MCP vs RAG vs NLWeb vs HTML: A Comparison of the Effectiveness and Efficiency of Different Agent Interfaces to the Web

Aaron Steiner
Data and Web Science Group
University of Mannheim
Mannheim, Germany
aaron.steiner@uni-mannheim.de

Ralph Peeters
Data and Web Science Group
University of Mannheim
Mannheim, Germany
ralph.peeters@uni-mannheim.de

Christian Bizer
Data and Web Science Group
University of Mannheim
Mannheim, Germany
christian.bizer@uni-mannheim.de

Abstract

LLM-based agents are increasingly used to automate web tasks such as product search, offer comparison, and order placement. Current research explores different interfaces through which these agents interact with websites, including traditional HTML browsing, retrieval-augmented generation (RAG) over pre-crawled content, communication via Web APIs using the Model Context Protocol (MCP), and natural-language querying through the NLWeb interface. Yet no systematic comparison of the effectiveness and efficiency of these interfaces on identical challenging task sets exists. To address this gap, we introduce a testbed consisting of four simulated e-shops, each offering its products via HTML, MCP, and NLWeb interfaces. For each interface (HTML, RAG, MCP, and NLWeb), we develop specialized agents that perform the same sets of tasks, ranging from simple product searches and price comparisons to complex queries for complementary or substitute products and checkout processes. We evaluate the agents using GPT-5 and GPT-5-mini. Our evaluation shows that RAG, MCP, and NLWeb agents outperform HTML browsing agents by 11 percentage points in task completion while requiring 2–5 times fewer tokens on search-oriented tasks. The GPT-5 RAG agent achieves the highest task completion rate (0.79) while maintaining moderate token consumption.

CCS Concepts

• **Computing methodologies** → **Intelligent agents**; • **Information systems** → *Web interfaces*.

Keywords

Web agents, LLM agents, RAG, MCP, NLWeb, electronic commerce

ACM Reference Format:
Aaron Steiner, Ralph Peeters, and Christian Bizer. 2026. 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), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3774904.3792893

WWW '26, Dubai, United Arab Emirates

ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792893

1 Introduction

There is currently a lot of experimentation with different interfaces that LLM-based agents [2, 4] may use to interact with websites [5, 7]. One line of work focuses on agents that interact with HTML pages by clicking on links and filling forms. Second, agents can rely on Retrieval-Augmented Generation (RAG) and interact with a search engine that has crawled and indexed relevant pages. Third, websites can expose Web APIs that agents invoke via the Model Context Protocol (MCP[1]), an open protocol that standardizes tool discovery and invocation. Fourth, websites can implement the NLWeb[2] interface, which allows agents to ask natural-language queries that are then answered by the website with JSON data using well-known vocabularies such as schema.org. Figure 1 gives an overview of the four architectures. What is still missing is a systematic comparison of the effectiveness and efficiency of these architectures using identical sets of challenging tasks.

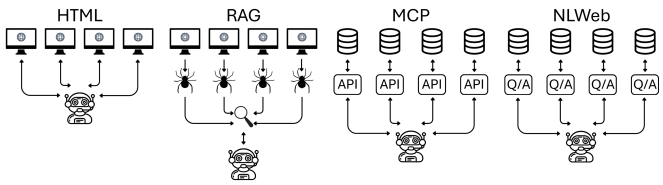

Figure 1: Overview of the four architectures.

This paper fills that gap by presenting an experimental comparison of the four architectures along an e-commerce use case that requires agents to search within several e-shops for products that fulfill specific or vague requirements, compare prices between shops, and finally order the chosen products.

This paper makes the following contributions:

(1) We introduce a testbed for comparing different agent architectures. The testbed builds on the WebMall benchmark [6]. It consists of four simulated e-shops, each offering its products via HTML, MCP, and NLWeb interfaces. For each of the four architectures (HTML, RAG, MCP, and NLWeb) the testbed contains specialized agents that interact with the e-shops using the respective interfaces.
(2) Using different sets of challenging e-commerce tasks, we systematically evaluate agent performance across interfaces and

[1] https://modelcontextprotocol.io/specification
[2] https://github.com/nlweb-ai/NLWeb

Table 1: Comparison of the four agent architectures.

Aspect	HTML	RAG	MCP	NLWeb
E-Shops				
Interface	HTML pages	Retrieval API	Proprietary APIs	Standardized API
Search functionality	Freetext search per shop	Search engine, crawled content	Per shop index; structured data	Per shop index; structured data
Search response	HTML result list including links	Pre-processed HTML pages	Heterogeneous JSON	Schema.org JSON
Agent				
Communication protocol	HTML over HTTP	Direct calls to search engine	JSON-RPC via MCP	JSON-RPC via MCP
Query strategy	Site search and browsing	Multi-query	Multi-query per shop	Multi-query per shop
Query refinement	Interactive page exploration	Self-evaluation & iteration	Self-evaluation & iteration	Self-evaluation & iteration
Add to cart / checkout	Clicking & form filling	Direct function calls	MCP tool invocation	MCP tool invocation

models (GPT-5, GPT-5-mini) and analyze the effectiveness, efficiency, and cost of the different agents.

The paper is organized as follows: Section 2 introduces the different architectures. Section 3 describes the benchmark tasks. Section 4 presents the results of the experiments. All code and data for reproducing the experiments are available in the accompanying GitHub repository[3].

2 Architectures and Interfaces

In the following, we introduce the four architectures, as well as the interfaces that agents use to interact with websites.

HTML Architecture: Within this architecture, e-shops expose traditional HTML interfaces intended for human consumption. The agents interact with these HTML pages by clicking hyperlinks and performing form-filling actions. For our experiments, we use the AX+MEM agent that was implemented as part of the WebMall benchmark [6]. The agent uses the AgentLab library and is executed within the BrowserGym framework [1]. The agent observes the accessibility tree (AXTree) of each page. It is equipped with the capability to store relevant information in a short-term memory in order to maintain context over multiple steps. We disabled visual perception for the HTML agent as adding screenshots to the prompts in addition to the AXTree decreased performance in the WebMall experiments [6].

RAG Architecture: The RAG [3] architecture includes a search engine that crawls all HTML pages from all four shops. The HTML pages are stripped of HTML markup and navigation elements, and the remaining textual content is indexed by the search engine. The RAG agent retrieves web content by querying the search engine. It does not visit the original websites. The agent iteratively formulates search queries, retrieves candidate documents, and refines subsequent queries based on intermediate results. For transactions (adding products to the cart or performing checkout), dedicated Python functions are exposed that the agent can invoke directly.

MCP Architecture: In the Model Context Protocol (MCP) architecture, e-shops expose functionalities for product search, cart manipulation, and checkout via proprietary APIs. Each shop hosts its own MCP server, defining available functions, parameter names, and JSON result formats. The MCP agent interacts with shops by calling these API endpoints and, like the RAG agent, can iteratively refine its search queries. This heterogeneity places the burden of unifying and reasoning over results on the agent, which must interpret different response structures.

NLWeb Architecture: The NLWeb interface extends the MCP protocol by requiring each website to offer a standardized, natural-language query endpoint. Each provider hosts an "ask" endpoint via MCP that accepts natural-language queries (e.g., "laptops under $1000 with 16GB RAM"). Given a query, the shop performs an internal search and returns results in JSON format using the schema.org product vocabulary. This design reduces interface heterogeneity and might make it easier for agents to understand search results due to all shops using a shared schema. Tools for cart management and checkout actions via MCP enable transactional workflows. Table 1 compares the main characteristics of the four architectures.

3 Use Case: Multi-Shop Comparison Shopping

We evaluate the four interfaces in a online shopping scenario, where a web agent must search across several shops, identify products, compare alternatives, and complete a purchase. The experiments use the product data and task sets of the WebMall benchmark [6]. WebMall is a multi-shop benchmark consisting of four stores containing a total of 4,421 product offers. The offers have been extracted from the October 2024 Common Crawl using schema.org annotations and are highly heterogeneous as they originate from a large number of real-world e-shops. The offers cover PC components, peripherals, and other electronics. The WebMall task set consists of 91 tasks that each requires agents to interact with multiple shops. The tasks are grouped into the following four categories. Table 2 shows an example task from each category.

Specific Product Search. Tests whether agents can locate all relevant offers for a specified product across multiple shops. Multiple shop may contain relevant offers, with some tasks requiring 11 offers to be found. Success depends on broad search, remembering earlier results, and merging them into a complete list.

Vague Product Search. Requires agents to find products based on vague, open-ended requirements. This includes searching the shops for substitute or compatible products given an initial product.

Cheapest Product Search. Evaluates agents on identifying the lowest-priced relevant offer. This requires agents to first retrieve all relevant offers, reason about price ordering, and finally select the cheapest option.

Transactional Tasks. Measure an agent's ability to add products to the cart and to complete checkout, which includes providing a delivery address and credit card details. Some tasks require

[3] https://github.com/wbsg-uni-mannheim/WebMall-Interfaces

Table 2: Example tasks from each category.

Task Category	Example Task
Specific Product Search (23 tasks)	Find all offers for Fractal Design PC Gaming Cases which support 240mm radiators and 330mm GPUs.
Vague Product Search (19 tasks)	Find all offers for compact keyboards that are best suited for working with a laptop remotely.
Cheapest Product Search (26 tasks)	Find the cheapest offer for a new Xbox gaming console with at least 512 GB disk space in white.
Transactional Tasks (15 tasks)	Add the product on page {url} to the shopping cart and complete the checkout process.

purchases across multiple shops, which further tests procedural coordination, for example, adding items in separate carts before initiating the respective checkout flows.

4 Results

We evaluate each agent together with two LLMs (gpt-5-2025-08-07, gpt-5-mini-2025-08-07) and compare the agent/model combinations concerning their effectiveness and efficiency.

4.1 Effectiveness

We assess effectiveness using two metrics: task completion rate (CR) and F1 score. The F1 score measures the overlap between the products returned by an agent and the ground-truth set, capturing partial correctness when outputs are incomplete or include irrelevant items. In contrast, CR captures strict success by indicating whether the returned or added product set exactly matches the ground truth. Table 3 shows the F1 results for the four agents grouped by task set and averaged across the two models. The results show that the MCP, NLWeb, and RAG agents outperform the HTML browsing agent by 11 F1 points on average accross all task categories. The average gap is widest for transactional tasks. The difference is smallest in vague product search, where all agents converge toward similar F1 scores. RAG reaches the highest F1 score on the cheapest product search tasks (Table 3, rows Cheapest Product Search). In terms of task completion averaged over all tasks, the HTML agent using GPT-5 reaches a CR of 0.69, while RAG achieves 0.79 and both MCP and NLWeb reach 0.73. These results show that API-based (MCP and NLWeb) and indexed (RAG) interfaces outperform HTML browsing.

Table 3: Average F1 performance per task set and interface. Best result per task set bold, 2nd best underlined.

Task Set / Agent	HTML	RAG	MCP	NLWeb
Specific Product Search	0.81	0.94	0.93	**0.95**
Vague Product Search	0.70	0.74	0.69	**0.75**
Cheapest Product Search	0.66	**0.77**	0.65	0.67
Transactional Tasks	0.60	0.76	**0.92**	0.86

Specific Product Search. Table 4 (rows Specific Product Search, columns CR and F1) shows that both models perform best when API (MCP or NLWeb) or RAG interfaces are used. GPT-5 reaches 0.96 F1 with RAG, MCP, and NLWeb, compared to 0.83 F1 for HTML, confirming the advantage of indexed or API-based access. GPT-5-mini follows the same trend with ~10 to ~15 F1 points higher performance than HTML. Completion rates mirror this pattern: GPT-5 achieves 0.87 CR for MCP and NLWeb versus 0.74 for HTML.

Table 4: Results averaged by task set. Best result per task set bold, 2nd best underlined.

Agent	Model	CR	F1	Tokens	Cost	Time
Specific Product Search						
HTML	GPT-5	0.74	0.83	314,231	$0.62	624 s
HTML	GPT-5-mini	0.57	0.79	233,453	$0.09	236 s
RAG	GPT-5	0.83	**0.96**	100,707	$0.18	123 s
RAG	GPT-5-mini	0.78	0.93	**32,565**	**$0.01**	**47 s**
MCP	GPT-5	**0.87**	**0.96**	134,190	$0.20	98 s
MCP	GPT-5-mini	0.74	0.90	90,948	$0.03	81 s
NLWeb	GPT-5	**0.87**	**0.96**	42,380	$0.07	62 s
NLWeb	GPT-5-mini	0.78	0.93	97,861	$0.03	106 s
Vague Product Search						
HTML	GPT-5	0.60	0.78	288,320	$0.60	641 s
HTML	GPT-5-mini	0.32	0.60	255,852	$0.09	248 s
RAG	GPT-5	**0.74**	**0.82**	111,872	$0.20	147 s
RAG	GPT-5-mini	0.58	0.66	**68,452**	**$0.02**	83 s
MCP	GPT-5	0.47	0.65	154,716	$0.23	119 s
MCP	GPT-5-mini	0.53	0.73	105,082	$0.03	101 s
NLWeb	GPT-5	0.53	0.72	77,641	$0.12	**62 s**
NLWeb	GPT-5-mini	0.63	0.77	119,435	$0.04	126 s
Cheapest Product Search						
HTML	GPT-5	**0.75**	0.75	194,594	$0.37	465 s
HTML	GPT-5-mini	0.54	0.59	209,629	$0.08	197 s
RAG	GPT-5	0.72	**0.78**	70,736	$0.14	129 s
RAG	GPT-5-mini	0.69	0.76	29,754	**$0.01**	46 s
MCP	GPT-5	0.69	0.72	94,017	$0.14	76 s
MCP	GPT-5-mini	0.54	0.57	95,723	$0.03	75 s
NLWeb	GPT-5	0.65	0.75	58,185	$0.09	60 s
NLWeb	GPT-5-mini	0.54	0.59	71,877	$0.03	93 s
Transactional Tasks						
HTML	GPT-5	0.67	0.64	182,320	$0.35	395 s
HTML	GPT-5-mini	0.53	0.56	149,165	$0.05	153 s
RAG	GPT-5	**0.93**	**0.98**	18,631	$0.04	35 s
RAG	GPT-5-mini	0.47	0.54	**12,380**	**$0.01**	**26 s**
MCP	GPT-5	**0.93**	**0.98**	98,426	$0.15	89 s
MCP	GPT-5-mini	0.73	0.88	138,755	$0.04	65 s
NLWeb	GPT-5	**0.93**	**0.98**	50,786	$0.09	49 s
NLWeb	GPT-5-mini	0.67	0.87	118,826	$0.04	82 s

Vague Product Search. Performance on Vague Product Search drops substantially compared to the Specific Product Search category, highlighting the difficulty of handling ambiguous queries as well as queries for substitute and compatible products. Table 4 (rows Vague Product Search, columns F1 and CR) shows that RAG with GPT-5 still performs best with 0.74 CR and 0.82 F1, but all interfaces exhibit lower values overall.

Cheapest Product Search. This category asks agents to identify the lowest-priced valid offer given specific or vague requirements. Introducing this requirement lowers effectiveness across all interfaces: averaged over agents and models, F1 decreases by about 0.13. The drop is most pronounced for NLWeb (~0.20), whereas HTML

and RAG retain more of their baseline performance (~0.10 reduction). Closer inspection of precision and recall reveals declines in both metrics, indicating that agents struggle with retrieval as well as selection. The performance loss, therefore, cannot be attributed solely to difficulties in re-ranking offers by price.

Transactional Tasks. This category measures the agents ability to execute transactions (adding products to cart and completing checkout). Using GPT-5, the RAG, MCP, and NLWeb agents all reach peak scores of 0.93 CR and 0.98 F1 (Table 4, row Transactional Tasks). Results for agents using GPT-5-mini are significantly lower and more mixed. The HTML agents struggle with the transactional tasks independent of the model, reaching completion rates of 0.67 and 0.53. This is due to navigational and form filling problems.

4.2 Efficiency

Table 4 also reports the token consumption, runtime, and cost (API usage fees) of the different agents averaged over all tasks within the respective task category. The costs are based on the following OpenAI pricing: GPT-5 $1.25/Mtok in, $10.00/Mtok out; GPT-5-mini $0.25/Mtok in, $2.00/Mtok out. The columns Tokens and Cost in Table 4 aggregate input and output tokens. Results split by in- and output are found in our GitHub repository.

Table 4 reports results averaged over all tasks within the respective category and split by interface and model. To make overall results comparable across interfaces, we compute micro-averages by summing token usage and runtime over all tasks for both models and then divide by two times the number of tasks. On this basis, the RAG agents on average completes a task in 83 seconds using 57.7k tokens, NLWeb agents in 81 seconds using 77.7k tokens, MCP in 87 seconds using 112k tokens. The HTML agents completes a task on average in 367 seconds using 234.8k. The results show that RAG and NLWeb are the most token- and time-efficient architectures, followed by MCP, while HTML is outperformed by a wide margin. Lower token usage directly translates to lower costs, making RAG and NLWeb not only the fastest but also the most cost-efficient architectures. On average, indexed (RAG) or API-based (MCP and NLWeb) access reduces token consumption by roughly factor three and runtime by factor four compared to HTML. In the transactional category, token differences arise not only from the add to cart and checkout flow itself but also from how each agent accesses product information, e.g. for translating URLs into product identifiers which are required by the AddtoCart functions.

Agents usually issue multiple search queries before completing a task: RAG typically submits two to six search queries per task to the search engine. The MCP and NLWeb agents execute on average four to six queries per task, but since each shop requires its own request, four of these queries are already needed to cover each shop once. As a result, RAG can further refine its search given the same number of interactions. This partly explains the high performance (0.74 CR) of the RAG/GPT-5 agent on the vague task set, while still maintaining moderate token usage. The HTML agent performs on average 23 steps per task including form filling (search box, checkout process) as well as navigation steps.

5 Related Work

In the following, we discuss related work on different architectures that LLM agents may use for interacting with websites. Song et al. [7] compare HTML browsing with direct API calling. They show that API agents reach 15% higher success rate on the WebArena benchmark [8] than HTML agents, given that good APIs are available. DeepShop [5] compares browsing-based and RAG-based agents on shopping tasks. They report a ten-point F1 improvement for RAG over browsing, while noting declines for more complex comparison tasks. Their experiments are not executed in a controlled environment, but on the live Web which limits the reproducibility of their results. These works confirm that the interface strongly influences agent success but they remain limited to comparing two architectures each. We expand this line of research by providing the first comparison of four architectures - HTML, RAG, MCP, and NLWeb - using identical tasks and a controlled environment that enables the reproduction of the results.

6 Conclusion

We introduce a testbed for comparing different interfaces that agents use to interact with websites. We use this testbed to compare four architectures: HTML browsing, an index-based RAG architecture, and two API-based architectures (MCP and NLWeb). Across models and tasks, the RAG, MCP, and NLWeb agents outperform the HTML agent by 11 F1 points on average. The gap is largest for search tasks with clearly specified requirements. All agents showed weaknesses on challenging tasks, such as vague or cheapest product searches. For transactional tasks, the API-based agents again outperformed the HTML agent. Efficiency gains are even stronger: The RAG, MCP, and NLWeb agents require two to five times fewer tokens on search-oriented tasks, resulting in substantially shorter execution times. These improvements mainly stem from reduced input tokens and the removal of browsing overhead. In summary, our results show that API-based architectures are more effective and more efficient compared to HTML-browsing. However, offering such APIs requires additional development and maintenance effort, making widespread adoption uncertain. For situations where offering APIs is not feasible, crawling web content and accessing it via RAG has proved to be an effective and efficient alternative in our experiments. On the downside, the RAG architecture requires regular re-crawling of websites in order to keep the index up to date.

References

[1] Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, et al. 2024. The BrowserGym Ecosystem for Web Agent Research. arXiv:2412.05467 [cs]
[2] Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. 2025. From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review. arXiv:2504.19678 [cs]
[3] Yunfan Gao, Yun Xiong, Xinyu Gao, et al. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs] doi:10.48550/arXiv.2312.10997
[4] Lars Krupp, Daniel Geißler, Paweł W. Woźniak, et al. 2025. Quantifying Web Agents-a Survey on Web Agent Performance and Efficiency. *OSF Preprints* (2025).
[5] Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, et al. 2025. DeepShop: A Benchmark for Deep Research Shopping Agents. arXiv preprint.
[6] Ralph Peeters, Aaron Steiner, Luca Schwarz, et al. 2025. WebMall – a Multi-Shop Benchmark for Evaluating Web Agents. arXiv:2508.13024 [cs.CL]
[7] Yueqi Song, Frank F. Xu, Shuyan Zhou, and Graham Neubig. 2025. Beyond Browsing: API-Based Web Agents. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, Vienna, Austria, 11066–11085. doi:10.18653/v1/2025.findings-acl.577
[8] Shuyan Zhou, Frank F. Xu, Hao Zhu, et al. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations (ICLR)*.