WebEvolve: Diagnosing and Improving Web Agent Adaptability under Natural Website Drift

Chenming Ge, Chengyang Shi, Yifei Xu, Yuxiang Yang, Binglin Zhong
{gecm, chengysh, xuyifei, yyxxyj, tozenith}@umich.edu

March 16, 2026

Abstract

The vast majority of existing web agent benchmarks are based on the assumption that websites are static. This means that these benchmarks do not evaluate the reliability that web agents maintain over time. In fact, website drift occurs in a number of different ways. For instance, a website's user interface may be modified, the website's DOM may be restructured, features may be added or modified, or the website's workflow may be changed. We introduce WebEvolve, a version-aware benchmark for evaluating the adaptability of web agents. WebEvolve consists of five components: a taxonomy of website drift, a mapping from website drift to agent capabilities, a version selection pipeline, a knowledge graph diff-based task update procedure, and a grounding-aware evaluation procedure. Through our empirical evaluations on Linkding, Amazon, SimpleWiki, and TikTok using Qwen3-VL-8B-Instruct, we found that failures are often concentrated in a number of different website drift categories. This demonstrates that a version-aware benchmarking approach provides a more informative point of view on the robustness and adaptability of web agents than a traditional evaluation based on overall success rates.

1 Introduction

In the domains of web navigation information, retrieval, and execution of tasks on the web, web agents have been found to hold significant potential. The existing benchmarks have played an important role in quantifying the potential of the agents. However, the existing benchmarks are based on a static web environment. In a static environment, the tasks that the agent is required to perform are designed based on a particular version of the website.

In a practical scenario, the websites are subject to evolution. Even slight changes to the user interface, the DOM structure, the features of the website, or the workflows can result in the failure of the strategies that the web agent had adopted to perform the tasks. Therefore, the existing benchmarks may not reflect the robustness of the web agents. This has a significant impact on the practicality of the existing evaluation results. This may also result in an overly positive evaluation of the potential of the web agents. Therefore, the evaluation of the robustness of the web agents is essential. This is the motivation for our research on the evaluation of the robustness of the web agents.

In this work, we propose a new framework **Webevolve** for evaluating the robustness of web agents. Our framework is based on changes across different website versions and thus provides a version-aware benchmark for robustness evaluation.

2 Proposed method

We propose **WebEvolve**, a benchmark framework for diagnosing how natural website evolution affects web-agent behavior across versions. Rather than positioning version change itself as the sole novelty, WebEvolve focuses on three underexplored aspects of web-agent evaluation: *(1)* a structured taxonomy of website drift, *(2)* an explicit mapping from drift types to agent capability dimensions, and *(3)* a maintenance-aware benchmark pipeline that identifies and updates version-affected tasks. In this way, WebEvolve treats website evolution not merely as a source of benchmark breakage, but as a measurable signal for understanding agent robustness, grounding, planning, exploration, and adaptation. Concretely, our framework consists of five components: (1) a comprehensive taxonomy of version changes, (2) a mapping from change types to interpretable agent capability dimensions, (3) a principled version selection pipeline, (4) a knowledge-graph-based method for affected-task detection and scalable task generation, and (5) a grounding-aware evaluation protocol. Figure 1 provides an overview of the full pipeline.

2.1 Problem Formulation

We model website evolution as a family of version-indexed web environments. Let $\mathcal{V} = \{v_1, \ldots, v_K\}$ denote a chronologically ordered set of website snapshots.

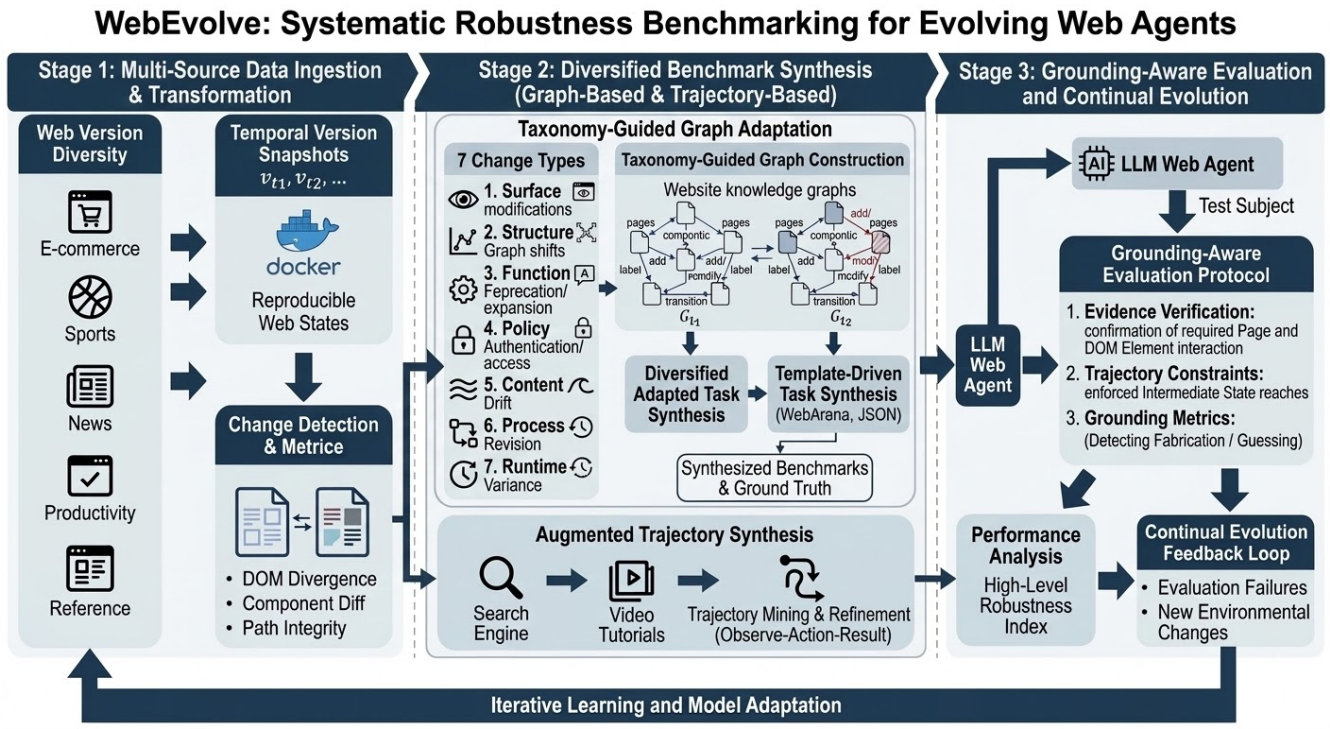

Figure 1: Overall pipeline for WebEvolve Benchmark

For each version $v \in \mathcal{V}$, the website induces a POMDP

$$\mathcal{M}^v = \langle \mathcal{S}^v, \mathcal{A}, \mathcal{O}^v, \mathcal{T}^v, \Omega^v \rangle,$$

where different versions may differ in their states, observations, and transition dynamics due to changes in layout, structure, functionality, content, or runtime behavior. Given a task instruction i, the agent interacts with version v using policy π:

$$a_t \sim \pi(i,\, o_t^v,\, a_{1:t-1},\, o_{1:t-1}), \qquad s_{t+1}^v \sim \mathcal{T}^v(s_t^v, a_t).$$

A task is defined as

$$\tau = (i,\, V_\tau,\, \mathcal{C}_\tau,\, \mathcal{D}_\tau,\, \mathcal{K}_\tau,\, r_\tau),$$

where $V_\tau \subseteq \mathcal{V}$ is the set of target versions, $\mathcal{C}_\tau$ is the set of exercised change types, $\mathcal{D}_\tau$ is the set of assessed capability dimensions, $\mathcal{K}_\tau$ denotes required key nodes or grounding evidence (1), and r_τ is the evaluation function. A task is *version-affected* between (v_j, v_k) if its execution depends on an element or transition modified in the structured diff $\Delta(v_j, v_k)$.

Evaluation combines task success with grounding verification:

$$r_\tau(v, \mathbf{a}, \mathbf{s}) = r_{\text{outcome}}(s_T, \tau) \cdot \mathbb{1}[\forall k_m \in \mathcal{K}_\tau,\ \exists t \text{ s.t. } s_t \models k_m]. \tag{1}$$

The first term checks whether the task is completed, while the second ensures that the agent actually visits the required evidence.

For each capability dimension d, we report cross-version robustness as

$$\text{Robust}_d(\pi) = \frac{1}{|\mathcal{T}_d|} \sum_{\tau \in \mathcal{T}_d} \frac{1}{|V_\tau|} \sum_{v \in V_\tau} r_\tau(v, \mathbf{a}^v, \mathbf{s}^v), \tag{2}$$

where $\mathcal{T}_d = \{\tau \mid d \in \mathcal{D}_\tau\}$. This enables fine-grained analysis of which agent capabilities degrade under website drift.

2.2 Taxonomy of Websites and Its Variations

2.2.1 Taxonomy of Web Version Changes

The foundation of our benchmark is a fine-grained taxonomy that categorizes the types of changes a website may undergo over time. We identify seven categories, organized from surface-level to system-level, shown in Table B in the Appendix B.

This taxonomy serves as the benchmark's core annotation asset: every task instance is tagged with the specific change types it exercises, enabling per-category performance analysis and fine-grained diagnosis of agent weaknesses.

2.2.2 Website Type Coverage

Different categories of websites exhibit systematically different change patterns. For example, news and entertainment sites experience rapid content drift and fre-

quent UI experiments; e-commerce platforms undergo heavy A/B testing of checkout flows and promotional overlays; productivity tools, like Google Docs, Calendar, steadily introduce new features and modify collaboration entry points; and reference sites, like Wikipedia, maintain stable structures but update content. To ensure that our benchmark is representative rather than biased toward a single change pattern, we sample target websites across multiple categories, like entertainment/news, sports, e-commerce, productivity, and reference, and verify that the resulting task set provides balanced coverage across our change taxonomy.

2.2.3 From Change Types to Capability Dimensions

To make the evaluation results interpretable and actionable, we define a set of agent capacity dimensions and establish explicit mappings from change types to these dimensions. This allows us to move beyond aggregate success rates and instead identify which specific abilities an agent lacks when it fails under version drift.

We propose five capability dimensions, which is shown in Table C in Appendix C.

Each task in our benchmark is annotated with both the change types present between the version pair and the capability dimensions being evaluated, enabling a two-dimensional analysis of agent performance.

2.3 Version Selection: Detect, Measure, Select

We monitor target websites over time by periodically crawling and recording page structure. For each crawl, we extract a structured representation of the page including the DOM skeleton, the set of visible interactive elements, and the navigation graph. Change detection is performed by comparing consecutive snapshots to identify structural modifications, element additions/removals, and link changes. **(1) Detect.** We periodically crawl target websites and extract structured page representations, like DOM skeletons, interactive-element inventories, and navigation graphs, then diff consecutive snapshots to flag changes. **(2) Measure.** We quantify pairwise divergence between snapshots using DOM/text edit distance, Jaccard distance over interactive components, and a navigation-path integrity score. **(3) Select.** We choose version pairs that maximize change-type diversity subject to a minimum divergence threshold, annotate each pair with structured change descriptions, and package the resulting snapshots as reproducible Docker images.

2.4 Automatic Task Generation

2.4.1 Version-Aware Task Generation via Knowledge Graph Differencing

Manually re-authoring tasks for every version pair is prohibitively expensive, so we adopt a scalable, semi-automated pipeline based on knowledge graph differencing. For each website snapshot v_t, we construct a typed directed graph $G_t = (V_t, E_t)$, where nodes represent semantically normalized pages and interactive elements, and edges capture executable relations such as navigation, triggering, and reveal dependencies. Given two versions G_{t_1} and G_{t_2}, we align semantically corresponding nodes across versions and compute a structured graph diff $\Delta_{t_1 \to t_2}$, with each node- or edge-level edit tagged by the corresponding change type in our taxonomy. We then model each benchmark task as a goal-conditioned execution trajectory over the website graph, and mark a task as *version-affected* whenever its reference trajectory intersects the changed region induced by $\Delta_{t_1 \to t_2}$, yielding an automatic and explainable criterion for deciding which tasks require re-evaluation. The same diff further supports task creation: for newly added, rewired, or deprecated subgraphs, we instantiate candidate tasks from JSON templates inspired by WebArena (2), use an LLM to convert graph-local changes into natural language task specifications, and retain only those validated by human annotators. This design naturally enables continual benchmark maintenance, as each newly detected website version can be compared against prior snapshots to automatically identify impacted tasks and trigger targeted task generation without full manual re-annotation.

2.4.2 Scalable Trajectory Collection via Web Tutorials

To support both training and evaluation, we collect interaction trajectories at scale by leveraging publicly available web tutorials. **(1) Harvest.** We query search engines for how-to tutorials covering common tasks on target websites. **(2) Execute.** An LLM-based agent follows each tutorial in the live or snapshotted environment, producing observation-action-result trajectories. **(3) Verify.** Rule-based checks combined with LLM-as-judge evaluation determine success; failed attempts are analyzed for common failure modes. **(4) Store.** Successful trajectories are saved as structured records. To encourage exploration, we also run a variant where the agent receives only the task goal without step-by-step instructions, producing trajectories that more closely reflect real deployment conditions.

2.5 Benchmark Design

2.5.1 Grounding-Aware Evaluation Protocol

Language models can sometimes produce correct final answers by hallucinating plausible responses rather than actually navigating to the relevant page. To address this, we augment standard outcome-based evaluation with grounding constraints: for information-seeking tasks, the agent must visit and reference a specific page or DOM element as evidence—answers without valid evidence are marked as failures regardless of correctness; we record the full execution trajectory and verify that it passes through required intermediate nodes, following the key-node concept of WebCanvas (1). This protocol evaluates whether the agent's actions are anchored in real web interactions, not just the language model's ability to produce plausible-sounding outputs.

2.5.2 Design for Website Environments

Dynamic Track is used in our benchmark for evaluation depth. The dynamic track packages complete website versions as reproducible Docker environments, allowing agents to interact freely with fully functional websites across different version snapshots. The same task is executed on multiple versions, and we compare success rates, behavioral trajectories, and failure types across versions. This track provides the most realistic assessment of version robustness, covering the full range of change types including structural shifts, workflow changes, policy gating, and runtime variance.

3 Related work

Web Evolution and Diagnostic Evaluation Traditional benchmarks predominantly focus on static snapshots or binary success rates, which often obscure the root causes of failure. Recent research has shifted toward diagnostic taxonomies: **WebSuite** (3) and **GUI-Robust** (4) disaggregate failures into specific action and UI anomalies, while **WorkArena++** (5) highlights how structural drift taxes long-horizon planning. Crucially, **WebEvo** (6) proposed a method to distinguish content drift from changes in semantic structure using history-based DOM analysis. We extend this diagnostic tradition by mapping specific categories of natural version drift, which is detected via semantic structure analysis, to orthogonal agent capabilities such as grounding and reasoning.

From Passive Maintenance to Version-Aware Benchmarking Online environments such as **WebCanvas** (1) and **Online-Mind2Web** (7) confront website evolution as a maintenance burden, typically retiring broken tasks or relying on reactive crawls. While **TimeWarp** (8) introduces containerized historical versions to study robustness, it remains a static collection of snapshots requiring human-refined execution plans. Our work departs from this paradigm by treating version drift as an active evaluation signal. Instead of retiring expired tasks, we propose a self-maintaining loop where drift is automatically quantified to identify affected tasks, transforming maintenance from a manual overhead into a systematic study of agent resilience.

Automated Benchmark Synthesis and Evolution Our approach builds on emerging techniques for scalable trajectory and task generation. **InSTA** (9) and **Go-Browse** (10) demonstrate that structured exploration and synthetic task generation can achieve internet-scale coverage. Furthermore, **WebEvolver** (11) employs co-evolving world models to enhance agent self-improvement in dynamic settings. To enable programmable environment updates, **ProEvolve** (12) introduces a graph-based framework where environment evolution is expressed as graph transformations across data, tools, and schemas. We integrate these concepts into a knowledge-graph-based (KG) maintenance pipeline: by differencing KGs across versions and applying graph transformations, we automatically detect functional regressions and regenerate task trajectories. This shifts the benchmark from a fixed dataset to a dynamic system that evolves in lockstep with the live web, supporting both reproducible evaluation and scalable data harvesting.

4 Experimental Results

We evaluate our agent across four diverse branches: Linkding, Amazon, SimpleWiki, and Social Media (TikTok) on **Qwen3-vl-8b-instruct**. These domains represent varying levels of structural and content volatility . The evaluation involves over hundreds of unique tasks across multiple chronologically ordered snapshots. Fig 2. shows the general agent performance over version-drifted domains.

4.1 Milestones and Main Results

The evaluation results reveal that failure modes are highly concentrated within specific clusters rather than being randomly distributed. For feat-benchmark-share, errors are localized in the F6, F7, and F8 versions of Linkding, while amazon_mirror failures are strictly tied to the workflow_change category. Similarly, simplewiki issues are confined to version 3.2.0, and tiktok_mirror shows a clear bottleneck in the v2025 search button selector and the v2023 environment within the "Hard" suite.

Technically, these branches share a unified architecture where Playwright drives the page interactions and

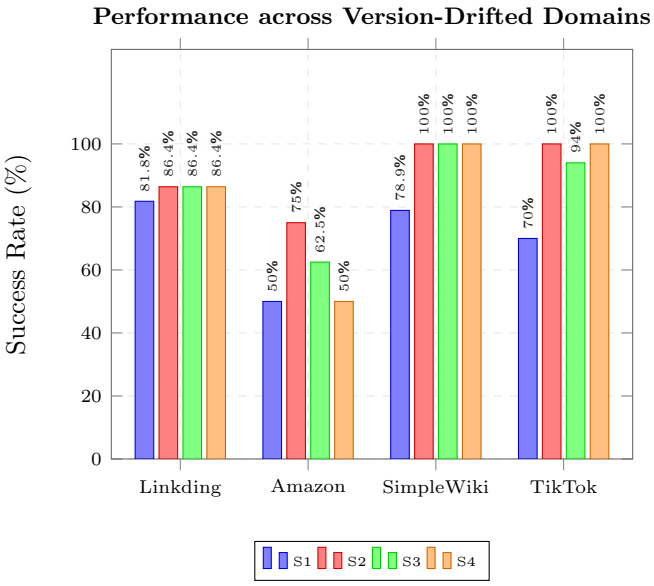

Figure 2: Comparative analysis of agent Success Rate (SR) across four domains. Each group represents a chronological evolution of the website environment. **Linkding**: S1 (1.4.0), S2 (1.14.0), S3 (1.25.0), S4 (1.45.0). **Amazon**: S1 (v2023), S2 (v2024), S3 (v2025), S4 (v2026). **SimpleWiki**: S1 (v3.2.0), S2 (v3.4.0), S3 (v3.6.0), S4 (3.8.0). **TikTok**: S1 (v2023), S2 (v2024), S3 (v2025), S4 (v2026. Performance dips in Amazon v2026 and SimpleWiki v3.2.0 highlight vulnerabilities to structural and functional drift.

OpenAI-compatible vision or text models generate the actions. The system maintains high observability by saving result.json and trace.jsonl files for every task, capturing actionable screenshots at each step, and performing automated data synthesis across category, family, and version dimensions. Table 4.1. summarizes the milestones achieved across our benchmark branches.

4.1.1 Linkding

The Linkding branch achieved an overall success rate of 85.2%, with consistent performance across versions 1.14.0, 1.25.0, and 1.45.0 (86.4%), while version 1.4.0 lagged slightly at 81.8% due to unique failures in the F15_bookmarklet_section. Systemic failures were highly localized, as all four versions failed completely in the 'F6 consistency', 'F7 tag filter', and 'F8 search filter' categories.

Primary failure modes were driven by two behavioral patterns: stagnant wait loops and action execution cycles. In the majority of cases (e.g., Task 6005), the agent failed to adapt when search queries returned null results, repeatedly triggering wait() commands until a 'stuck wait loop' termination occurred. A secondary issue (e.g., Task 6029) involved the agent entering a redundant press('Enter') cycle while attempting to verify content that was not present on the current UI state, eventually triggering loop detection after multiple stagnant actions.

4.1.2 Amazon

The Amazon branch exhibited a performance split between the small-scale robustness evaluation (59.4%) and the more stable GUI suite (96.3%). Within the small-scale run, success rates varied by version, with v2024 performing best at 75.0%, followed by v2025 at 62.5%, and both v2023 and v2026 at 50.0%. In the GUI suite, all 11 failures were exclusively concentrated in the 'workflow change' category, while other categories such as 'button position change' and 'ab testing ui' achieved perfect success rates.

The primary failure modes involved execution timeouts and contract mismatches. In cases like 'workflow change 02', the agent reached the 'max steps reached' limit after repeatedly clicking the same product without advancing the page state or triggering a recovery. Conversely, in 'workflow change 14', the task failed despite the agent successfully navigating to the shopping cart; this was attributed to a 'stop without state success' error caused by a discrepancy between the benchmark's required "basket" URL string and the mock site's actual cart routing.

4.1.3 SimpleWiki

The SimpleWiki branch demonstrated exceptional stability across modern versions, achieving a 100% success rate for versions 3.4.0, 3.6.0, and 3.8.1. Performance dips were isolated exclusively to version 3.2.0, which yielded a success rate of 78.9% (30/38) and established a 'robust sr' of 78.9%. The eight failures were primarily categorized by failed semantic validations, including 'title match', 'url includes', and 'string match'.

Unlike other branches, these failures were not characterized by visible action loops but rather by a discrepancy between the legacy Kiwix 3.2.0 routing structure and the agent's navigation assumptions. For instance, in task 9505, the agent failed to execute the transition from "Earth" to "Moon," remaining on the original page and resulting in failed checks for title and URL matches. Consequently, while the 'runtime error' logs remained empty, the 'failed checks' field confirmed that the agent failed to adapt to the older version's DOM and navigation logic.

4.1.4 Tiktok

The Tiktok branch maintained high performance with a 96.7% success rate in the Medium suite and 91.0% in the Hard suite. Failures in the Medium suite were exclusively restricted to the v2025 environment due to 'click selector fallback failed' errors across several categories. In the Hard suite, failures were more frequent in v2023

Domain/Branch	Tasks	Success Rate	Version Stability	Primary Failure
Linkding	88	85.2%	High	Wait loops
Amazon	332	96.3% (GUI)	Moderate	Workflow mismatch
SimpleWiki	152	78.9%–100%	Low (v3.2.0)	DOM-level drift
TikTok	500	91.0% (Hard)	Moderate	Feature change

Table 1: Benchmark Performance Summary across Evaluated Branches

(15 cases), primarily manifesting as 'max steps reached', while v2025 accounted for the remaining 3 failures.

Two primary technical bottlenecks were identified. First, the v2025 environment suffers from a persistent selector mismatch regarding the "Magnify" search button; although the button is visually present, the agent's fallback CSS and ARIA strategies fail to interface with the mock site's DOM structure, leading to repeated failed click attempts in both suites. Second, the v2023 Hard tasks often fall into a "search-regression" loop, where the agent successfully executes searches but inexplicably returns to the homepage (goto('/')) instead of progressing to the state-check requirement, eventually exhausting the 40-step limit.

4.2 Capability Dimension Analysis

Our evaluation confirms that agent performance is intrinsically tied to the five predefined capability dimensions (Table C). **Robustness** and **Grounding** remain the primary bottlenecks, as evidenced by failures in Linkding and TikTok due to DOM-level drift and selector mismatches. The Amazon results highlight challenges in **Planning**, where workflow shifts led to max-step timeouts despite successful individual actions. Conversely, the high success rates in modern SimpleWiki versions demonstrate strong **Exploration** and **Adaptation** capabilities when structural volatility is low. Overall, these results validate that our benchmark effectively stress-tests the agent's ability to navigate content drift and functional-layer changes across diverse domains.

5 Future Milestones

In the next six weeks, we plan to gradually turn WebEvolve from an initial prototype into a more complete and reliable benchmark:

- **Week 1 (March 16 - 22):** Polish the benchmark to customize more sophisticated tasks, specifically in a small range of websites.
- **Week 2 (March 23 - 29):** Make the version selection process more systematic and easier to explain.
- **Week 3 (March 30 - April 5):** Expand the benchmark to cover more websites and a wider range of change types.
- **Week 4 (April 6 - 12):** Optinal: Find the benchmark for redirecting websites.
- **Week 5 (April 13 - 19):** Run more experiments with different agents and settings, and analyze the results from the perspective of change types and capability dimensions. Prepare for the final presentation.
- **Week 6 (April 20 - 26):** Improve the benchmark's reproducibility and maintainability, and polish the final paper draft.
- **Week 7 (April 27 - 30):** Polish and submit the final report.

6 Conclusion

In this project, we use **Qwen3-vl-8b-instruct** as the evaluation object. During the progress, we preliminarily construct 4 evaluation tasks for 4 different platforms: Linkding, Amazon, SimpleWiki, Tiktok. For SimpleWiki and Linkding, we use 4 different versions of real website and only consider the success rate as the evaluation of the AI agent. And for the Amazon and Tiktok, we made two mirror websites and test the AI agent in five different website categories of change:A/B testing UI, Button Position Change, Feature Change, Layout adjustment and workflow. And it is found that qwen3-vl-8b-instruct is higher stable in Linkding and lower stable in SimpleWiki. And we found that the model in mirror Amazon has low robustness in Workflow Change and Tiktok has low robustness in Feature Change.

What's more, one problem in our project is that most of the benchmarks we designed are simple and most of the result remains very high. So it seems to be difficult to accurately tell the web agent's robustness of different change in different version of website. And for all the benchmarks now only focus on manipulation in only one website.

In the future plan, we want to design more complex benchmark and make the test to determine the web agent's ability and robustness of redirecting to various websites and manipulating in various websites. Then we aim to extract the features from the optimized benchmark for different website. Last we will use the features to construct a dynamic evaluation benchmark generator for different platforms.

7 Author Contributions

Chenming Ge: Responsible for reviewing the literature, exploring the algorithm for knowledge graph and trajectory synthesis, and authoring the Proposed Method and Literature Review part of the report.

Chengyang Shi: Responsible for developing the project's main framework, building benchmarks for Linkding and SimpleWiki(covering website deployment, evaluation setup, and task design), and authoring the Introduction section of the report.

Yifei Xu: Responsible for summarizing the results of the first stage experiment and organizing the main results. And adjust the report structure.

Yuxiang Yang:

Responsible for developing the project's main framework, building benchmarks for Amazon and Tiktok(covering website deployment, evaluation setup, and task design), and authoring the conclusion section of the report.

Binglin Zhong: Responsible for gaining overview of the project and proposing future milestones that need to be accomplished weekly. Also participated in manufacturing and polishing the abstract.

References

[1] Y. Pan, D. Kong, S. Zhou, C. Cui, Y. Leng, B. Jiang, H. Liu, Y. Shang, S. Zhou, T. Wu, *et al.*, "Webcanvas: Benchmarking web agents in online environments," *arXiv preprint arXiv:2406.12373*, 2024.

[2] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, *et al.*, "Webarena: A realistic web environment for building autonomous agents," *arXiv preprint arXiv:2307.13854*, 2023.

[3] E. Li and J. Waldo, "Websuite: Systematically evaluating why web agents fail," *arXiv preprint arXiv:2406.01623*, 2024.

[4] J. Yang, Z. Song, J. Chen, M. Song, S. Zhou, X. Ouyang, C. Chen, C. Wang, *et al.*, "Gui-robust: A comprehensive dataset for testing gui agent robustness in real-world anomalies," *arXiv preprint arXiv:2506.14477*, 2025.

[5] A. Drouin, M. Gasse, M. Caccia, I. H. Laradji, M. Del Verme, T. Marty, L. Boisvert, M. Thakkar, Q. Cappart, D. Vazquez, *et al.*, "Workarena: How capable are web agents at solving common knowledge work tasks?," *arXiv preprint arXiv:2403.07718*, 2024.

[6] F. Shao, R. Xu, W. Haque, J. Xu, Y. Zhang, W. Yang, Y. Ye, and X. Xiao, "Webevo: taming web application evolution via detecting semantic structure changes," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 16–28, 2021.

[7] T. Xue, W. Qi, T. Shi, C. H. Song, B. Gou, D. Song, H. Sun, and Y. Su, "An illusion of progress? assessing the current state of web agents," *arXiv preprint arXiv:2504.01382*, 2025.

[8] M. F. Ishmam and K. Marino, "Timewarp: Evaluating web agents by revisiting the past," *arXiv preprint arXiv:2603.04949*, 2026.

[9] B. Trabucco, G. Sigurdsson, R. Piramuthu, and R. Salakhutdinov, "Insta: Towards internet-scale training for agents," *arXiv preprint arXiv:2502.06776*, 2025.

[10] A. Gandhi and G. Neubig, "Go-browse: Training web agents with structured exploration," *arXiv preprint arXiv:2506.03533*, 2025.

[11] T. Fang, H. Zhang, Z. Zhang, K. Ma, W. Yu, H. Mi, and D. Yu, "Webevolver: Enhancing web agent self-improvement with co-evolving world model," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8970–8986, 2025.

[12] G. Li, Y. Xie, Y. Liu, Z. Dong, X. Pan, T. Zheng, J. Choi, M. J. Morais, B. Jha, S. Mishra, *et al.*, "The world won't stay still: Programmable evolution for agent benchmarks," *arXiv preprint arXiv:2603.05910*, 2026.

[13] S. Yao, H. Chen, J. Yang, and K. Narasimhan, "Webshop: Towards scalable real-world web interaction with grounded language agents," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20744–20757, 2022.

[14] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2web: Towards a generalist agent for the web," *Advances in Neural Information Processing Systems*, vol. 36, pp. 28091–28114, 2023.

[15] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. Lim, P.-Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, and D. Fried, "Visualwebarena: Evaluating multimodal agents on realistic visual web tasks," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

[16] G. Mialon, C. Fourrier, T. Wolf, Y. LeCun, and T. Scialom, "Gaia: a benchmark for general ai assistants," in *The Twelfth International Conference on Learning Representations*, 2023.

[17] B. Gou, Z. Huang, Y. Ning, Y. Gu, M. Lin, W. Qi, A. Kopanev, B. Yu, B. J. Gutiérrez, Y. Shu, *et al.*, "Mind2web 2: Evaluating agentic search with agent-as-a-judge," *arXiv preprint arXiv:2506.21506*, 2025.

A Comparison of different web agent benchmarks

Benchmark	# Tasks	Dynamic Env.	Online	Reproducible	Maintained	Intermediate Eval.	Version-Aware	Eval Method
WebShop (13)	12k	✓	✗	✓	✗	✗	✗	Reward function
Mind2Web (14)	2,350	✗	✗	✓	✗	✗	✗	Action matching
WebArena (2)	812	✓	✗	✓	✗	✗	✗	Functional correctness
VisualWebArena (15)	910	✓	✗	✓	✗	✗	✗	Functional correctness
GAIA (16)	466	✓	✓	✗	✗	✗	✗	Answer matching
WebCanvas (1)	542	✓	✓	✗	✓	✓	✗	Key nodes
Online-Mind2Web (7)	300	✓	✓	✗	✓	✗	✗	WebJudge (LLM)
Mind2Web 2 (17)	130	✓	✓	✗	✓	✓	✗	Agent-as-a-Judge
Ours	TBD	✓	✗	✓	✓	✓	✓	Multi-version diff

Table 2: Comparison of web agent benchmarks. **Dynamic Env.**: whether agents interact with a live or functional environment rather than static snapshots. **Online**: whether evaluation runs on real live websites. **Reproducible**: whether the environment can be exactly replicated across runs. **Maintained**: whether the benchmark is actively updated to reflect website changes. **Intermediate Eval.**: whether evaluation captures intermediate states, not just final outcomes. **Version-Aware**: whether the benchmark explicitly tests agent robustness across different website versions. "—" indicates the property is not applicable.

B Detailed description of the layers of web environment change

Layer	Scope	Description
Surface	Visual presentation	Modifications to text labels, color schemes, layout adjustments, or button repositioning. These changes alter the appearance but not the underlying functionality.
Structural	DOM & information architecture	Changes to the DOM hierarchy, information architecture, or navigation entry points. For example, a settings page previously accessible from a sidebar may be relocated under a dropdown menu.
Functional	Feature addition / removal	Addition or removal of features, including new interactive workflows, deprecated functionality, or newly introduced entry points for existing features.
Access	Access control mechanisms	Introduction or modification of access control mechanisms such as login walls, geographic restrictions, A/B testing gates, or cookie consent dialogs that alter the conditions under which content or functionality is accessible.
Content	Dynamic content	Changes to dynamic content such as news feeds, recommendation streams, sports schedules, or leaderboards. These changes make grounding particularly challenging, as the specific elements an agent must interact with are no longer static.
Process	Multi-step workflows	Modifications to multi-step processes, including changes in the number of required steps, mandatory fields, or the paths available for exporting, sharing, or completing transactions.
Runtime	System behavior & timing	Changes in system behavior such as loading latency, asynchronous rendering timing, or failure-retry mechanisms that affect the agent's ability to reliably observe and interact with page elements.

Table 3: Detailed description of the seven layers of web environment change.

C Detailed description of agent capability dimensions

Dimension	Primary Driver	Description
Robustness	UI & structure shifts	The ability to complete a previously solvable task when the UI layout or DOM structure has changed, without requiring any new knowledge about the website.
Exploration	Structure, function & policy shifts	The ability to discover alternative navigation paths when a previously known entry point is no longer available.
Planning	Workflow shifts	The ability to decompose tasks into appropriate sub-goals and execute multi-step strategies, spanning both *short-horizon planning* (1–3 step local decisions) and *long-horizon planning* (cross-page, multi-constraint strategy execution).
Grounding	Content drift	The ability to correctly identify and interact with the specific web elements required by a task, rather than relying on superficial heuristics or hallucinated information. Content drift poses the greatest challenge, as target elements change across versions.
Adaptation	Function-layer changes	The ability to discover and leverage newly introduced functionality that may offer more efficient or previously unavailable paths to task completion.

Table 4: Detailed description of agent capability dimensions.