WEB$HELL: A UNIFIED INTERFACE FOR AGENTIC WEB ACCESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents are becoming direct users of the open web. As they take on increasingly complex and information-intensive tasks, effective web access is becoming a core capability for agentic problem solving. Existing approaches have advanced agentic web access through different *access models*, including search-augmented agents, browser-based agents, and, more recently, filesystem-based agents. Nevertheless, as users expect agents to complete long-horizon, end-to-end tasks, they increasingly need to conduct broad information gathering, perform browser actions, and revisit acquired evidence within a single trajectory. Yet our preliminary analyses reveal that existing access models struggle to meet these demands. They primarily support either information gathering or browser interaction, while combining their capabilities can involve substantial costs or performance trade-offs. To this end, we present WEB$HELL, an interface that unifies web access under a single access model. Using a common schema defined by WEBCONTRACT, WEBCOMPILER compiles live webpages into addressable information and action *objects*. These objects are maintained in WEBCACHE and handled through a shared set of operations, providing agents with consistent, operable, and persistent web access. Through extensive experiments across web search, web navigation, and deep research benchmarks, we show that WEB$HELL achieves strong task performance at lower cost. Our results further indicate the importance of interface design for effective web access, with WEB$HELL opening new possibilities for more capable and efficient LLM agents on the web. [CODE].

1 INTRODUCTION

Large language model agents are becoming direct users of the open web. As they take on increasingly information-intensive and open-ended tasks, effective web access is becoming a core capability for agentic problem solving (Deng et al., 2023; Kim et al., 2026). Yet making effective use of the web is far from trivial, requiring substantial reasoning to locate information and interact with heterogeneous webpages. This raises a fundamental question of how agents should access the web.

Existing research has approached this problem through different *access models*. Search-augmented agents (Li et al., 2025a; Jin et al., 2025; Zheng et al., 2025) operate through search APIs and cleaned text, while browser-based agents (Zhou et al., 2024; Yang et al., 2025; Chae et al., 2025) operate through page observations and actions. More recently, filesystem-based methods (Liu et al., 2026) have enabled agents to store web documents and selectively inspect their contents through shell operations (e.g., `grep`, `read`), reducing the need to read the entire contents. For the most part, these lines of work have made substantial progress within their respective access models. However, as users expect agents to complete long-horizon, end-to-end tasks, agents are increasingly required to perform multiple forms of web access within a single trajectory. For example, a shopping query may require an agent to discover relevant products, navigate pagination to find reviews, and retain useful information for later comparison. Yet these capabilities have largely been developed through separate access models, motivating a rethinking of how they can be integrated for agentic web access.

Accordingly, this paper begins by investigating how well existing access models generalize across web search and web navigation, two representative types of web tasks. The key bottleneck is that each access model supports certain capabilities at the expense of others. We find that (i) search-augmented and filesystem-based agents struggle with web navigation since clean-text representa-

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.

tions often omit page controls, while browser-based agents are less effective at broad information gathering due to their lengthy, noisy observations; (ii) combining search and browser-based access broadens task coverage but incurs substantial costs from repeated webpage interpretation; and (iii) adding filesystem-based capabilities such as `grep` reduces these costs. However, selective inspection can miss the context needed to use page controls, degrading web navigation performance.

Together, these findings highlight three requirements for integrating the capabilities of existing access models. First, ***operability*** requires supporting both efficient information gathering and effective webpage interaction. Second, ***consistency*** calls for unifying webpage representations under common schemas, reducing the need to reinterpret the same page. Finally, ***persistence*** requires retaining retrieved web documents for selective access, lowering cost while preserving operability.

To this end, we present WEB$HELL, an interface that unifies web access under a single access model. Our key idea is to provide agents with an intermediary that standardizes webpage representations and browser actions. Instead of coordinating separate access models, agents interact with WEB$HELL through a request-response loop. Specifically, WEBCONTRACT defines how information and available actions are represented as objects, organized into pages, and accessed through shared operations. Each object combines a *value*, an *address*, and *associations* that preserve its context and references to related objects. Following this contract, WEBCOMPILER transforms live webpages into structured representations using deterministic rules derived from HTML semantics. WEBCACHE then retains the compiled pages for selective access and reuse throughout a task. Importantly, WEB$HELL allows agents to use object addresses to specify which information to inspect or which action to execute in subsequent requests. The interface then carries out the requested operation on the agent's behalf and returns the outcome, thereby enabling unified agentic web access.

We conduct extensive experiments on six benchmarks spanning web search, web navigation, and deep research. Notably, WEB$HELL achieves the highest task performance on all six benchmarks at comparable or lower cost. Our analyses suggest that an effective web interface should make clear how webpage elements relate to one another. These connections help agents gather more evidence, make browser actions more intuitive, and potentially guide exploration beyond the content already inspected. We further show that WEB$HELL remains robust to irrelevant content and maintains its efficiency as retrieved documents accumulate. Together, these findings demonstrate WEB$HELL's effectiveness across complex, information-intensive web tasks. Our contributions are as follows:

- We identify the need for unified agentic web access, investigating the limitations of existing access models and the challenges of integrating their capabilities.
- We propose WEB$HELL, an interface that provides consistent, operable, and persistent web access, enabling agents to handle diverse web tasks under a single access model.
- We demonstrate that WEB$HELL achieves strong task performance and high efficiency across diverse web tasks, providing insights to guide future research on agentic web access.

2 RELATED WORK

Agent for Web. Research on web-capable agents has advanced both information seeking and web navigation. Search-augmented agents interleave web search with reasoning (Li et al., 2025a; Jin et al., 2025), while browser-based agents use page observations to guide their actions (He et al., 2024; Zheng et al., 2024). More recently, inspired by direct corpus interaction (Li et al., 2026; Zhuang et al., 2026; Lu et al., 2026), a complementary line of research has explored filesystem-based web access, where webpages are fetched into persistent workspaces and selectively inspected with shell operations (Liu et al., 2026). Open-source software such as CLI-Anything Browser (Yang et al., 2026) has also adopted a similar approach, supporting web navigation by applying shell operations to the accessibility tree of a webpage. Nevertheless, these lines of work have largely developed web search and web navigation within separate access models. WEB$HELL unifies their capabilities at the interface level, standardizing web access through common representations and operations.

Interface for Agents. Recent work on agent-computer interfaces (Yang et al., 2024) and model harnesses (Lee et al., 2026; Wang et al., 2026; Jiang et al., 2026) has established interface design as an important research problem for improving agent performance. On the web, this perspective initially took the form of simplifying webpage representations, including element filtering and DOM

pruning (Yang et al., 2025; Zhang et al., 2026). Subsequent work (Lee et al., 2025; Kwon & Lee, 2026; Hwang et al., 2026) has moved toward enhancing page representations, organizing webpage elements into regions or summaries that support targeted interaction. Yet these approaches require additional model inference or extensive offline traversal, limiting their practicality for open-web access. Alternatively, WebMCP and NLWeb (Steiner et al., 2026) place the responsibility for interface design on website providers, who encapsulate site-specific functions and content for agents. These works share the premise that interfaces should help agents access and use the web (Zhong et al., 2026), but each applies this premise within a limited scope. WEB$HELL extends it toward general web access, providing agents with an operable, consistent, and persistent interface.

3 PRELIMINARY ANALYSIS: CAN AGENTS EFFECTIVELY ACCESS THE WEB THROUGH EXISTING ACCESS MODELS?

To understand how web access should be designed for LLM agents, we first analyze the capabilities and limitations of existing access models. Specifically, our analysis addresses two questions:

Question I: How well do existing access models generalize across web tasks?

Question II: What limitations remain when access models are combined?

Table 1: Overview of different web access models.

Method	Representation		Interface		
	Clean Text	AXTree	Search API	Browser	Filesystem
Search-augmented					
Fetch-and-Read	✓	✗	✓	✗	✗
Search-o1	✓	✗	✓	✗	✗
WebThinker	✓	✗	✓	✗	✗
Browser-based					
Agent-E	✗	✓	✗	✓	✗
AgentOccam	✗	✓	✗	✓	✗
BrowserUse	✗	✓	✗	✓	✗
Filesystem-based					
Fetch-then-Explore	✓	✗	✓	✗	✓
Hybrid Config.					
$\text{Hybrid}_{\text{SB}}$	✓	✓	✓	✓	✗
$\text{Hybrid}_{\text{SBF}}$	✓	✓	✓	✓	✓
WEB$HELL (ours)	Addressable Objects		Unified Interface		

Benchmarks. We consider two types of web tasks: web search and web navigation. We use WideSearch (Wong et al., 2026) for web search, which requires broad information gathering to populate tables, and WebArena-Verified (hattami et al., 2025) for web navigation, which requires deeper, multi-step website interaction (e.g., reordering products). We randomly sample 100 queries from each benchmark and report item-level F1 for WideSearch and success rate for WebArena-Verified.

Access Models. We evaluate three families of agents with different web access models (Table 1).

- ***Search-augmented Agents***. These agents primarily address web search tasks using search APIs and clean-text representations of webpages. Fetch-and-Read directly reads webpages returned by a web search. Search-o1 (Li et al., 2025a) uses its Reason-in-Documents module to summarize relevant information from these pages, while WebThinker (Li et al., 2025b) uses an explorer agent to browse deeper into websites for additional information.
- ***Filesystem-based Agents***. Fetch-then-Explore (FtE) (Liu et al., 2026) also targets web search tasks, but stores cleaned webpage text in a filesystem workspace. Agents can then selectively locate and read relevant information using shell operations such as `grep` and `read`.
- ***Browser-based Agents***. These agents are primarily designed for web navigation tasks, using structured page observations such as accessibility trees (AXTrees) to guide browser interactions. We include AgentOccam (Yang et al., 2025), Agent-E (Abuelsaad et al., 2024), and BrowserUse (Müller & Žunič, 2024) as representative browser-based agents.

For our analysis, we refer to search-augmented agents and Fetch-then-Explore as *search-oriented*, and web navigation agents as *navigation-oriented*. For evaluation on WebArena-Verified, we augment the search-oriented agents with the browser actions introduced in WebArena (Zhou et al., 2024), while retaining their clean-text observations. We use GPT-5.6-Luna (OpenAI, 2026) as the backbone for all agents (reasoning: *high*). Further details are provided in Appendix B.1.

3.1 ANALYSIS I: CAN EXISTING ACCESS MODELS GENERALIZE ACROSS WEB TASKS?

Existing access models struggle beyond their primary tasks. We report cross-task performance of navigation-oriented agents on WideSearch and search-oriented agents on WebArena-Verified, using the other group's average performance on each benchmark as a reference (Figure 1).

We find that agents consistently fall below the corresponding reference average outside their primary task setting. These cross-task performance gaps can be partly explained by how each access model presents webpages to the agent. Clean-text observations are well suited for gathering information across many pages. However, text extraction can discard the structure and attributes needed to use controls such as buttons and input fields, limiting support for browser interaction. Although URL-based fetching supports limited navigation by allowing agents to follow links, it does not enable them to operate these controls. Browser observations preserve interactive elements, but their length and noise can consume substantial context when gathering information across many sources.

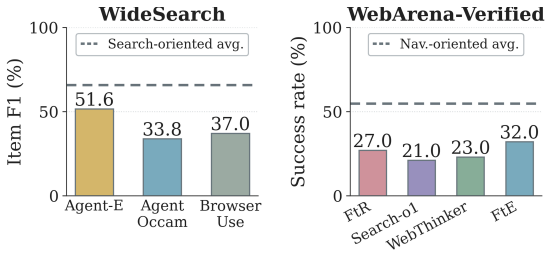

Figure 1: Cross-task performance of navigation-oriented agents on WideSearch and search-oriented agents on WebArena-Verified. Dashed lines indicate the average performance of search-oriented and navigation-oriented agents, respectively.

3.2 Analysis II: What limitations remain when access models are combined?

Combining search and browser-based access supports both web tasks, but incurs substantial cost. Motivated by the findings in Analysis I, we first examine whether a hybrid access model can support both web search and web navigation tasks. We introduce Hybrid$_{SB}$ (*Search + Browser*), which provides clean text for information gathering and AXTrees with browsing tools for interaction. Hybrid$_{SB}$ performs comparably to the search-oriented average on WideSearch (66.0% vs. 65.8%) and exceeds the navigation-oriented average on WebArena-Verified (74.0% vs. 54.7%). However, it incurs substantially higher input token usage and LLM cost (Table 2). We observe that Hybrid$_{SB}$ repeatedly accesses the same source through both clean text and AXTrees for two purposes: (1) when the desired information is missing or unclear in one representation, agents use the other to find it. (2) agents cross-check page representations to verify their completeness. These repeated checks significantly increase the cost, indicating the need for a unified access model that reduces reliance on separate representations.

Table 2: Mean input tokens (k/call) and LLM cost (USD/100 calls). Δ: relative increase over reference averages.

Agent	Input tokens		LLM cost	
	Avg.	Δ	Avg.	Δ
WideSearch				
Search avg.	46.7	--%	0.33	--%
Hybrid$_{SB}$	182.2	+290%	1.21	+272%
WA-verified				
Nav. avg.	10.2	--%	0.19	--%
Hybrid$_{SB}$	74.2	+626%	0.39	+103%

Adding filesystem access with shell operations creates a tradeoff between web search and web navigation. We next examine Hybrid$_{SBF}$, which extends Hybrid$_{SB}$ with a filesystem workspace and shell operations. We observe that Hybrid$_{SBF}$ performs better at lower cost on WideSearch, where tasks require broad source coverage but only specific evidence from each source (Figure 2, left). However, performance declines on WebArena-Verified (Figure 2, right), where completing tasks requires identifying and using page controls. Selective inspection may reveal relevant text without showing how to interact with the page, causing agents to exhaust their turn budget searching for the required controls.

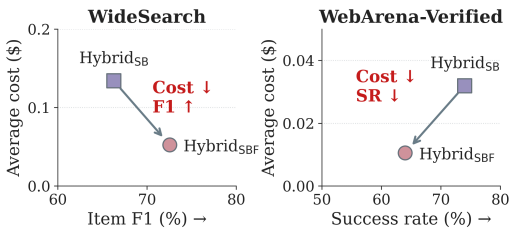

Figure 2: Performance and cost of hybrid configurations on WideSearch and WA-Verified.

3.3 Insights from Preliminary Analysis

Our analyses identify three requirements for agentic web access. **Operability** preserves the context needed for browser interaction while enabling efficient information gathering. **Consistency** calls for a common representation of information and browser actions, reducing repeated webpage interpretation. **Persistence** retains retrieved web documents for selective access, reducing the need to read pages in full. Together, these requirements guide the design of a unified web access model.

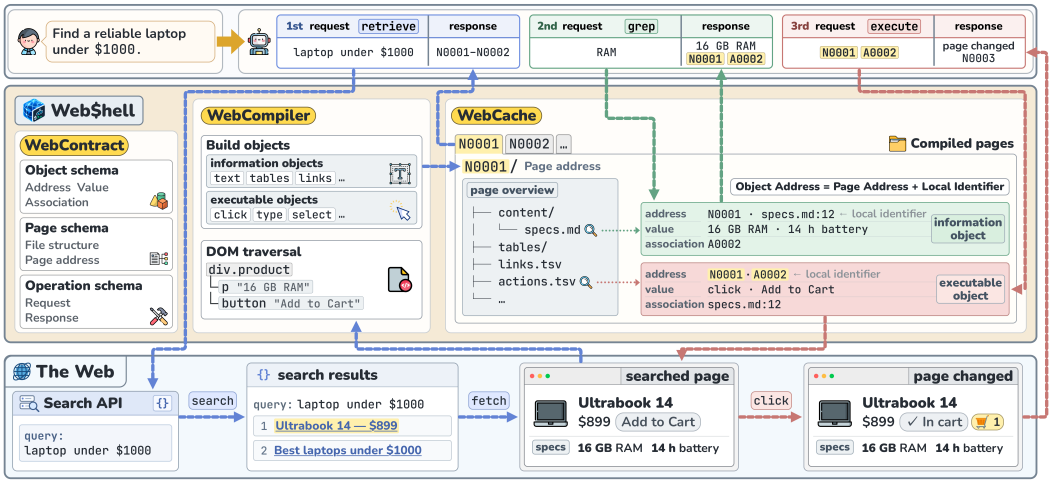

Figure 3: Overview of WEB$HELL. The agent retrieves webpages, selectively inspects cached objects, and executes associated actions through a unified request–response interface.

4 WEB$HELL

To this end, we introduce WEB$HELL, an interface for agents that unifies web access under a single access model. Specifically, our approach builds on three core components. (1) WEBCONTRACT establishes common rules for representing and accessing webpages across websites. (2) WEBCOMPILER compiles fetched webpages into operable representations that follow these rules. (3) WEBCACHE preserves compiled webpages for selective and continued access throughout the task.

4.1 INTERACTION LOOP

Given a user query, the agent interacts with WEB$HELL through the *Request–Response* loop illustrated in Figure 3. Requests and responses follow the common rules defined by WEBCONTRACT.

Agent side. At each step, the agent considers its current context to determine what information or interaction it needs. It selects an operation (e.g., `retrieve`, `grep`, `execute`) and sends a request. After receiving the response, it chooses its next request or produces a final answer.

Interface side. The interface carries out the requested operation on the agent's behalf. For `retrieve`, it searches the web, fetches the top-ranked webpages, and captures their current states. WEBCOMPILER then compiles each captured webpage into *information objects* that represent its content and *action objects* that describe its available browser actions. It organizes these objects into a common file structure following WEBCONTRACT, providing the agent with a consistent view across websites. Each compiled page is assigned a page address and stored in WEBCACHE for subsequent requests. For `grep`, the interface locates matching content within these cached pages. For `execute`, it invokes the specified action object to perform the corresponding interaction on the live webpage. Updated page states are compiled and reflected in WEBCACHE, while earlier states are kept read-only. The interface returns the requested information or execution outcome in its response.

4.2 WEBCONTRACT: STANDARDIZING WEB ACCESS

Object Schema. The first step in designing WEB$HELL is to determine what *objects* agents should operate on. Webpages encode content and controls in individual elements, but the context needed to interpret or use them may be encoded in separate elements. For example, a price entry is only meaningful together with the product name and option label, which may appear in separate elements. We therefore define an *object* as a unit of information or action with the following three properties.

- **Address.** A stable reference through which the agent can identify the object.
- **Value.** The information or action represented by the object.
- **Association.** The context needed to interpret the value, including references to other objects.

This shared schema provides a consistent, addressable, and directly interpretable unit of interaction.

Page Schema. The page schema defines a common file structure for representing a webpage. A unique *page address* identifies each fetched webpage as a whole. Each object has a unique local identifier within its page. This identifier is further combined with the page address to form its *object address*. Within each page, objects are organized into groups using a finite set of types, such as text, tables, links, and actions, grounded in HTML semantics. Text objects are further organized into sections based on the webpage's heading structure. A table of contents lists the available object groups and shows how to access them. The detailed page schema is presented in Appendix C.1.

Operation Schema. The operation schema defines how agents communicate with the interface through *requests* and *responses*. Information and action objects serve as the basic units of operation. An agent-to-interface request specifies an operation type, its arguments, and an object address when needed. An interface-to-agent response returns the outcome along with any resulting content and references. These common rules apply to operations such as `retrieve`, `grep`, `read`, and `execute`. The detailed operation schema and the list of operations are provided in Appendix C.1.

4.3 WEBCOMPILER: COMPILING AN OPERABLE OBSERVATION SPACE

Object Compilation. WEBCOMPILER applies deterministic rules derived from HTML semantics without relying on model inference. Tags and roles guide how element content and properties are compiled into an object's *value*. Nesting and references indicate how elements are related, providing the *associations* needed to interpret that value. For example, a dropdown menu offers users several options to choose from. In HTML, the menu is defined by a `<select>` tag, with each choice specified by a nested `<option>` tag. The compiler represents this menu as an action object whose *value* specifies what the agent can do (select) and which options it can choose. The context for this action comes from the text in a `<label>` tag, such as *"clothing size"*. The compiler includes this context in the object's *association*, indicating what the options represent. We implement these transformation rules as a finite set of *constructors*, which the compiler applies while traversing the captured DOM tree in document order. Constructors can recursively process nested elements, linking the resulting objects through references. When an element has no dedicated constructor, the compiler processes its text content and recursively applies the same transformation rules to nested elements. Text fragments separated by formatting tags are combined into a single text object.

Finally, WEBCOMPILER organizes the resulting objects into a file tree following the page schema in WEBCONTRACT, preserving references between related objects. The compiled page is registered in WEBCACHE and assigned a *page address*, allowing agents to access it in subsequent requests.

4.4 WEBCACHE: PROVIDING PERSISTENT AND STATEFUL WEB ACCESS

Persistent Access. After receiving web search results, the interface fetches the top-ranked webpages, compiles them, and stores the compiled pages in WEBCACHE. Preparing these pages in advance provides three benefits. First, the interface reports failed fetches (e.g., 404 errors or CAPTCHA challenges), so the agent does not need to spend additional turns identifying these access failures. Second, the agent can get a quick overview of what each page offers without reading it in full. The compiler provides a table of contents showing the available object groups and how to access them. Third, operations support selective access to cached pages, allowing the agent to explore multiple webpages more efficiently and revisit them as the task progresses.

Stateful Access. A core feature of WEB$HELL is that WEBCACHE tracks page state changes while preserving access to earlier states. When the agent submits an `execute` request, the interface uses the connection retained during compilation to operate the corresponding control on the live webpage. If the action changes the webpage, the interface compiles the updated page state and caches it under a new page address. The action that produced the change is logged and reported to the agent in the response. The earlier state remains read-only, so its objects can still be inspected even when the corresponding actions are no longer available on the live webpage. This allows the agent to revisit past page states and review how the task progressed.

5 EXPERIMENTS

Benchmarks. We evaluate WEB$HELL across three categories of web tasks: web search, web navigation, and deep research. Web search includes WideSearch (Wong et al., 2026), which requires collecting information across many sources into tables, and SealQA (Pham et al., 2026),

Table 3: Performance (upper) and efficiency (lower) comparison on web search, web navigation, and deep research benchmarks. Efficiency reports average turns, tokens, and cost per task.

Agent	Web Search		Web Navigation		Deep Research	
	WideSearch *Item F1* ↑	**SealQA** *Acc.* ↑	**WA-Verified** *SR* ↑	**WebWalker** *Acc.* ↑	**DR-Bench** *RACE Overall* ↑	**DW-Bench** *Overall* ↑
$\text{Hybrid}_{\text{SBF}}$	0.738	0.646	0.578	0.590	<u>0.528</u>	28.37
Fetch-then-Explore	<u>0.745</u>	<u>0.665</u>	0.264	<u>0.636</u>	0.527	28.27
CLI-Any-Browser	0.660	0.650	0.575	0.588	<u>0.528</u>	26.59
WEB$HELL (ours)	**0.782**	**0.720**	**0.679**	**0.671**	**0.534**	**31.11**

Agent	Web Search			Web Navigation			Deep Research		
	Turns ↓	**Tok. (M)** ↓	**Cost ($)** ↓	**Turns** ↓	**Tok. (M)** ↓	**Cost ($)** ↓	**Turns** ↓	**Tok. (M)** ↓	**Cost ($)** ↓
$\text{Hybrid}_{\text{SBF}}$	28.3	1.84	0.070	<u>24.1</u>	<u>0.58</u>	<u>0.024</u>	30.9	2.75	0.111
Fetch-then-Explore	<u>26.3</u>	**1.24**	<u>0.063</u>	27.3	0.60	0.027	<u>27.8</u>	<u>2.20</u>	<u>0.097</u>
CLI-Any-Browser	47.4	3.08	0.102	25.6	1.15	0.044	52.4	4.03	0.137
WEB$HELL (ours)	**20.8**	<u>1.51</u>	**0.062**	**15.0**	**0.47**	**0.022**	**17.6**	**1.87**	**0.096**

which features challenging questions with conflicting search results. For web navigation, we prioritize controlled evaluation settings. WebArena-Verified (hattami et al., 2025) provides simulated environments to evaluate task completion under reproducible conditions, while WebWalkerQA (Wu et al., 2025) requires browsing within the same domain from a given starting URL. For deep research, we cover research report writing in DeepResearchBench (Du et al., 2026) and quantitative analysis of evidence in DeepWeb-Bench (Xie et al., 2026). For evaluation, we adopt the official metrics for each benchmark. Further benchmark details are available in Appendix A.1.

Baselines. We mainly compare WEB$HELL with filesystem-based approaches, motivated by their competitive performance in our preliminary analysis (Section 3.2). We include: (1) Fetch-then-Explore (FtE) (Liu et al., 2026), which stores cleaned webpage text in files for selective reading and revisit; (2) $\text{Hybrid}_{\text{SBF}}$, a unified access model with a direct combination of search, browser, and filesystem access. Additionally, we include (3) CLI-Anything (Yang et al., 2026), a software tool for shell-based browsing over accessibility trees, as a practical alternative to research baselines.

Implementation Details. We use `gpt-5.6-luna` as the main-agent backbone with the reasoning effort set to `high`. The turn budgets for Web Search, Web Navigation, and Deep Research are set to 100, 50, and 200, respectively. For baselines, we follow their official implementations. Experiments for each benchmark are run concurrently to reduce temporal variation in web search results. We use the Serper API for web search*, and exclude HuggingFace pages containing reference answers to prevent answer leakage. Further implementation details are provided in Appendix A.3.

5.1 MAIN RESULTS

WEB$HELL performs competitively across diverse web tasks. Table 3 (Upper) compares WEB$HELL with existing access models across Web Search, Web Navigation, and Deep Research.

On ***Web Search***, WEB$HELL outperforms Fetch-then-Explore, the strongest baseline on both WideSearch and SealQA. This advantage may partly stem from how WEBCONTRACT organizes content into common object types, such as body text, tables, and links, and defines consistent operations for accessing them across websites. This distinction matters because agents must locate and revisit evidence across sources while building an answer. A common page organization gives them predictable locations to inspect, helping them direct subsequent requests to relevant content.

On ***Web Navigation***, $\text{Hybrid}_{\text{SBF}}$ performs similarly to CLI-Anything Browser on WebArena-Verified, whereas WEB$HELL achieves the best performance. This comparison suggests that combining access capabilities alone does not necessarily improve task completion. Navigation also requires agents to identify available actions and verify their outcomes. WEB$HELL supports these requirements by explicitly listing page controls and providing feedback on the effects of actions. Specifically, WEBCOMPILER compares previous and updated page states and reports what has

*`https://serper.dev`

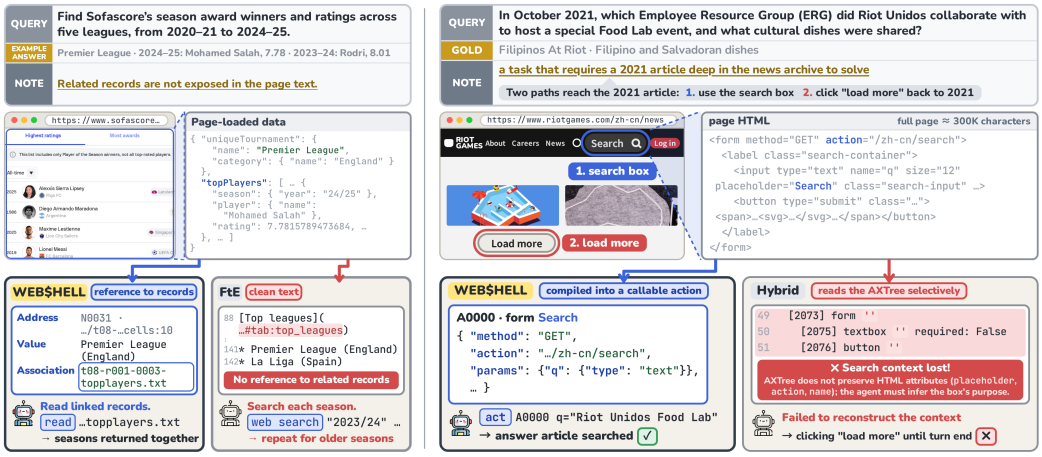

Figure 4: Case studies of WEB$HELL vs. FtE in search (left) and Hybrid$_{SBF}$ in navigation (right).

changed. Together with the Web Search results, these gains suggest that a unified interface can support effective interaction while retaining the benefits of selective information access.

On ***Deep Research***, WEB$HELL shows its clearest improvement on DeepWeb-Bench, which requires agents to gather numerical data from multiple sources and derive answers through multi-step reasoning. When an evidence search (e.g., using `grep`) does not return the required information, agents may be uncertain about where to look next. In such cases, WEB$HELL's object association field may guide further inspection by providing references to related sections and tables, a role we examine further in Section 6. Overall, WEB$HELL's benefits are not limited to web search and navigation, but also extend to demanding deep research.

WEB$HELL achieves higher performance at lower cost. As shown in Table 3 (Lower), the advantage is particularly evident in Deep Research, where agents must search and synthesize information from multiple sources. Here, CLI-Anything Browser uses nearly three times as many turns as WEB$HELL (52.4 vs. 17.6), while Hybrid$_{SBF}$ uses nearly twice as many (30.9 vs. 17.6). These results demonstrate the value of unified access for effective and cost-efficient agentic web access.

6 ANALYSIS

Here, we analyze the effectiveness and robustness of WEB$HELL. See Appendix B.2 for details.

**How does WEB$HELL help information seeking?** We observe that WEB$HELL achieves the highest evidence coverage on both web search benchmarks, measured by how much gold evidence is present in the agent's context (Figure 5). To understand how it gathers more evidence, we conduct a case study comparing WEB$HELL with FtE, the strongest baseline on WideSearch (Figure 4, left). The query asks for season award winners and their ratings across five football leagues. A core distinction is that WEB$HELL provides guidance for further inspection alongside the information it returns. FtE's clean-text representation lists the *"Premier League"* but omits the season results required to answer the query, leading to additional web searches. In contrast, WEB$HELL presents the league as an information object, with references to related season data in its association field. These findings suggest that WEB$HELL facilitates further exploration.

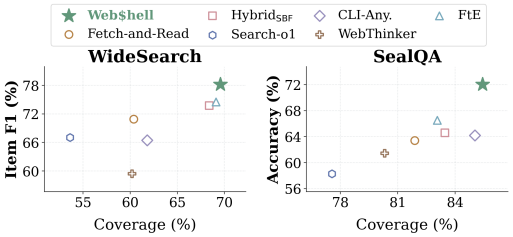

Figure 5: Evidence coverage and Accuracy on WideSearch and SealQA. Search-augmented agents are additionally included for comparison.

**How does WEB$HELL help interaction?** A key feature of WEB$HELL is that webpage controls are compiled into action objects with the context needed to interpret and use them. Figure 4 (right) illustrates this benefit in a task requiring an older article from a news archive. The agent can either search for the article directly through the search form or use *load more* to browse older entries. Hybrid$_{SBF}$'s observation, however, presents the form as an unlabeled text box and button, requiring

the agent to reconstruct the context needed to understand and use it. Hybrid instead repeatedly selects *load more*, exhausting its turn budget without finding the article. In contrast, WEBCOMPILER preserves the label and query parameter in a self-contained action object, enabling WEB$HELL to use the site search and reach the article. This case illustrates how WEB$HELL supports web navigation by reducing the need to infer how controls work.

How does unified access to information and actions help? WEB$HELL presents information together with references to its associated actions, allowing agents to move directly from reading page content to interacting with it. For example, a `grep` result for *Set status* can provide the action reference needed to open the status dialog. To examine the benefit of this connection, we remove these references and require agents to obtain available actions from a separate listing. We observe that this separation consistently degrades performance and increases cost across the two web navigation benchmarks (Figure 6). Connecting information and actions within the interface thus plays an important role in task completion.

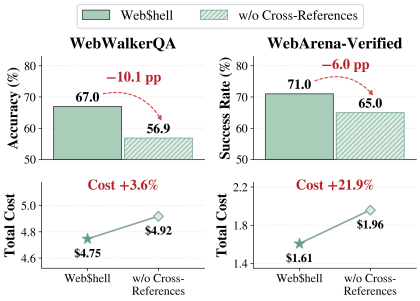

Figure 6: Effect of removing associations between information and actions.

WEB$HELL remains robust to irrelevant information. To test robustness to irrelevant information, we manually inject 50, 100, 200, or 400 distractor pages into WEB$HELL's WEBCACHE. We find that WEB$HELL maintains largely stable accuracy and inference costs even as the amount of irrelevant information increases substantially (Figure 7). The baselines also maintain accuracy, but Fetch-then-Explore's cost more than doubles and $\text{Hybrid}_{\text{SBF}}$'s rises steadily. Fetch-then-Explore often lists the entire workspace before selecting what to read, so the agent processes more file names as pages accumulate. $\text{Hybrid}_{\text{SBF}}$ often avoids this step by searching a fetched page directly with `grep`, using the file path returned by `fetch`. However, keeping track of these paths becomes harder as pages accumulate, and path errors sometimes lead the agent back to a full workspace listing. In contrast, WEB$HELL's page address system provides short references that agents can reuse in subsequent searches and reads, suggesting that reliable addressing helps selective access remain efficient as irrelevant content accumulates.

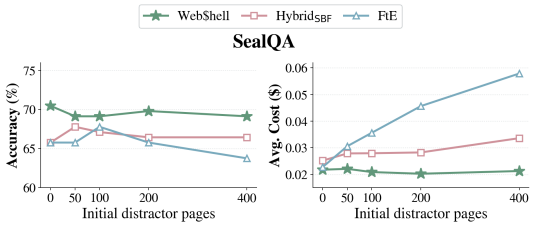

Figure 7: Effect of distractor pages on SealQA.

WEB$HELL supports broader web access at a stable inference cost. We vary the number of prefetched pages per search ($k \in \{5, 7, 10\}$) on WideSearch to examine how WEB$HELL handles a growing cache. Performance varies modestly, while average inference cost remains nearly unchanged (Figure 8). The agent continues to inspect a similarly sized subset even as more pages become available. These results support an interface that keeps a broad collection of sources accessible while allowing the agent to determine the scope of inspection. WEB$HELL supports this flexibility without substantially increasing inference cost across the tested settings.

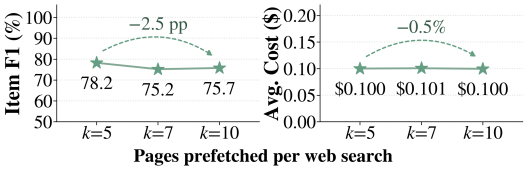

Figure 8: Effect of pages prefetched per search.

7 CONCLUSION

This paper proposes WEB$HELL, a unified interface for agentic web access. It standardizes webpage information and actions as addressable objects, compiles live webpages into operable representations, and preserves them for selective and stateful access throughout a task. Our experiments demonstrate strong performance and efficiency across web search, web navigation, and deep research, while further analyses show the benefits of connecting information with associated actions and maintaining persistent, addressable access as retrieved content accumulates. We hope that WEB$HELL will contribute to future work on more effective interfaces for agentic web.

AI USE STATEMENT

LLMs were used to support manuscript writing and editing for clarity and readability, as well as to assist with code implementation for the experiments. All AI-assisted text was reviewed by the authors, and all AI-generated code was manually inspected and verified before being used in the experiments. The authors retain full responsibility for the final manuscript and experimental code, including any content produced with AI assistance.

REFERENCES

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. Agent-e: From autonomous web navigation to foundational design principles in agentic systems, 2024. URL https://arxiv.org/abs/2407.13032.

Hyungjoo Chae, Namyoung Kim, Kai Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 63707–63738, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a00548031e4647b13042c97c922fadf1-Paper-Conference.pdf.

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. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114. Curran Associates, Inc., 2023. doi: 10.52202/075280-1220. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Licheng Zhang, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 42414–42448, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/465f22be10e07b301c6ed58f0472f704-Paper-Conference.pdf.

Amine El hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. Webarena verified: Reliable evaluation for web agents. In *Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=94tlGxmqkN.

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/.

Jayoo Hwang, Xiaowen Zhang, and Vedant Padwal. Webchallenger: A reliable and efficient generalist web agent, 2026. URL https://arxiv.org/abs/2606.10423.

Pengcheng Jiang, Zhiyi Shi, Kelly Hong, Xueqiang Xu, Jiashuo Sun, Jimeng Sun, Hammad Bashir, and Jiawei Han. Harness-1: Reinforcement learning for search agents with state-externalizing harnesses, 2026. URL https://arxiv.org/abs/2606.02373.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Sunghwan Kim, Ryang Heo, Yongsik Seo, Jinyoung Yeo, and Dongha Lee. Agenticshop: Benchmarking agentic product curation for personalized web shopping. In *Proceedings of the ACM*

Web Conference 2026, WWW '26, pp. 2489–2500, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723070. doi: 10.1145/3774904.3792724. URL https://doi.org/10.1145/3774904.3792724.

Donguk Kwon and Dongha Lee. Region4web: Rethinking observation space granularity for web agents, 2026. URL https://arxiv.org/abs/2605.07134.

Dongjun Lee, Juyong Lee, Kyuyoung Kim, Jihoon Tack, Jinwoo Shin, Yee Whye Teh, and Kimin Lee. Learning to contextualize web pages for enhanced decision making by llm agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 81039–81072, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c995a913bc20ca39ce2231b8973be90a-Paper-Conference.pdf.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=tmbOUyFx3R.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5420–5438, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.276. URL https://aclanthology.org/2025.emnlp-main.276/.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 120091–120131. Curran Associates, Inc., 2025b. doi: 10.52202/085713-4011. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/ae03bdef276132fae089692445725635-Paper-Conference.pdf.

Zhuofeng Li, Haoxiang Zhang, Cong Wei, Pan Lu, Ping Nie, Yi Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Yuyu Zhang, Jianwen Xie, Yejin Choi, James Zou, Jiawei Han, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Beyond semantic similarity: Rethinking retrieval for agentic search via direct corpus interaction, 2026. URL https://arxiv.org/abs/2605.05242.

Qi Liu, Yiqun Chen, Zidan Chen, Yan Gao, Yi Wu, Yao Hu, Jiaxin Mao, Fengbin Zhu, and Tat-Seng Chua. Fetch-then-explore: Decoupling selection from extraction over a persistent workspace for search agents, 2026. URL https://arxiv.org/abs/2608.02097.

Yi Lu, Zhuofeng Li, Ping Nie, Haoxiang Zhang, Yuyu Zhang, Kai Zou, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Dr-dci: Scaling direct corpus interaction via dynamic workspace expansion, 2026. URL https://arxiv.org/abs/2606.14885.

Magnus Müller and Gregor Žunič. Browser use: Enable ai to control your browser, 2024. URL https://github.com/browser-use/browser-use.

OpenAI. GPT-5.6: Frontier Intelligence That Scales with Your Ambition. https://openai.com/index/gpt-5-6/, July 2026. Accessed: 2026-09-15.

Thinh Pham, Nguyen Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. Sealqa: Raising the bar for reasoning in search-augmented language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 98989–99017, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/a0e0f9c39b7f6d07a266a3326216ec40-Paper-Conference.pdf.

Aaron Steiner, Ralph Peeters, and Christian Bizer. Mcp vs rag vs nlweb vs html: A comparison of the effectiveness and efficiency of different agent interfaces to the web. In *Proceedings of the ACM Web Conference 2026*, WWW '26, pp. 8493–8496, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723070. doi: 10.1145/3774904.3792893. URL https://doi.org/10.1145/3774904.3792893.

Xiaoxuan Wang, Haixin Wang, Alexander Taylor, Jason Cong, Yizhou Sun, and Wei Wang. Harnessbridge: Learnable bidirectional controller for llm agent harness, 2026. URL https://arxiv.org/abs/2606.12882.

Ryan Wong, Jiawei Wang, Junjie zhao, Li Chen, Yan Gao, Zhanglong, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, YANG WANG, and Ke Wang. Widesearch: Benchmarking agentic broad info-seeking. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 10012–10086, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1119587863e78451f080da2a768c4935-Paper-Conference.pdf.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. WebWalker: Benchmarking LLMs in web traversal. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10290–10305, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.508. URL https://aclanthology.org/2025.acl-long.508/.

Sixiong Xie, Zhuofan Shi, Haiyang Shen, Jiuzheng Wang, Siqi Zhong, Mugeng Liu, Chongyang Pan, Peilun Jia, Baoqing Sun, Xiang Jing, and Yun Ma. Deepweb-bench: A deep research benchmark demanding massive cross-source evidence and long-horizon derivation, 2026. URL https://arxiv.org/abs/2605.21482.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 97533–97565, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f2c6e459b95694a24ac69c469a4ee746-Paper-Conference.pdf.

Yuhao Yang, Tianyu Fan, and Chao Huang. Cli-anything: Towards agent-native computer use, 2026. URL https://arxiv.org/abs/2606.03854.

Jiayuan Zhang, Kaiquan Chen, Zhihao Lu, Enshen Zhou, Qian Yu, and Jing Zhang. Prune4web: Dom tree pruning programming for web agent. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'26/IAAI'26/EAAI'26. AAAI Press, 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i41.40772. URL https://doi.org/10.1609/aaai.v40i41.40772.

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.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.22. URL https://aclanthology.org/2025.emnlp-main.22/.

Shanshan Zhong, Kate Shen, and Chenyan Xiong. Agentwebbench: Benchmarking multi-agent coordination in agentic web. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=aNtMUFWjnD.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf.

Shengyao Zhuang, Yuansheng Ni, Hengxin Fun, Jimmy Lin, and Xueguang Ma. Towards retrieving interaction spaces for agentic search, 2026. URL https://arxiv.org/abs/2606.06880.

A Experimental Setup

A.1 Benchmarks and Metrics

We evaluate on six benchmarks covering web search, web navigation, and deep research. Together, they test information gathering across sources, interaction with websites, and synthesis of evidence into reports and quantitative answers.

Web Search.

- **WideSearch** (Wong et al., 2026) evaluates broad information gathering across webpages. Agents find entities that meet a query's conditions and collect their attributes in a table. The benchmark contains 100 English and 100 Chinese queries. We report item-level F1 by comparing generated tables with reference tables. We follow the benchmark's column-specific evaluation criteria, using rule-based checks for exact matches, numerical tolerances, dates, and URLs, and `gpt-5.6-luna` for columns requiring semantic judgments. The same model also aligns differently worded entity names before row matching.
- **SealQA** (Pham et al., 2026) focuses on questions with conflicting, misleading, or incomplete search results. Answering them requires resolving ambiguity or combining evidence across sources. We evaluate all 254 questions in Seal-Hard and report answer accuracy.

Web Navigation.

- **WebArena-Verified** (hattami et al., 2025) provides reproducible, self-hosted websites for controlled evaluation of web interaction. Tasks involve finding information, navigating between pages, and modifying content on shopping sites, discussion forums, and other web applications. The benchmark contains 812 tasks, but 23 could not be reproduced because the required Wikipedia corpus could not be downloaded. Our evaluation therefore covers 789 tasks. We report task success rate using the benchmark's deterministic evaluator, which checks final responses and recorded interactions against task-specific completion criteria.
- **WebWalkerQA** (Wu et al., 2025) evaluates evidence gathering through navigation from a given starting URL. Questions range from locating evidence on one subpage to combining information across multiple subpages. We exclude questions whose reference pages are inaccessible or no longer contain the evidence for the gold answer after website updates. We report answer accuracy on the remaining 420 questions.

Deep Research.

- **DeepResearchBench** (Du et al., 2026) contains 100 tasks that require agents to synthesize web evidence into research reports. We report RACE Overall, which compares generated and reference reports across comprehensiveness, insight/depth, instruction following, and readability. An LLM judge applies task-specific criteria and weights to both reports. The overall score ranges from 0 to 1, with 0.5 indicating equal scores for the generated and reference reports.
- **DeepWeb-Bench** (Xie et al., 2026) contains 100 tasks requiring quantitative analysis across sources. Agents gather evidence and derive quantities to compare eight entities across eight analytical dimensions. An LLM judge evaluates each cell against a reference answer and a task-specific rubric, which allows partial credit and specifies when information should be reported as unavailable. We report the overall rubric score on a 0–100 scale.

A.2 Baselines

The preliminary comparison covers search-augmented, filesystem-based, and browser-based agents. The main comparison includes Fetch-then-Explore, CLI-Anything Browser, and Hybrid$_{\text{SBF}}$.

Search-augmented agents.

- **Fetch-and-Read** follows a naive search-and-read approach, fetching retrieved webpages and passing their text directly to the agent.The agent reads the returned text and searches for further evidence as needed.

- **Search-o1** (Li et al., 2025a) processes retrieved passages with its Reason-in-Documents module. Given the search query and the reasoning so far, this module extracts relevant information and returns it to the main agent for further reasoning.
- **WebThinker** (Li et al., 2025b) delegates information gathering to a Deep Web Explorer. The explorer can issue further searches and open additional pages before returning its findings to the main agent.

Filesystem-based agents.

- **Fetch-then-Explore** (Liu et al., 2026) saves fetched webpages as cleaned text in a persistent filesystem workspace. The agent uses `grep` to locate relevant passages and `read` to inspect them, and can revisit saved pages later in the task.
- **CLI-Anything Browser** is an open-source tool for shell-based web interaction. We include it to compare a browser-oriented tool with WEB$HELL's unified interface for information seeking and interaction across webpages.

Browser-based agents.

- **Agent-E** (Abuelsaad et al., 2024) separates task planning from browser execution. A planner assigns steps to a browser agent, which inspects page content and controls, performs actions, and reports the results to the planner.
- **AgentOccam** (Yang et al., 2025) presents a simplified accessibility tree with identifiers for actionable elements. The agent selects actions using the current observation and interaction history, and maintains a plan that it can branch or revise during navigation.
- **BrowserUse** (Müller & Žunič, 2024) combines structured webpage observations with browser actions in an iterative agent loop. It uses the current page and prior interaction history to select actions such as clicking, entering text, and navigating, then inspects the updated page.

Hybrid agents.

- **Hybrid$_{\mathbf{SB}}$** combines web search and browser access. It receives cleaned webpage text for information gathering and accessibility-tree observations for browser interaction.
- **Hybrid$_{\mathbf{SBF}}$** adds a persistent filesystem workspace to Hybrid$_{\mathrm{SB}}$. It stores these observations as files for selective inspection with shell operations.

A.3 Evaluation Settings

Implementation Details. We use `gpt-5.6-luna` as the main-agent model for all methods, with reasoning effort set to `high`. For established baselines, we use prompts from their original implementations, adapted to the benchmark instructions and available tools. All methods receive the same main-agent turn budget within each task category, with 100 turns for Web Search, 50 for Web Navigation, and 200 for Deep Research. We apply a 75-second page-access timeout and a 60-minute execution limit per task across methods. Methods with web-search access use Serper[†]. For recoverable browser-loading failures, we fall back to a direct HTTP request to retrieve the document. Error pages and CAPTCHA challenges are reported as access failures rather than treated as page evidence. To prevent answer leakage, we exclude benchmark pages containing reference answers, including those hosted on Hugging Face, from search results and page access.

Answer evaluation. We use `gpt-5.6-luna` for LLM-based evaluation on WideSearch, SealQA, WebWalkerQA, and DeepResearchBench, and `gpt-5.5` on DeepWeb-Bench. For SealQA and WebWalkerQA, the judge receives the question, final answer, and reference answer and checks whether the required information is correctly provided, allowing equivalent wording. Each benchmark uses the same evaluation criteria and judge model across methods.

[†] `https://serper.dev/`

B EXPERIMENTAL DETAILS

B.1 PRELIMINARY ANALYSIS

We randomly sample 100 tasks from each of WideSearch and WebArena-Verified for the preliminary analysis. Our comparison examines whether the designs developed for information seeking remain effective when tasks also require browser interaction. For WebArena-Verified, we provide search-oriented agents with the browser actions introduced in WebArena (Zhou et al., 2024), including clicking, typing, scrolling, tab management, and page navigation. We preserve their clean-text observations because these representations are integral to how the methods gather and process web evidence. Adding accessibility trees would address a different question: whether extending those methods with browser-oriented observations improves navigation. We examine this extension through $\text{Hybrid}_{\text{SB}}$ in Analysis II.

Per-call input and cost. We use per-call metrics to distinguish the context processed at each model invocation from the number of invocations needed to complete a task. This distinction matters because some baselines distribute work across a main agent and auxiliary planners or explorers. We include all such calls and divide their total input tokens and inference cost by the total number of LLM calls. We report mean input tokens per call and inference cost per 100 calls, with the latter scaled for readability. The resulting comparison captures the average inference burden across each method's model calls, including work delegated to auxiliary modules.

B.2 MAIN ANALYSIS

Evidence coverage. We use evidence coverage to assess how effectively an interface helps the agent access the information needed to answer a question. Using each benchmark's reference answers as the gold information, we check how much of that information appears in the tool responses exposed to the agent during task execution. This distinction matters for interfaces that store webpages for selective reading. Relevant information may be present in the workspace without ever reaching the agent's context. We therefore count only the content actually returned to the agent, including passages retrieved through `grep` or `read`. On WideSearch, coverage is the fraction of required reference- table cells whose values appear in this content. On SealQA, it is the fraction of questions whose reference answer appears. Each reference item is counted once, regardless of repeated exposure.

Action-reference ablation. We examine which aspects of the unified interface contribute to WEB$HELL's effectiveness. The preliminary analysis motivates reducing the need to switch representations when moving from information gathering to browser interaction. This capability is supported jointly by WEBCONTRACT, WEBCOMPILER, and WEBCACHE, which define, construct, and retain objects with their associated actions. Because these components operate on the same object schema, removing an entire component would alter several aspects of the interface together. We therefore focus on the references linking information objects to action objects, removing these references while retaining both types of objects and WEBCACHE. Without these references, the agent must request a separate control listing to obtain executable identifiers. These identifiers remain valid until the page changes or another listing is requested. We compare task performance and LLM inference cost on 100 WebArena-Verified tasks and 100 WebWalkerQA questions.

Noise analysis. We examine whether agents can locate relevant information efficiently when unrelated pages accumulate in their workspace. For each SealQA run, we place 50, 100, 200, or 400 distractor pages in WEB$HELL's initial WEBCACHE and in the filesystem workspaces of Fetch-then-Explore and $\text{Hybrid}_{\text{SBF}}$. These pages are available for inspection but are not inserted directly into the agent's context. We measure answer accuracy and LLM inference cost at each noise level. We exclude CLI-Anything Browser because it exposes webpages through open browser tabs. Introducing distractors would therefore require additional browser tabs (e.g., 400 chrome tabs), rather than stored pages in a workspace. This experiment focuses on the latter setting.

Table 4: Illustrative information and action objects from a product options table. Clicking the Medium cell selects that size. The cell's association provides the corresponding action address. Addresses are simplified.

Property	Information object	Action object
Address	`tables/options.cells:12`	`A0000`
Value	`Medium`	Click to select Medium.
Association	Column: *Size*. Related action: `A0000`.	Table: `tables/options.cells`. Row: *Medium*; column: *Size*.

Table 5: Page schema. Representative files and the information preserved in their records.

Files	Record content
`content/`, `boilerplate/`	Text sections and surrounding page content, with headings and references to associated objects.
`tables/`	Each cell stores its row identifier, row label, column label, and value. The table identifies its source.
`links.tsv`	Link identifier, destination, displayed text, and containing sections.
`images.tsv`	Image identifier, source URL, available description, and containing sections.
`actions.tsv`	Action identifier, type, label, context, input specification, and containing section.
`facts.kv`	Named values with keys identifying their source fields.
`resources.tsv`	Resource identifier, source URL, content type, and a reference to the stored response.
`long-values/`	Complete values stored separately, with their addresses and lengths retained in the parent records.

C Implementation Details

C.1 WebContract

Object schema. WebContract specifies how information and actions are addressed, interpreted, and accessed. Every object has an *address* identifying it within a captured page, a *value* containing information or an action specification, and *associations* providing context or references to related objects. These properties are encoded in text, table, and action records according to the object's type. For example, a table cell retains its row and column labels, while a selection action retains its purpose and available inputs. Table 4 shows both objects on an illustrative clothing page.

An object's full address combines its page address with a local identifier. Text and table entries use a file path and line number; links, images, resources, and actions use identifiers such as `L0000`, `I0000`, `R0000`, and `A0000`. These addresses remain valid for the stored capture even after the live page changes.

Page schema. Each compiled page includes a table of contents that lists its text sections and object groups, together with their addresses for access. Stored in `_map.md`, this table of contents provides an overview of the page layout shown in Table 5. Text sections follow the webpage's heading structure and include references to the tables and actions they contain. The agent can therefore begin with the table of contents or a search match, then follow these references to inspect related objects.

Operation schema. A request specifies an operation, its arguments, and a page or object address when needed. The response reports the outcome and returns the requested content or resulting page, together with addresses for subsequent requests. Table 6 gives the request and response for each operation. For the table cell in Table 4, `lookup` with row *Medium* and column *Chest (cm)* returns

Table 6: Requests and responses under WEBCONTRACT.

Operation	Request	Response
`retrieve`	A web search query.	Ranked results, cached page addresses, and page access outcomes.
`list`	No target required.	Cached page addresses, titles, and live-page status.
`grep`	A pattern, optionally restricted to a page or record group.	Matching content with its addresses.
`read`	A page or object address, optionally with a content range.	Requested content or page index, with a continuation position when needed.
`lookup`	A page or table address, with row and column criteria.	Matching cells with row context and associated actions.
`resolve`	A page address and an in-page reference.	The referenced content and its compiled address.
`execute`	A destination or action address, with any required input values.	The execution outcome and resulting page address.
`focus`	A page address.	The current state of the corresponding live page.

`96` with its row context. For the selection action, `execute` with page `N0000`, action `A0000`, and `size=m` selects *Medium* and returns the resulting page address.

C.2 WEBCOMPILER

Compilation procedure. HTML describes a webpage's elements and their relationships, including headings, table cells, and input controls. WEBCOMPILER traverses the captured document in order and applies a constructor for each supported element type. Constructors process nested elements recursively and preserve their relationships as associations. Elements without a dedicated constructor contribute their text and nested content to the containing section. The compiler then assigns file locations, connects references, and writes the page index without model inference.

Text and table construction. Headings divide text into sections, while formatting such as boldface remains part of the surrounding text. Navigation, headers, and footers are stored separately when the page identifies these regions. For tables, the compiler accounts for cells spanning multiple rows or columns and associates each value with its row and column labels.

Controls and forms. A control is an interactive element, such as a button, text field, or selection menu. HTML and accessibility annotations called ARIA describe its type and label. The compiler combines these declarations with input options, current values, and constraints such as required fields. In the clothing example, the `<label>` references the `<select>` menu, connecting the purpose *Clothing size* to the selection action. Its nested `<option>` elements supply the displayed names and corresponding input values. For a form, related fields and their submission control become one action.

Additional page data. A webpage may receive records that are not all displayed on screen. The compiler also extracts structured data embedded in the page and text responses captured from the same origin during page loading. It organizes these data into facts or tables, retaining references to their source and related records.

Long values. Individual fields can be much larger than the surrounding content; one vehicle-model field in a captured car-review page contains 1,767,349 characters. Table and metadata values longer than 2,000 characters are stored separately, with their addresses and lengths retained in the parent records. These large values can then be retrieved incrementally by requesting specific line or character ranges, while the complete value remains stored.

C.3 WEBCACHE

Page acquisition and reuse. WEBCACHE stores pages obtained through web search or opened directly by the agent, making them available throughout the task. For each search, the interface attempts to fetch the ranked results and returns an address for each stored page, along with any access failures. The agent can then search or read the compiled content without fetching the page again. Browser tabs may also remain open, allowing the agent to return to a page and interact with it after inspecting other pages. These tabs share the browser session's cookies and local storage, while each stored page state has its own address.

Live page states. After a control executes, the interface captures the resulting page. Navigation to another URL produces a page capture; an update at the same URL produces a state capture when the compiled content changes. For a state update, the response identifies the previous state and executed action, and lists changed, added, or removed files. An unchanged result creates no new state, and a previously reached state reuses its cached address. The action still executes even when its result is already cached.

Reading history and executing actions. As the agent navigates and interacts with a website, WEBCACHE preserves previously visited page states so that earlier information remains available for reading. The stored addresses identify this information, but accessing them does not restore the browser to a previous state. To distinguish stored records from pages that remain open, `list` marks the most recently used live page as `[active]` and other retained live pages as `[live]`. After an interaction changes a page, its tab is associated with the resulting state, and `focus` selects a retained tab and returns its current state. Action execution depends on the browser state currently available. If a page must be reopened, the interface checks its current content before attempting the requested action. If the requested control cannot be confirmed, it returns the current page information without executing the action. If the tab for a saved interaction state is no longer available, the response explains that the action cannot be executed directly from that state.

D ADDITIONAL RESULTS AND ANALYSES

Performance across domains. We examine whether the overall gains are concentrated in particular domains. As shown in Figure 9, WEB$HELL matches or exceeds $\text{Hybrid}_{\text{SBF}}$ across all categories in SealQA and WebWalkerQA, and outperforms CLI-Anything Browser in each category. The comparison with Fetch-then-Explore reveals a more even performance profile on WebWalkerQA. Although Fetch-then-Explore achieves higher accuracy on Education, its accuracy falls to 52.3% on Game, whereas WEB$HELL remains between 63.5% and 69.7% across the four categories. Together, these results suggest that WEB$HELL's effectiveness extends across different subject areas and website categories, rather than relying on exceptional performance in a narrow domain.

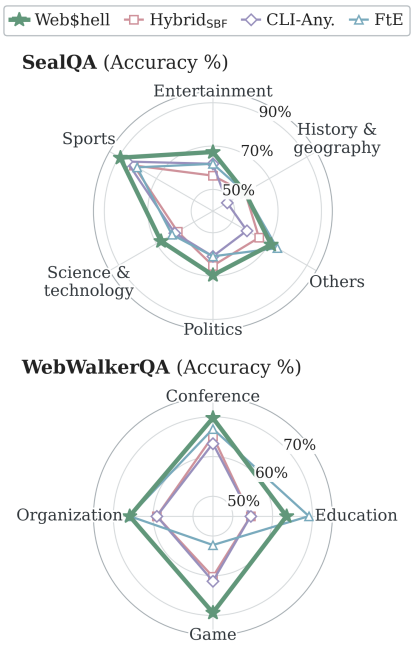

Figure 9: Domain analysis

Noise and selective access. Figure 7 shows that inference costs respond differently as irrelevant pages accumulate, even though all three methods support selective reading. The trajectories suggest that this difference partly arises from how agents manage file paths as the workspace grows. Fetch-then-Explore typically starts with a full workspace listing, whose length increases with the number of stored pages. $\text{Hybrid}_{\text{SBF}}$ often avoids this step by passing the file path returned by `fetch` directly to `grep`. As more pages accumulate, however, keeping track of the relevant paths becomes more difficult, and we observe cases where incorrectly reproduced paths lead to

full workspace listings. WEB$HELL provides short addresses for both pages and individual objects, helping agents specify subsequent searches and reads more reliably. These observations suggest that efficient selective access requires manageable references to stored content, so that locating evidence does not become increasingly costly as the workspace grows.

Scaling and evidence retention. Figure 8 examines how WEB$HELL handles a growing collection of prefetched pages. By storing pages in WEBCACHE and exposing them through selective operations, WEB$HELL makes more information available without requiring the agent to read every page. The agent inspects a similarly sized subset as the collection grows, with the fraction of cached pages inspected decreasing from 14.7% at $k = 5$ to 10.4% at $k = 10$. Yet unaccessed content can still contain useful evidence. At $k = 10$, 9.5% of the gold information in WEBCACHE has not reached the agent's observations. Retaining this content preserves the opportunity to retrieve it through later requests. This highlights the value of separating information availability from immediate consumption, while motivating cache management that preserves potentially useful evidence.

Table 7: Mean compilation time per page for WEB$HELL.

Benchmark	Time (s)
WideSearch	1.62
SealQA	2.02
WebArena-Verified	0.98
WebWalkerQA	0.46
DeepResearchBench	1.38
DeepWeb-Bench	2.79

Compilation overhead. WEB$HELL and the main baselines share the same web-search API. However, page loading time depends heavily on the websites accessed and their rendering behavior, making total elapsed time an indirect measure of interface overhead. WEB$HELL also supports fallback access when retrieval fails and closes failed pages to avoid unnecessary continued processing. We therefore measure compilation overhead directly as the time required to compile acquired content and store the resulting objects. Across benchmarks, this takes 0.46–2.79 seconds per page on average (Table 7), indicating a modest per-page compilation cost.

Cost comparison with Fetch-then-Explore. WEB$HELL achieves higher performance than Fetch-then-Explore across Web Search and Deep Research with fewer model calls and comparable inference cost per task (Table 3). To understand this balance, we examine the cumulative token usage per task (Table 8). On Web Search, WEB$HELL uses fewer uncached input and output tokens, although its cached input is higher. The input totals include instructions explaining how to use each interface and its tool definitions. Under the evaluated pricing, cached input costs 10% of regular input, keeping the total inference cost comparable despite WEB$HELL's higher total token count. Thus, WEB$HELL achieves stronger information-seeking performance within a similar inference budget, while additionally supporting browser interaction.

Table 8: Mean token usage (k) and LLM inference cost (USD) per task. Cached input refers to the model provider's prompt cache. Each task category averages its two benchmark means. Lower values in each column are bold.

Agent	Web Search				Deep Research			
	Cached	Uncached	Output	Cost	Cached	Uncached	Output	Cost
Fetch-then-Explore	**1,120.5**	110.5	10.0	0.0626	2,059.8	**120.6**	21.2	0.0969
WEB$HELL	1,410.3	**87.2**	**9.1**	**0.0623**	**1,705.9**	141.3	**20.3**	**0.0957**