Loop Detection with Reflection Prompts as the Primary Driver of Web Agent Performance: An Ablation Study on Adaptive Feedback-Driven DOM Pruning

Monish Soundar Raj Jefrey Bergl Rishabhdev Potti
Department of Computer Science
University of North Carolina at Chapel Hill

April 2026

Abstract

LLM-based web agents face a persistent challenge: modern web pages produce DOM structures ranging from 10,000 to 100,000 tokens, overwhelming context windows and degrading task performance. We present AFDP (Adaptive Feedback-Driven DOM Pruning), a system that combines dynamic observation pruning, adaptive feedback, and loop detection with reflection-based recovery. Through a four-condition ablation study on 168 tasks from the WebArena-Verified Hard subset, we identify a surprising finding: loop detection with reflection prompts—not pruning—is the primary driver of improvement, yielding a 14.9 percentage point increase in task success rate over a baseline agent. Conservative pruning provides a 52% reduction in per-step observation tokens but has negligible effect on success rate (-1.3pp). Analysis reveals that DOM elements critical for navigation do not correlate with task-intent keywords, limiting the effectiveness of content-based pruning. We argue that web agent optimization should prioritize failure recovery over observation reduction, and propose cost-per-completed-task as the appropriate evaluation metric.

1 Introduction

The deployment of large language models (LLMs) as autonomous web agents has attracted significant research interest [Zhou et al., 2024, Deng et al., 2023]. These agents observe web page structure, reason about task objectives, and execute actions such as clicking, typing, and navigating. However, a fundamental bottleneck persists: the Document Object Model (DOM) of modern web pages routinely produces token counts that exceed the practical limits of current LLMs.

Recent work has addressed this through observation pruning. Prune4Web [Zhang et al., 2025] employs programmatic scoring scripts to achieve 25–50$\times$ element reduction. FocusAgent [Kerboua and Omidi Shayegan, 2025] uses a lightweight retriever to reduce token counts by over 50%. Both approaches apply static, one-shot pruning strategies that cannot adapt to execution outcomes or recover from common failure modes such as infinite action loops.

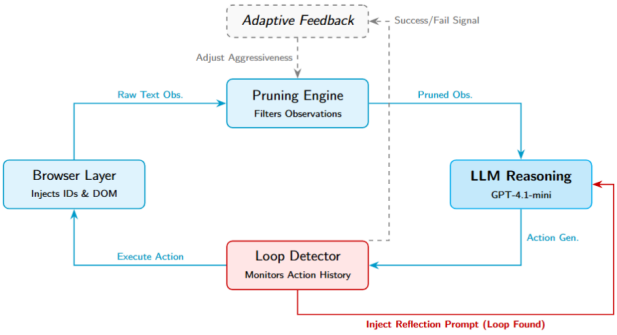

Figure 1: AFDP system architecture. The browser interaction layer injects element IDs and constructs text observations. The pruning engine filters observations before LLM inference. The loop detector monitors action history and injects reflection prompts upon detecting non-productive cycles. Adaptive feedback adjusts pruning aggressiveness based on task outcomes.

We present AFDP (Adaptive Feedback-Driven DOM Pruning), a system that extends standard web agent architectures with three components: (1) dynamic observation pruning with adaptive aggressiveness, (2) a feedback loop where task success or failure adjusts pruning parameters, and (3) loop detection with reflection-based recovery that injects re-planning prompts when the agent enters non-productive action cycles.

Our central contribution is empirical. Through a four-condition ablation study on 168 tasks from the WebArena-Verified Hard subset [El hattami et al., 2025], we demonstrate that:

- Loop detection with reflection prompts improves success rate by 14.9 percentage points (pp) over a baseline agent, from 44.6% to 59.5%.
- Conservative pruning reduces per-step observation tokens by 52% but has negligible effect on success rate (-1.3pp).
- The combined system achieves $+10.2$pp, underperforming loop-only detection because pruning occasionally removes navigation-critical elements.
- Cost-per-completed-task, rather than raw token efficiency, is the appropriate metric for evaluating web agent systems.

2 Related Work

Observation management for web agents. The challenge of large DOM structures has been addressed through several approaches. Prune4Web [Zhang et al., 2025] generates Python scoring scripts that evaluate element relevance, achieving substantial element reduction on the WebArena benchmark. FocusAgent [Kerboua and Omidi Shayegan, 2025] trains a lightweight retriever to select relevant elements from accessibility trees, reducing input tokens by over 50% while maintaining task performance. Both methods apply fixed pruning policies that do not adapt based on execution feedback. AgentOccam [Yang et al., 2025] demonstrates that simplified observation spaces can match complex agent architectures, suggesting that observation management is a critical design dimension.

Failure recovery in LLM agents. The problem of non-productive action loops in LLM agents is well-documented but rarely addressed directly. Most web agent frameworks [Zhou et al., 2024, Yao et al., 2023] implement loop detection as a termination condition: if the agent repeats the same action k times, the task is marked as failed. Reflexion [Shinn et al., 2023] introduced verbal self-reflection for task-level retry, but operates at the episode level rather than within a single task execution. To our knowledge, no prior work has measured the effect of injecting step-level reflection prompts upon loop detection in web agents.

WebArena benchmark. WebArena [Zhou et al., 2024] provides a realistic web environment with four self-hosted sites (shopping, admin, forum, code repository) and 812 verified tasks with deterministic evaluators. WebArena-Verified [El hattami et al., 2025] extends this with version-controlled environments, optimized Docker images, and a curated Hard subset of 258 tasks. We evaluate on this Hard subset to maximize the discriminative power of our ablation.

3 Method

3.1 Browser Interaction Layer

Both baseline and experimental agents share a common browser interaction layer implemented via Playwright [Microsoft, 2024]. A JavaScript function injects unique `data-wid` attributes into all visible interactive elements (links, buttons, inputs, headings, table cells, list items, and elements with ARIA roles), skipping hidden elements with zero dimensions. A second function traverses annotated elements and constructs a flat text observation where each line follows the format `[id] <tag attributes> visible_text`. This representation enables the LLM to reference specific elements by ID when generating actions.

The action executor parses LLM outputs in the format `ACTION: type [id] text` and dispatches Playwright locator calls targeting `[data-wid="N"]`. Supported actions include `click`, `type`, `scroll`, `goto`, and `stop`. Failed actions are logged but do not terminate the task.

3.2 Baseline Agent

The baseline agent operates in a standard observe-reason-act loop. At each step, it obtains the full page observation, truncates to 12,000 tokens if necessary by dropping lines from the end, and queries GPT-4.1-mini via Azure OpenAI. The system prompt includes action format instructions and pre-configured login credentials for all four WebArena sites. If the same action is produced three consecutive times, the task is terminated as a detected loop.

3.3 AFDP Agent

The AFDP agent extends the baseline with three components:

Pruning engine. The pruner operates on the text-based observation, applying a cascade of rules: (1) remove elements with non-functional tags (`script`, `style`, `svg`, `meta`, etc.); (2) always retain interactive elements (`a`, `button`, `input`, `select`, headings, table cells); (3) conditionally retain generic containers (`div`, `span`, `p`) based on text content length and, at higher aggressiveness levels, keyword overlap with the task intent. An embedding-based semantic scorer using `text-embedding-3-small` was implemented to score element-task similarity via cosine distance, but was configured to activate only at aggressiveness levels above 0.6 and did not fire in our experiments (Section 5).

Loop detector. The detector monitors action history using two strategies: (1) same-action repetition (three identical consecutive actions), and (2) action-URL pair cycling (the last N state-action pairs matching the preceding N). Upon detection, rather than terminating, the agent injects a reflection prompt instructing the LLM to attempt a fundamentally different approach.

Adaptive feedback. After each task, the pruner receives a binary success/failure signal. On success, the aggressiveness parameter increases by 0.05 (max 0.9); on failure, it decreases by 0.10 (min 0.2). In the final version, per-site policies maintain separate aggressiveness values for each domain.

3.4 Ablation Design

To isolate component contributions, we evaluate four conditions by toggling the pruning and loop recovery flags:

- **Baseline**: No pruning, no loop recovery. Terminates on loop detection.
- **Pruning-only**: Pruning enabled, loop recovery disabled.
- **Loop-only**: No pruning, loop recovery enabled.
- **Full AFDP**: Both pruning and loop recovery enabled.

4 Experimental Setup

Benchmark. We evaluate on the WebArena-Verified Hard subset [El hattami et al., 2025], selecting 168 tasks through stratified sampling (seed 42) from 233 eligible tasks across four domains: Shopping, Shopping Admin, Reddit, and GitLab (42 tasks per domain). Tasks requiring Wikipedia or Map were excluded due to infrastructure constraints.

Infrastructure. All experiments ran on an Azure Standard_D8as_v5 VM (8 vCPUs, 32GB RAM, 512GB SSD). The four WebArena sites were deployed as Docker containers using the optimized `am1n3e/webarena-verified-*` images.

LLM configuration. All conditions use GPT-4.1-mini via Azure OpenAI with temperature 0.0, max output tokens 200, and a maximum of 20 steps per task. System prompts are identical across conditions except that AFDP prompts note the observation has been pruned.

Metrics. We report task success rate (percentage of tasks where the agent issued a `stop` action), average steps per task, per-step observation tokens (before and after pruning), total input tokens, and cost-per-successful-task.

5 Results

5.1 Loop Detection Is the Primary Contributor

Table 1 presents the main ablation results. Loop-only detection achieves the highest success rate at 59.5%, a 14.9pp improvement over the 44.6% baseline. This corresponds to 25 additional task completions (100 vs. 75 out of 168). The full AFDP system achieves 54.8% (+10.2pp), while pruning-only is statistically indistinguishable from baseline at 43.3% (−1.3pp).

The individual component contributions sum to $14.9 + (-1.3) = 13.6$pp, but the combined system yields only +10.2pp, suggesting mild interference: pruning occasionally removes elements that would have enabled an alternative navigation path after loop recovery.

5.2 Per-Site Analysis

Table 2 presents per-site success rates. Shopping Admin exhibits the largest improvement under loop detection, increasing from 24% to 57%—a 2.4× improvement. This is attributable to the complex nested menu structures in administrative dashboards, where agents frequently enter loops by clicking incorrect navigation items. Loop recovery enables the agent to explore alternative menu paths.

Shopping tasks, already at 88% baseline, see a modest gain to 93% with loop detection. However, full AFDP reduces Shopping performance to 81%, indicating that pruning removes product listing elements necessary for task completion. GitLab improves from 40% to 55%, reflecting the benefits of recovery when navigating repository interfaces. Reddit, the most challenging domain, shows a consistent but modest improvement from 26% to 33%.

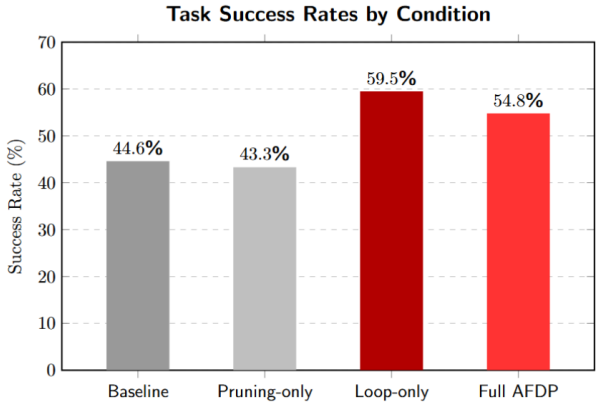

Figure 2: Task success rates across ablation conditions. Loop-only detection achieves the highest rate at 59.5%, outperforming the combined system (54.8%) and baseline (44.6%).

Loop-only detection achieves the highest or tied success rate on every site, confirming it as a domain-general improvement mechanism.

5.3 Why Pruning Is Neutral

Conservative pruning reduces raw observation tokens from 8,416 to 4,038 per step on average (52% reduction), yet achieves a negligible −1.3pp change in success rate. Analysis of failure cases reveals a structural limitation: DOM elements required for task completion fall into two categories. *Task-relevant* elements (data tables, answer text) contain keywords that match the task intent and survive pruning. *Navigation-essential* elements (menus, settings links, breadcrumbs) do not contain task-related words but are required to reach the page where the answer resides. Even conservative pruning occasionally removes these elements.

This finding has implications for the broader observation management literature: content-based relevance scoring, whether keyword-based or embedding-based, is insufficient for web agent pruning because it cannot distinguish navigation affordances from irrelevant noise.

5.4 Adaptive Feedback

The per-site feedback mechanism converged to differentiated policies: Shopping settled at aggressiveness 0.50 (6 successes, 1 failure), while Shopping Admin, GitLab, and Reddit all converged to the minimum of 0.20. This correctly identifies Shopping as having cleaner DOM structure amenable to pruning, while the other sites require more elements for navigation. The embedding-based semantic scorer was implemented with a conservative activation threshold (aggressiveness > 0.6). Since no site reached this level, semantic pruning did not activate during experiments and its effectiveness remains untested.

Table 1: Ablation results on 168 WebArena-Verified Hard tasks. Pruning-only was evaluated on a 30-task pilot subset. Loop-only achieves the highest success rate, outperforming the combined system.

Metric	Baseline	Pruning-only	Loop-only	Full AFDP
Tasks evaluated	168	30	168	168
Success rate (%)	44.6	43.3	**59.5**	54.8
Success count	75	13	**100**	92
Δ vs. baseline (pp)	—	-1.3	$+14.9$	$+10.2$
Avg. steps	8.5	9.4	11.5	12.3
Avg. obs tokens/step	4,454	3,475	4,588	4,038
Token reduction (%)	—	11.5	6.7	17.0
Total tokens (M)	35.1	5.7	68.9	66.8
Cost/success (K tokens)	468	—	689	726

Table 2: Per-site success rates. Shopping Admin shows the largest improvement with loop detection (24% $\rightarrow$ 57%).

Site	Baseline	Loop	Full
Shopping	88%	93%	81%
Shopping Admin	24%	**57%**	55%
GitLab	40%	55%	50%
Reddit	26%	33%	33%

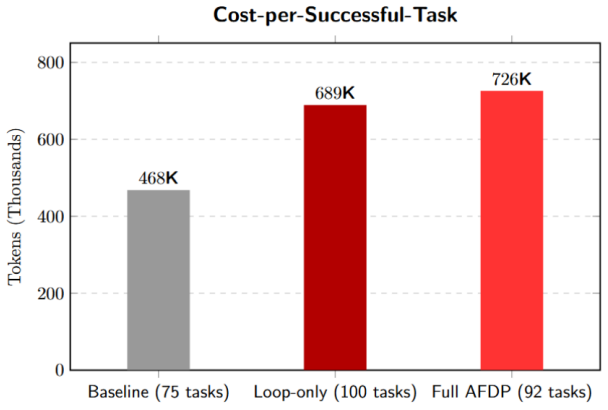

Figure 3: Cost-per-successful-task across conditions. Loop-only uses more tokens per success but completes 25 additional tasks that baseline cannot solve at any cost.

5.5 Cost-per-Completed-Task

Loop-only detection consumes 689K tokens per successful task compared to 468K for baseline. However, it completes 100 tasks versus 75—the 25 additional completions represent tasks that baseline cannot solve regardless of token budget. We argue that cost-per-completed-task is the appropriate evaluation metric for web agent systems, as it accounts for both the cost of succeeding and the cost of failing. Raw token efficiency is misleading when a cheaper system simply fails on harder tasks.

6 Discussion

Our results challenge a prevailing assumption in the web agent literature: that observation reduction is the primary lever for improving agent performance. While pruning provides meaningful token savings (52% per-step reduction), these savings do not translate to higher success rates. Loop detection with reflection prompts, a mechanism that most frameworks implement only as a termination condition, yields a 14.9pp improvement when repurposed for recovery.

This finding aligns with insights from Reflexion [Shinn et al., 2023], which demonstrated that verbal self-reflection improves LLM task performance. However, Reflexion operates at the episode level (retry the entire task), while our approach operates at the step level (recover within a single execution). The step-level granularity is critical for web agents, where partial progress (e.g., successful login) should be preserved.

The interference between pruning and loop recovery ($+10.2$pp combined vs. $+14.9$pp loop-only) suggests that these components interact non-additively. Pruning reduces the action space available for recovery: when the agent needs to find an alternative path after a loop, pruned elements may have included the necessary navigation affordances. This interaction effect merits further investigation.

7 Limitations

Several limitations constrain the generalizability of our findings. All experiments use a single LLM (GPT-4.1-mini); the effect of loop recovery may vary across model families and sizes. The pruning-only condition was evaluated on a 30-task pilot rather than the full 168-task set, yielding wider confidence intervals. We use agent-reported `stop` actions as a proxy for success rather than the WebArena-Verified functional evaluators, which check backend state for ground-truth correctness; some reported successes may be false positives. The embedding-based semantic scorer, while implemented, did not activate during experiments and remains untested. Finally, two of WebArena's six domains (Wikipedia, Map) were excluded due to infrastructure and budget constraints (˜$200

total).

8 Conclusion

We present an ablation study of AFDP on 168 WebArena-Verified Hard tasks that identifies loop detection with reflection prompts as the primary driver of web agent performance improvement (+14.9pp). Conservative observation pruning provides 52% token reduction but does not improve success rate, due to the systematic removal of navigation-critical elements that do not match task-intent keywords. We propose that web agent development should prioritize failure recovery mechanisms over observation reduction, and advocate for cost-per-completed-task as the standard evaluation metric. Future work should explore navigation-aware pruning that learns from successful agent trajectories, and multi-model evaluation to confirm the generalizability of reflection-based loop recovery.

AI disclosure. This paper was written with the assistance of AI tools (Claude, Anthropic).

References

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 *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Amine El hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. Webarena verified: Reliable evaluation for web agents. In *Workshop on Scalable Evaluation of AI Systems (SEA), NeurIPS 2025*, 2025.

Imene Kerboua and Sahar Omidi Shayegan. Focusagent: Simple yet effective ways of trimming the large context of web agents, 2025.

Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. `https://playwright.dev/`, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Jiayuan Zhang, Kaiquan Chen, Zhihao Lu, and Enshen Zhou. Prune4web: Dom tree pruning programming for web agent, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.