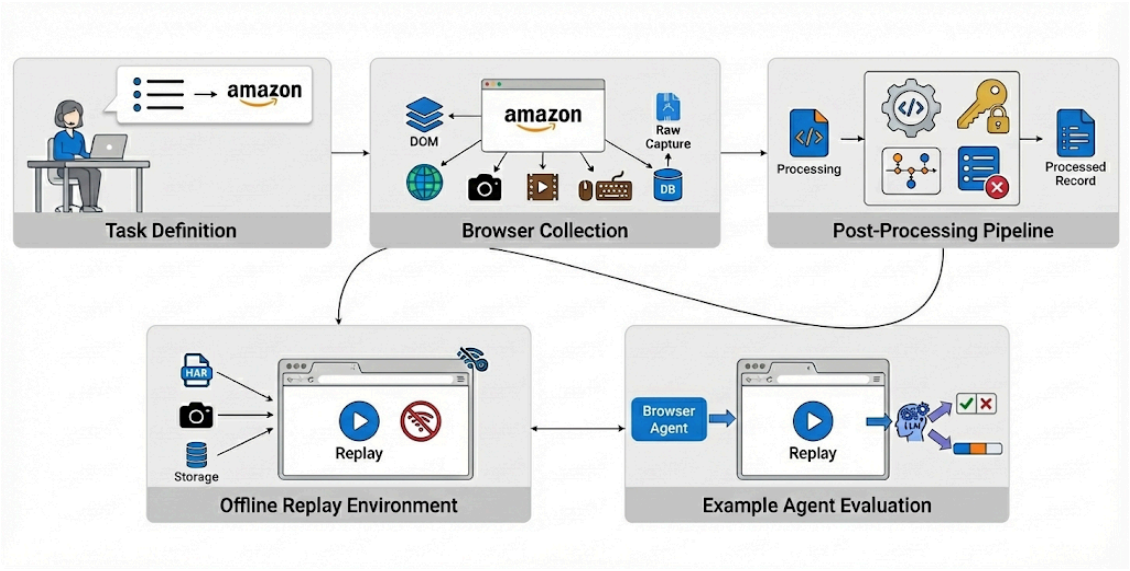

Figure 1: TRACE Pipeline. Collection, Post-processing, Offline Replay, Agent Evaluation.

1 Introduction

Autonomous web and computer-use agents built on large language models are beginning to automate tasks such as booking travel, configuring SaaS tools, and executing complex enterprise workflows. Recent benchmarks—including Mind2Web and its follow-up Mind2Web 2, which annotate web tasks on real sites [5,6], WebArena [7], REAL [8], OSWorld [9], BEARCUBS [1], BrowseComp [2], TheAgentCompany [10], BrowserAgent [11], and GAIA [4]—have driven rapid progress by defining realistic tasks and environments spanning live websites, deterministic replicas, and full operating systems. However, these efforts also highlight how expensive high-fidelity environments are to build and maintain, and how brittle evaluation can be as the web changes over time. WebArena and REAL, for example, require hand-crafted replicas of full websites with custom evaluators for each task [7,8], OSWorld must script fine-grained OS state for hundreds of tasks [9], and TheAgentCompany reports roughly 3,000 person-hours by 20 contributors to design its company-scale work environment and task suite [10]. BEARCUBS and BrowseComp show complementary challenges on the live web, where question validity and trajectories must be continuously refreshed as content changes [1,2].

At the same time, there is growing recognition that many benchmark tasks are only loosely connected to real economic value. A large subset of benchmarks fall into **deep research** tasks (e.g., GAIA, BEARCUBS, BrowseComp, parts of Mind2Web 2) that emphasize hard information-seeking and multi-hop reasoning, often framed as trivia-like questions with little evidence that anyone would pay for their completion [1–4,6]. Others focus on **information-seeking** with light interaction, such as much of Mind2Web's live-web and snapshot tasks [5]. A third category, **execution-based** benchmarks (Mind2Web, WebArena, WebVoyager, OSWorld, REAL, TheAgentCompany) includes more action-oriented tasks—editing content, filing forms, manipulating spreadsheets—but still mixes a large fraction of low-value, atomic operations with only a modest core of realistic, economically meaningful workflows [5,7–10]. Mind2Web 2 explicitly categorizes existing live-web benchmarks by horizon and time variance and shows that most suites concentrate on short-horizon tasks, with relatively few long-horizon workflows that resemble real work [6].

To systematically compare this growing ecosystem of web and computer-use benchmarks, we also built a unified "web-evals" dashboard (https://web-evals.streamlit.app) that ingests GAIA, Mind2Web, Mind2Web2, BrowseComp, WebArena, WebVoyager, REAL, BEARCUBS, Agent-Company, and OSWorld into a common schema, making it easy to filter, search, and inspect tasks across benchmarks [29].

There are promising attempts to move toward explicitly economically grounded evaluations. REAL includes tasks such as scheduling property tours on real estate sites, which more closely resemble paid work but remain limited in number (roughly a hundred tasks overall, of which only a subset are clearly economically significant) [8]. OpenAI's GDPVal benchmark evaluates models on 220 real-world tasks across 44 occupations, constructed from representative work by industry experts, and explicitly measures performance on economically valuable knowledge work rather than synthetic puzzles [21, 22]. SWE-Lancer defines over 1,400 freelance software engineering tasks from Upwork, representing $1M in actual payouts, and maps model performance to monetary value [23]. Mercor's APEX benchmark similarly assesses models on professional work across investment banking, consulting, law, and medicine, but the underlying tasks are not fully released and are not framed as browser environments [1,7,21,24]. These efforts demonstrate that it is possible to ground evaluations in real economic activity, but they

are either domain-specific (software engineering), closed, or do not expose reusable trajectories and environments that other researchers can build on.

The Evaluation practice further complicates progress. Many web and computer-use benchmarks ultimately report a binary or near-binary notion of success—short-answer correctness, a rubric-based pass/fail judgment, or a single scalar task score—even when they internally use richer evaluators [1,2,4,7–9]. This makes leaderboards easy to interpret but limits diagnostic power: it is difficult to attribute failures to specific missteps, to understand how far an agent progressed before derailing, or to reuse trajectories as dense reward signals for reinforcement learning. "An Illusion of Progress?" shows that such coarse metrics can overstate capability gains when benchmarks themselves drift or contain exploitable artefacts [12]. TheAgentCompany explicitly introduces checkpoint-based scoring over realistic employee-style tasks, but building its environment and evaluators required months of manual effort [10]. There remains no lightweight, reusable way to attach granular progress signals to arbitrary browser tasks on the live web.

TRACE (Trajectory Recording and Capture Environments) is motivated by this gap. Rather than handcrafting miniature internets or company intranets, TRACE provides a toolchain that lets an expert complete a task once in an instrumented browser and automatically turns that demonstration into a self-contained environment that can be replayed offline. The collector logs the expert's full trajectory—DOM evolution, HTTP traffic, screenshots, and videos—while they interact with real websites, and a post-processing pipeline converts these raw captures into shareable bundles by parsing tool-like actions, scrubbing credentials, inserting checkpoints, and determining which hosts can be ignored during replay. A replay module then serves the captured assets locally so that agents can be evaluated against an exact snapshot of the original session, without contacting the live web. In contrast to bespoke environment-building projects such as WebArena, REAL, OSWorld, and TheAgentCompany [7–10], TRACE aims to make it as easy to produce a fully fledged browser environment as it is to record a screen-share.

Commercial platforms underscore both the demand for, and the concentration of, high-quality environments. Mechanize, Plato, Deeptune, PreferenceModel, and others offer hosted browser environments for training and evaluation, often based on reproducible copies of common websites and complex workflows [15–19]. Leading labs such as OpenAI, Anthropic, and others run reinforcement learning and large-scale agent training on top of these environments, treating them as core infrastructure and paying correspondingly high prices for access. This has effectively turned browser environments into a strategic, privately held asset: valuable, but closed, expensive, and tied to a specific vendor's stack. While this model has enabled rapid progress inside well-resourced organizations, it also creates a barrier for the broader research community: there is still no practical way to scale this approach to the long tail of real websites, and academic and open-source groups generally cannot afford to license these environments at the scale needed to train and iterate on competitive agents. As a result, the ability to run controlled experiments and iteratively improve agents on realistic, reproducible browser tasks remains largely concentrated in a small number of well-funded companies and labs.

Concretely, our contributions are:

- **TRACE collection tool.** A Playwright-based browser collector (CLI and desktop) that captures expert trajectories—including DOM snapshots, network logs, screenshots, and videos—together with task and metadata, sufficient to reconstruct a local browser environment.

- **Post-processing pipeline.** A sequence of scripts that parse high-level tool-calls from the expert's reasoning, redact credentials, annotate intermediate checkpoints, and filter irrelevant network hosts.

- **Replay environment.** A launcher that replays captured websites entirely offline, including DOM, static assets, and browser state, with options to execute the human trajectory for debugging.

- **Example evaluation runner.** A minimal, working harness for browser-use agents, plus binary and checkpointed LLM-based graders; a parallel OpenAI Computer Use harness exists but is experimental and not part of the validated pipeline.

- **Demo dataset.** An initial open dataset, $dataset, with $n expert demonstrations and linked captured environments derived from Mind2Web tasks [5], intended as a template for future, more economically grounded datasets.

- **Benchmarks dashboard (supporting artifact).** A unified web-evals dashboard for exploring 10+ browser and computer-use benchmarks under a common schema, used in this work to analyze task properties and motivate TRACE's design.

2 Related Work

2.1 Benchmarks and Datasets for Web and Computer-Use Agents

Mind2Web introduced one of the first large-scale, open benchmarks for generalist web agents operating on live websites, with crowdsourced annotations for diverse user tasks [5]. Mind2Web 2 extends this line of work by proposing agent-as-a-judge evaluation and agentic search, further emphasizing live-web evaluation [6]. WebArena instead constructs a **self-hosted web environment** with fully functional websites across four domains (shopping, forums, content management, and collaborative development), along with programmatic validators for task success [7,26]. REAL follows a similar philosophy but focuses on deterministic simulations of 11 widely used websites, providing 112 realistic tasks and a robust evaluation harness mixing programmatic checks and rubric-guided LLM judgments [8]. OSWorld moves beyond the browser to full operating systems, offering 369 tasks across real web and desktop applications and highlighting how far current multimodal agents lag behind humans [9].

BEARCUBS targets computer-using web agents, unifying web and desktop interactions in a benchmark for frontier models [1]. BrowseComp introduces a competition-style benchmark for browsing agents that emphasizes simple yet challenging information-seeking tasks on the live web [2]. TheAgentCompany defines even more consequential tasks involving coding and terminal use, building on the WebArena ecosystem [7,10]. BrowserAgent focuses on grounded, test-time adaptation in real websites using a unified browser agent framework [11]. Together, these works demonstrate a vibrant landscape of web and computer-use benchmarks, but also underscore how each new environment requires substantial bespoke engineering.

Benchmark	Category	Horizon	Limitation
GAIA	Deep research	Medium	Trivia-style, multi-hop QA; browser often unnecessary.
BEARCUBS	Deep research	Short	Complex search questions, manual eval.
BrowseComp	Deep research	Long	Long-horizon scavenger hunts; strong reasoning, not realistic. LM as a judge.
Mind2Web 2	Deep research	Long	Long-horizon, multi-hop QA, LM Code gen Eval.
Mind2Web	Info seeking / Exec	Short–Medium	Mix of IR and simple actions; curated checkpoints, no golden trajectory.
WebVoyager	Info seeking	Short	Long-horizon navigation label, but many tasks are small, atomic info-seeking steps.
WebArena	Execution-based	Short–Medium	Action-based on cloned sites; many tasks are atomic, limited high-value multi-step flows.
REAL	Execution-based	Short–Medium	Deterministic replicas; ~50 with clear economic value;
OSWorld	Execution-based	Short–Medium	Real desktop apps; useful tasks but small dataset and heavy, task-specific state machinery.
Agent-Company	Execution-based	Medium	Realistic workflows, expensive to build.

Table 1: Taxonomy of Existing Browser based Benchmarks and their limitations. Overall they highlight the importance and effort required of offline replicas.

2.2 Trajectory Collection and Self-Improving Agents

Beyond static benchmarks, several works study how to **collect trajectories and let agents learn from interaction**. NNetNav proposes unsupervised learning of browser agents through environment interaction "in the wild," using large-scale, unlabeled trajectories rather than fully curated supervision [3]. Agent Learning via Early Experience explores how generalist web agents can benefit from temporally credit-assigned early experiences to improve downstream performance [13]. SkillWeaver and related efforts develop self-improving agents that discover, refine, and reuse skills via APIs over web environments, often evaluated on WebArena and real websites [28]. These projects highlight the value of flexible data collection pipelines that can serve both supervised learning and RL-style improvement.

TRACE is closer to Mind2Web and WebArena in that it focuses on **expert demonstrations**, but it departs from both by providing end-to-end tooling to automatically capture and freeze the *entire environment* an expert interacted with (network, DOM, screenshots, videos, and browser state), rather than only task descriptions and action sequences [5,7]. This supports use cases ranging from supervised imitation to execution-based evaluation and RL fine-tuning, provided appropriate reward and logging machinery is attached.

2.3 Evaluation Methodology and LLM-as-a-Judge

Recent work has shown that evaluation choices can significantly distort perceived progress in web agents, especially when relying on coarse, binary metrics or poorly specified LLM judges. *An Illusion of Progress?* argues that benchmark design and evaluation practice can overstate gains, and proposes Online-Mind2Web with a more careful LLM-as-a-Judge setup as a partial remedy [12]. Benchmarks such as BrowseComp, REAL, OSWorld, and Beyond Browsing: API-Based Web Agents similarly combine programmatic checks with rubric-guided LLM judgments to handle free-form outputs and multiple valid strategies [2,8,9,14]. TheAgentCompany extends this line of work to consequential, long-horizon "employee-style" tasks, where recognizing partial progress is essential [10].

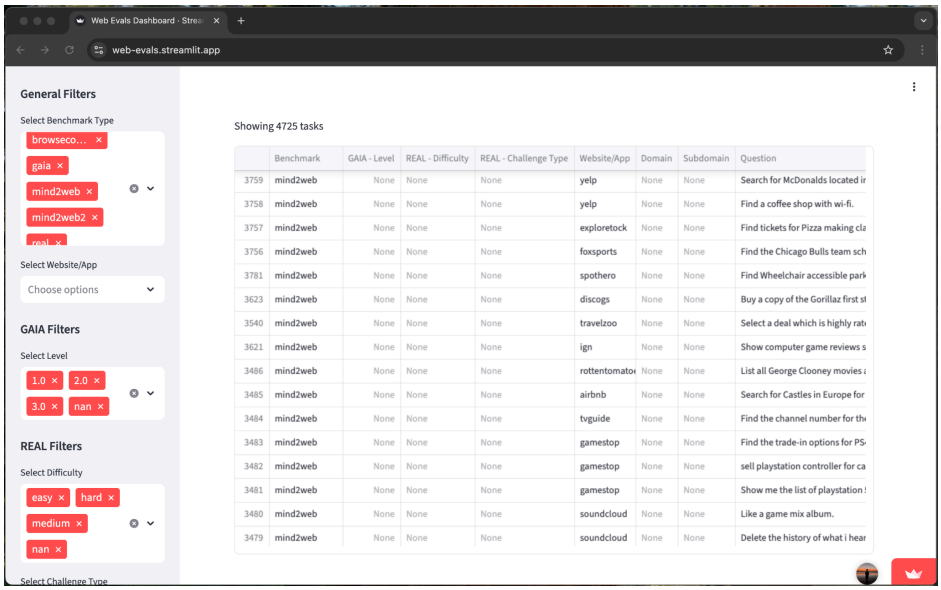

Figure 2: Dashboard that ingests the most used browser benchmarks in a standardized format, allowing for exploration and comparison.

TRACE follows this direction but keeps the evaluation layer intentionally minimal. The repository ships an **example evaluation runner** for browser-use agents with two simple modes: a **binary judge** that decides success or failure based on the task description, human reference, and the agent's final state, and a **checkpoint-based judge** agent that assigns partial credit when the agent reaches predefined semantic milestones, even if it follows a different trajectory than the human. Both modes are semantic rather than action-by-action, allowing alternative valid paths and enabling finer-grained error analysis and denser rewards for RL-style training. However, TRACE does not claim a full-fledged evaluation framework yet; its primary contribution is to provide reusable, replayable environments on top of which more robust and specialized evaluation schemes can be built.

2.4 Commercial Environment Platforms

Beyond academic benchmarks, a small ecosystem of commercial platforms now offers browser and computer-use environments as a service. Mechanize, Plato, Deeptune, PreferenceModel, and AGI Inc provide hosted environments, curated trajectories, and evaluation tooling that are used internally by large labs for training and benchmarking web agents [15–19]. These platforms confirm that environments themselves are a valuable asset, but they are typically proprietary and tightly coupled to a single vendor's infrastructure, limiting their use in open, reproducible research.

TRACE targets a different point in this landscape: it is not an environment service, but an open-source toolchain that enables researchers and practitioners to capture their own environments from the live web and replay them locally. In this sense, TRACE complements commercial platforms by making the core capability—building reproducible environments from demonstrations—available to the broader research and open-source communities.

3 TRACE

3.1 Overview

TRACE is a pipeline for **automated collection, post-processing, and replay of browser environments** from expert demonstrations. It is implemented in a single repository, "Web Environments: Browser Agent Data Collection," a modular design centered around four stages: (1) task collection via an instrumented browser, (2) post-processing, (3) environment replay, and (4) example evaluation.

At a high level, the user initiates the collection tool—either through a command-line interface or a small desktop application—to launch a **Playwright**-based **chromium browser** with a stealth configuration suitable for production websites. While the expert completes a task (e.g., "find flights under a given budget" or "filter products by price"), the tool records all browser events, including navigation, DOM mutations, screenshots, videos, keyboard and mouse actions, and HTTP traffic. These raw traces are stored across an sqlite database, and a structured folder hierarchy (for example, data/captures/<task_id>), forming the basis of an offline environment.

After collection, TRACE's **post-processing pipeline** transforms raw captures into cleaned, annotated artifacts: it takes the database steps table for a task and converts the events logged into high-level tool-calls as a human trajectory baseline, redacts credentials and potential secrets, inserts task-specific checkpoints and reward annotations, and determines which network hosts should be ignored during replay (e.g., analytics or third-party trackers). The resulting artifacts are designed to support reproducible replay and flexible evaluation, without leaking sensitive information.

The **replay module** then launches a local environment that reconstructs the recorded session, using stored DOMs, static assets, and HTTP archives. Agents can interact with this environment through a browser automation layer without contacting the live web, making experiments deterministic and robust to subsequent changes on the original websites. Finally, a minimal evaluation script demonstrates how to connect TRACE environments to the **browser-use** agent

framework and measure success over checkpoints; this runner is meant as a template rather than a definitive benchmark.

3.2 Task Collection

TRACE's collection phase is driven by a small set of commands. From the command line, users can run: uv run trace

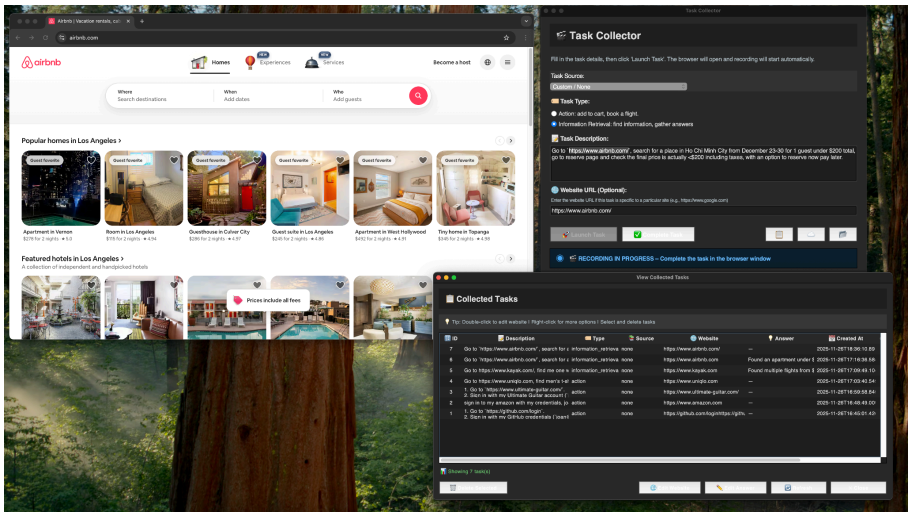

Figure 3: collecting a task using the Tkinter desktop app.

The CLI first prompts for (i) the **source** of the task (e.g., self, Mind2Web seed), (ii) **task type** (information retrieval vs. action), and (iii) a **natural-language description** plus the target **website**.

It then launches a stealth Playwright Chromium instance (custom arguments, anti-detection scripts, optional proxy) and begins recording once the browser opens. During recording, TRACE logs a rich set of events, including:

- Navigation events (page loads, redirects),
- DOM mutations and page-load events,
- Mouse clicks, keyboard input, scrolls, and form submissions,
- Screenshots and video frames,
- HTTP requests and responses, console logs, and storage snapshots.

Recording stops when the collector hits **Enter** (or Ctrl+C) in the terminal; some task types (information retrieval) additionally prompt for a final answer before closing.

For less technical annotators, TRACE also includes a desktop collector implemented with Tkinter. This application mirrors the CLI prompts in a GUI, launches the same stealth browser when **Start Task** is clicked, and writes into the same data/ directory, making it easier to distribute to external contributors who may not be comfortable with command-line tools.

Each completed task produces a **capture bundle** and corresponding database entries. Specifically, TRACE writes:

- Task and event rows into data/tasks.db (SQLite), including all recorded browser events (e.g., page_loaded, dom_mutation, click, type, request_finished),
- Raw per-step logs in data/steps/task_<id>.jsonl,
- A capture bundle in data/captures/task_<id>/ (manifest, HAR, resources, storage),
- Screenshots in data/screenshots/task_<id>/*.png, videos in data/videos/task_<id>/*.webm, and DOM snapshots in data/doms/task_<id>/step_<n>.txt .

In the first post-processing step (postprocess-toolcalls), each task and its associated events in tasks.db are transformed into a consolidated data/tasks.jsonl file, which then serves as the entry point for the rest of the pipeline.

3.3 Post-Processing

The post-processing pipeline consists of several scripts that operate over the raw capture bundles:

1. **Tool-call parsing.** All logs and web events (DOM events, Playwright actions, navigation) are converted into a standardized browser-agent DSL, yielding a sequence of high-level tool calls such as click, type, scroll, back, and goto. This replaces low-level event noise with a human-trajectory representation that agents and analysis code can consume directly.
2. **Credential extraction.** Instead of masking inputs, TRACE identifies credential-related interactions (e.g., login forms, tokens) from DOM and action logs and extracts them into a separate structured field. This allows different harnesses (browser-use, OpenAI, Anthropic, etc.) to inject credentials in whatever format their APIs expect, while enabling substitution with dummy values when sharing environments.
3. **Checkpoint selection.** A lightweight LM "judge" inspects the overall trajectory and DOM states and selects two semantic checkpoints that represent meaningful partial completion (e.g., reaching a results page or filling a form). These checkpoints enable dense reward and partial-credit evaluation rather than only binary task completion.
4. **Ignore-list construction.** Captured HTTP traffic is analyzed to determine which hosts and request patterns can be safely ignored during replay (e.g., analytics, ads, non-essential third parties). The resulting ignored.json is required for stable, deterministic environment replay and reduces noise during evaluation.

After post-processing, each task is represented as a compact JSONL record referencing the cleaned environment artifacts, standardized tool-call trajectory, extracted credential descriptors, checkpoints, and ignore list.

3.4 Environment Replay

The replay component of TRACE exposes a launch-environment command that reconstructs a captured task entirely offline:

uv run launch-environment data/captures/task_<id> --channel chrome --run-human-trajectory --ignore-cache

When invoked on a capture directory, the launcher loads the capture manifest and recording.har, builds a Chromium context with HAR replay routing, and uses the task-specific ignored.json to abort any URL that should not be served (e.g., analytics or other non-essential third parties). All responses are served from the capture bundle and the HAR file; by default no live network is touched unless --allow-network-fallback is explicitly enabled. For each outgoing HTTP request, TRACE first matches on method and URL, then applies **character-based similarity heuristics** over the path and query string to handle dynamically changing parameters (such as timestamps or extra tracking fields) that strict Playwright HAR matching would otherwise miss. When multiple HAR entries remain plausible, an LM-based matcher can break ties and cache the chosen mapping (unless --ignore-cache is set), ensuring deterministic replay across runs. The launcher can also restore storage state (cookies, localStorage, etc.) and optionally execute the human tool-call trajectory for visual debugging. From an agent's perspective this appears as a standard automated browser session, but all network and state transitions are driven by the recorded capture, providing deterministic environments built directly from real expert sessions rather than hand-constructed sites.

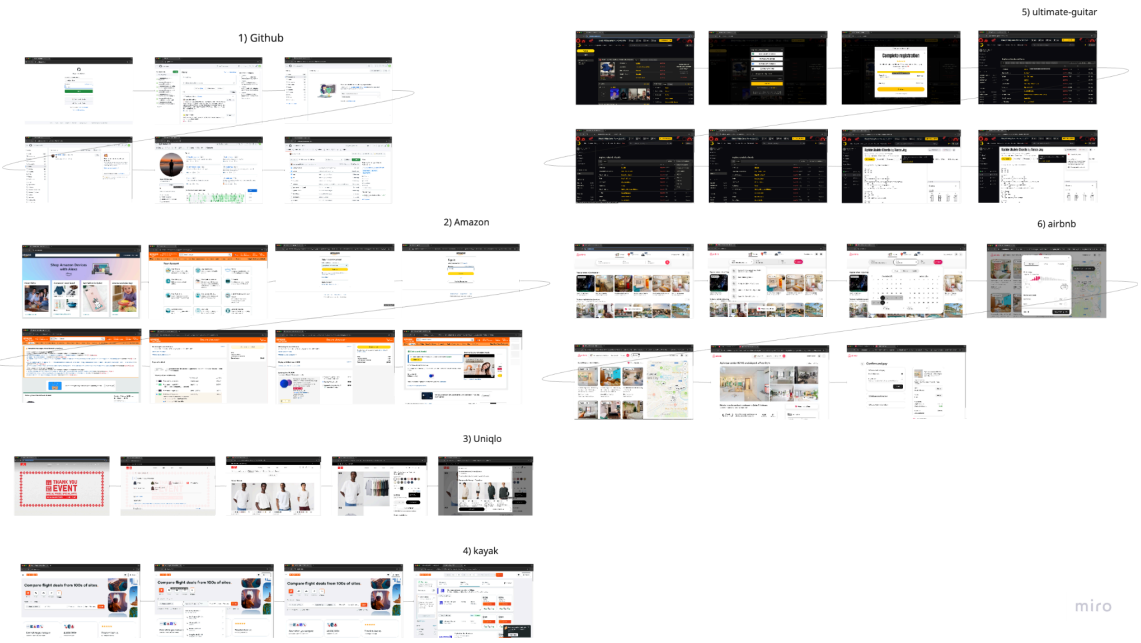

Figure 4: Offline replayed environments of each of the tasks provided, including the replay of sensitive interactions (sign-in/payments) in tasks (1, 2, 5).

3.5 Evaluation

TRACE includes an **example evaluation runner** for the browser-use framework, implemented as a thin wrapper that:

1. Loads a processed task and its associated environment bundle.
2. Launches the replayed environment.
3. Initializes a browser-use agent with the task description and target website.
4. Executes the agent's actions within the replayed environment.
5. Computes success at checkpoints by identifying semantically meaningful checkpoints from the human trajectory and comparing the agent steps against those.

This runner is intentionally minimalist and is primarily intended to demonstrate **how TRACE environments can be plugged into an evaluation harness**, rather than to define a new benchmark or claim rigorous evaluation results. The README also contains a partially implemented harness for OpenAI Computer Use / Operator, but this integration is currently **non-functional** and considered an experiment, not part of the supported pipeline.

Longer term, the same environments could support richer evaluation schemes inspired by REAL, OSWorld, BrowseComp, and Beyond Browsing (e.g., combining functional checks with rubric-guided LLM judgments, or measuring robustness across variations in instructions) [2,8,9,12,14]. However, those extensions are left as future work.

4 Results

Because the TRACE pipeline is primarily an **environment and collection framework**, this section provides a report of 6 collected environments

4.1 Collected Environments

TRACE has been used to produce TRACE Environments, a small public demo dataset hosted on Hugging Face at https://huggingface.co/datasets/josancamon/trace-environments. The current release contains 6 tasks captured on widely used production websites, covering both action-oriented and information-retrieval scenarios.

Each task is stored as a fully replayable environment bundle, including:

- A natural-language task description and task type (action vs. information retrieval),
- An expert "golden" trajectory expressed as typed actions (go_to, click, type, etc.),
- A capture bundle with HTTP logs (HAR), cached resources, and storage snapshots,
- Per-step DOM snapshots, screenshots, and video for visual inspection,

- Task-level metadata such as duration, number of tool calls, and approximate storage footprint.

Table 2 summarizes the six environments in the initial release:

Task	Website	Task Type	Tool Calls	Clicks	Credentials	HTTP Requests (HAR)	Cached Resources	Screenshots	DOM Snapshots	Duration (s)	Storage Size
1	github.com	Action	11	8	✓	715	311	12	35	39.0	64 MB
2	amazon.com	Action	14	11	✓	1,199	468	20	31	69.2	94 MB
3	ultimate-guitar.com	Action	19	15	✓	2,752	534	20	46	54.9	117 MB
4	uniqlo.com	Action	11	10	✗	1,315	783	12	12	46.1	128 MB
5	kayak.com	Info Retrieval	14	11	✗	816	452	32	17	110.7	154 MB
6	airbnb.com	Info Retrieval	15	12	✗	928	544	24	19	103.9	106 MB

Table 2: TRACE Environments dataset (6 captured tasks and their aggregate statistics)..

Together, these environments illustrate that even a small number of expert demonstrations produces rich, high-dimensional traces: each task involves hundreds to thousands of HTTP requests, dozens of DOM snapshots and screenshots, and tens to hundreds of megabytes of captured state, all of which can be replayed deterministically using TRACE's launcher.

4.2 Environment Examples

The six tasks in TRACE Environments are designed to exercise realistic web workflows rather than synthetic toy interaction:

1. GitHub (Action, with credentials). The collector signs in to GitHub, stars a target repository, uses the search bar to find a specific user, and follows that user. The environment captures the full sign-in flow, repository navigation, and user profile interactions.

2. Amazon (Action, with credentials). The collector signs in to Amazon, navigates to current deals (e.g., Black Friday deals), identifies a discounted item meeting specified constraints (such as a minimum percentage discount), adds it to the cart, and proceeds toward checkout using a saved payment method and default address.

3. Airbnb (Information retrieval, no credentials). The collector searches for stays on airbnb.com within a specified location, date range, and budget, applies relevant filters (such as entire place and minimum rating), and inspects one or more candidate listings. The resulting environment includes map interactions, scrolling, and detailed listing pages.

All six environments are replayable offline with TRACE's launch-environment command, allowing agents to be evaluated on realistic, real-world websites without contacting the live web. While the dataset is intentionally small, it demonstrates end-to-end coverage of credential handling, navigation, search and filtering, cart operations, and account-state changes, and serves as a concrete template for scaling TRACE to much larger, more economically grounded task suites.

> it took less than 40 minutes to collect this dataset.

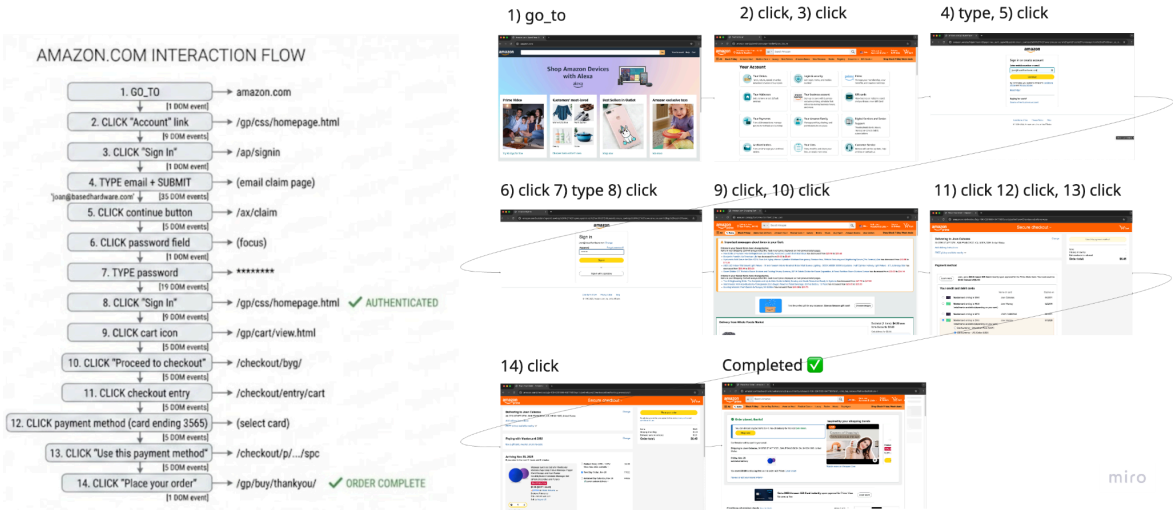

Figure 5: Offline replayed environment and tool call parsing of task "sign in to my amazon account with $email, $password, open the cart and buy the lacrosse ball that is in there with the card finishing in 3565 and the default address."

5 Conclusion

In this work, we presented **TRACE (Trajectory Recording and Capture Environments)**, a toolchain designed to simplify the collection, post-processing, and replay of browser environments for web and computer-use agents. TRACE focuses on **environment-level infrastructure** rather than defining yet another benchmark: it enables experts to perform tasks as usual in an instrumented browser while automatically capturing all relevant signals—DOM, network, interactions, screenshots, and videos—and then converts these captures into replayable environments.

By releasing the TRACE Environments dataset—a public collection of six fully captured, replayable tasks on real websites—and providing a minimal browser-use evaluation runner, TRACE offers a practical starting point for researchers who wish to build their own task suites and evaluation pipelines from live websites. In doing so, it complements existing benchmarks such as Mind2Web, WebArena, REAL, OSWorld, BEARCUBS, TheAgentCompany, BrowseComp, BrowserAgent, GAIA, and API-based web-agent frameworks [1–5,7–11,14]. TRACE's design directly addresses concerns about the fragility and cost of current benchmarks

by offering a scalable, open-source approach to freezing real-world tasks into reusable environments [8,9,12].

Ultimately, TRACE aims to make **environment creation itself** more accessible and reproducible. Rather than relying solely on centralized environment services or one-off benchmarks, researchers can use TRACE to iteratively collect and share high-value tasks tied to their own domains, thereby enriching the ecosystem of web and computer-use evaluations.

6 Next Steps

Several avenues remain for future work, many of which are already hinted at in the mini draft and design notes:

1. **Improved edge-case handling and stealth.** Hardening the collector and replayer against anti-automation measures, dynamic content, and complex authentication flows, potentially drawing on techniques from commercial platforms and recent environment frameworks such as REAL and OSWorld [8,9,15-19].

2. **Richer and more robust evaluation.** Integrating more sophisticated evaluation schemes, including hybrid programmatic + LLM-as-a-Judge scoring, stress tests for robustness (à la Illusion of Progress, BrowseComp, and TheAgentCompany) [2,8,10,12].

3. **RL pipelines.** Extending TRACE environments with step-level rewards and richer checkpoint structures to support RL-based training and finetuning of agents [26-28], including long-horizon credit assignment as in Early Experience [13] and skill discovery as in SkillWeaver [28].

4. **Larger-scale datasets.** Scaling beyond the current 6-task demo to hundreds or thousands of tasks, with a particular focus on economically valuable tasks, as discussed in prior notes on browser-automation economics. This includes expanding coverage over multiple domains (productivity, finance, travel, developer tools) and documenting curation costs.

5. **Beyond the browser to full computer use.** Adapting TRACE's capture and replay mechanisms to OS-level tasks, thereby bridging toward OSWorld-style environments and computer-use agents evaluated in BEARCUBS and RedTeamCUA [1,9,20].

By addressing these directions, TRACE can evolve from an environment collection toolkit into a central component of a broader ecosystem for **transparent, reproducible, and economically grounded** evaluation of web and computer-use agents.

References

1. Y. Song et al. **BEARCUBS: A Benchmark for Computer-Using Web Agents.** arXiv:2503.07919, 2025. Available at: https://arxiv.org/pdf/2503.07919

2. J. Wei et al. **BrowseComp: A Simple Yet Challenging Benchmark for Browsing Agents.** OpenAI blog and report. Available at: https://openai.com/index/browsecomp/

3. S. Murty et al. **NNetNav: Unsupervised Learning of Browser Agents Through Environment Interaction in the Wild.** alphaXiv / arXiv:2410.02907, 2024. Available at: https://www.alphaxiv.org/abs/2410.02907

4. M. J. A. Maia et al. **GAIA: A Benchmark for General AI Assistants in the Wild.** arXiv:2311.12983, 2023. Available at: https://arxiv.org/pdf/2311.12983

5. C. Zhang et al. **Mind2Web: Toward a Generalist Agent for the Web.** arXiv:2306.06070, 2023. Available at: https://arxiv.org/pdf/2306.06070

6. Z. Li et al. **Mind2Web 2: Evaluating Agentic Search with Agent-as-a-Judge.** arXiv:2506.21506, 2025. Available at: https://arxiv.org/pdf/2506.21506

7. S. Zhou et al. **WebArena: A Realistic Web Environment for Building Autonomous Agents.** arXiv:2307.13854, 2023. Available at: https://arxiv.org/pdf/2307.13854

8. D. Garg et al. **REAL: Benchmarking Autonomous Agents on Deterministic Simulations of Real Websites.** arXiv:2504.11543, 2025. Available at: https://arxiv.org/pdf/2504.11543

9. T. Xie et al. **OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments.** arXiv:2404.07972, 2024. Available at: https://arxiv.org/pdf/2404.07972

10. WebArena Team. **TheAgentCompany: A Benchmark of Consequential Tasks for Web and Terminal Agents.** arXiv:2412.14161, 2024. Available at: https://arxiv.org/pdf/2412.14161

11. Y. Feng et al. **BrowserAgent: Grounded Test-Time Adaptation for Web Agents.** arXiv:2510.10666v2, 2025. Available at: https://arxiv.org/html/2510.10666v2

12. T. Xue et al. **An Illusion of Progress? Assessing the Current State of Web Agents.** arXiv:2504.01382, 2025. Available at: https://arxiv.org/pdf/2504.01382

13. S. Murty et al. **Agent Learning via Early Experience: Generalist Web Agents with Temporal Credit Assignment.** arXiv:2510.08558, 2025. Available at: https://arxiv.org/pdf/2510.08558

14. L. Jiang et al. **Beyond Browsing: API-Based Web Agents.** Findings of ACL 2025. Available at: https://aclanthology.org/2025.findings-acl.577.pdf

15. Mechanize. **Mechanize: AI-Native Browser Automation Platform.** Company website. Available at: https://www.mechanize.work

16. Plato. **Plato: Agentic AI Data and Environment Tools.** Company website. Available at: https://plato.so

17. Deeptune. **Deeptune: Evaluation and Training Infrastructure for AI Agents.** Company website. Available at: https://deeptune.com

18. Preference Model. **PreferenceModel: Preference-Based Evaluation and Training for AI Agents.** Company website. Available at: https://www.preferencemodel.com

19. AGI Inc. **REAL Evals and Environment Services for Web Agents.** Company website. Available at: https://agi.inc

20. H. Yuan et al. **RedTeamCUA: Red-Teaming Benchmark for Computer-Use Agents.** arXiv:2502.12911, 2025. Available at https://arxiv.org/pdf/2505.21936

21. OpenAI. **GDPval: Evaluating AI Model Performance on Real-World Economically Valuable Tasks.** Available at https://arxiv.org/pdf/2510.04374.

22. OpenAI. **GDPval Benchmark Card and Evaluation Details.** OpenAI evaluation documentation, 2025. Available at: https://openai.com/index/gdpval/

23. S. Ramaswamy et al. **SWE-Lancer: Benchmarking LLMs on Freelance Software Engineering Tasks.** Available at: https://arxiv.org/pdf/2502.12115

24. Mercor. **APEX: Benchmarking Models on Professional Work Across Multiple Domains.** Company report / arXiv preprint, 2024. Available at https://arxiv.org/html/2509.25721v1

25. J. Deng et al. **WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents.** arXiv:2207.01206, 2022. Available at: https://arxiv.org/abs/2207.01206

26. OpenAI. **Gym.** GitHub repository. Available at: https://github.com/openai/gym

27. PrimeIntellect. **verifiers: Tools for Verifier-Guided RL and Evaluation.** GitHub repository. Available at: https://github.com/PrimeIntellect-ai/verifiers

28. (SkillWeaver authors). **SkillWeaver: Self-Improving Web Agents via Skill Discovery and Reuse.** arXiv preprint, 2024. Available at https://arxiv.org/pdf/2504.07079

29. J. Cabezas. **Web-Evals: Unified Dashboard for Web and Computer-Use Benchmarks.** Streamlit app and REST API, 2025. Available at: https://web-evals.streamlit.app