WABER: EVALUATING RELIABILITY AND EFFICIENCY OF WEB AGENTS WITH EXISTING BENCHMARKS

Su Kara *
Department of Computer Science
Stanford University
{sukara}@cs.stanford.edu

Fazle Faisal & Suman Nath
Microsoft Research
Redmond, USA
{fafisal,suman.nath}@microsoft.com

ABSTRACT

Most existing web agent benchmarks evaluate agents solely based on their task completion rate, excluding other crucial aspects of agent behavior that impact their usability and deployability in real-world. We propose incorporating two important metrics into a web agent benchmark: *reliability* that assesses how consistently the agent completes tasks despite transient web unreliability that are common in the wild, and *efficiency* that measures the speed and cost-effectiveness of the agent's task completion. Developing new benchmarks to measure these metrics would take significant efforts. To address this, we introduce a novel network proxy-based solution called WABER, which enables the evaluation of these two metrics on *existing agents and benchmarks without requiring any modifications to them*. This allows agent developers to adopt it effortlessly on any agent and benchmark, with zero developer effort. Using our WABER prototype, we evaluated two existing agents on the WebArena benchmark: Stacked LLM Policy and Agent Workflow Memory. Our results show that current SoTA agents struggle to complete tasks on the WABER framework, demonstrating the need to design agents that are able to generalize to real-word, unreliable scenarios. We make WABER available as an open-source tool.

1 INTRODUCTION

Autonomous web agents can boost human productivity by automating complex workflows and performing daily tasks on websites using natural language commands. Recent advances in reasoning and acting (ReACT) patterns (Yao et al., 2023), vision-based large language models (VLLMs), and agentic workflow frameworks like AutoGen (Wu et al., 2023) improve such agents' ability to understand user intent, perceive their environment, and take actions accordingly. Many agents developed in academia and industry showcase the rapid progress of these capabilities (e.g., SteP (Sodhi et al., 2024b), Agent Workflow Memory (AWM) (Wang et al., 2024c), WebPilot (Zhang et al., 2024), AutoAgents (Chen et al., 2023), WebNaviX (Shlomov et al., 2024a), Agent Q (Putta et al., 2024), Magentic-One (Fourney et al., 2024)).

Benchmarking is essential for evaluating and comparing different agents, as it provides a standardized set of realistic and deterministic tasks and evaluators. Benchmarking helps identify an agent's strengths and weaknesses, highlights areas for improvement, and ensures the agent is capable of performing the tasks defined by the benchmark. Many benchmarks exist for web agents that perform tasks on websites; examples include WebArena (Zhou et al., 2023), Visual WebArena (Koh et al., 2024), WorkArena (Drouin et al., 2024), Mind2Web (Deng et al., 2023), Web Voyager (He et al., 2024), WebLINX (Lù et al., 2024), and Windows Agent Arena (Bonatti et al., 2024).

Most existing benchmarks evaluate agents mainly by their *success rate*, the percentage of tasks they complete correctly. Typically, benchmarks use evaluators that check the binary outcome of a task, for example, by matching the URL or content of the final web page. However, success rate alone does not fully capture an agent's capabilities. For example, two agents can have the same success rate,

*Work done while at Microsoft Research.

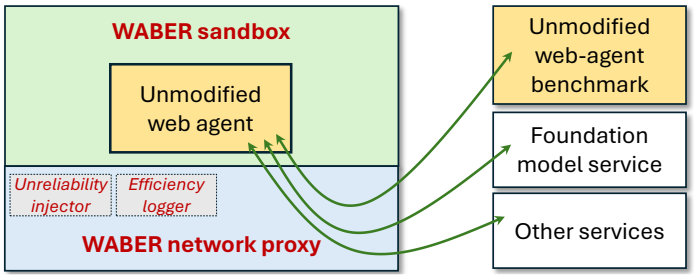


Figure 1: WABER architecture

but one can be much faster, more efficient, and more robust than the other. A benchmark that only measures the success rate cannot differentiate them or identify relative strengths and weaknesses of the underlying techniques.

To address this, we propose two additional metrics for evaluating web agents: reliability and efficiency. These metrics align with prior work emphasizing human similarity and efficiency in agent design(Wang et al. (2024a); Shlomov et al. (2024b); Goyal et al. (2024)). The reliability metric assesses how well an agent handles web tasks under transient failures, such as network slowdowns, server errors, or unexpected client-side pop-ups, that may happen in the wild. While humans adapt to such issues by reloading pages or dismissing pop-ups, agents may struggle. Measuring reliability highlights an agent's ability to mimic human resilience in unpredictable environments.

The efficiency metric measures an agent's speed and resource usage. Fast, cost-effective agents improve usability, especially in time-sensitive tasks like checking out a book online. Excessive use of resources, such as LLM tokens, can lead to higher costs or hitting usage limits, making the agent impractical. Prior research notes that many SoTA agents are unnecessarily slow and resource-intensive (Kapoor et al., 2024), undermining user experience despite task success.

It is possible to develop new benchmarks to evaluate these two metrics, with unreliable behaviors built into the benchmarks and efficiency explicitly measured by the web agents. For example, existing works introduce new benchmarks to assess metrics beyond success rate (e.g., ST-WebAgentBench (Levy et al., 2024) and R-judge for safety (Yuan et al., 2024)). However, creating new benchmarks require significant efforts. In this paper, we propose a novel technique that can evaluate reliability and efficiency metrics *for existing web agents on existing benchmarks without any modification to them.*

Our key insight is that web agents interact with benchmarks and LLM services through the network, and it is possible to introduce unreliabile behavior to the benchmarks and to measure efficiency of the agent by intercepting their communication, without any modification to the agent and the benchmark. Our solution runs agents in a sandbox, routing all network communication through a network proxy (Cortesi et al., 2010–) that (1) simulates client-, network-, and server-side unreliability during interactions with benchmarks and external services (e.g., LLMs), and (2) measures efficiency metrics like token usage and latency. The ability to use unmodified agents and benchmarks enables developers measure efficiency and reliability of their agents with low effort, with the flexibility of using their favorite benchmarks.

We have implemented a prototype of this solution, called WABER[1], and we release it as an open-source project. Our experimental results show that current SoTA agents, despite high success rates on the WebArena benchmark (Koh et al., 2024), struggle with the unreliable scenarios introduced by the WABER framework. Additionally, our framework enables seamless evaluation of agent efficiency, offering new insights into performance beyond success rates.

[1]WABER : Web Agent Benchmarking for Efficiency and Reliability, `https://github.com/SumanKNath/WABER.git`

2 WABER DESIGN

Figure 1 shows the high-level architecture of WABER. It consists of two main components: the WABER sandbox and the WABER network proxy. The sandbox isolates the web agent in a controlled environment that enforces all network interactions from inside the sandbox to go through the network proxy. An agent inside the sandbox can perform tasks as usual, interacting with external services like web-based benchmarks, foundation model APIs, and other endpoints.

The WABER network proxy intercepts all network interactions between the agent and external services. This enables the logging of key performance metrics, such as latency and the number of API calls. Additionally, it enables introducing disruptions through HTTP response modifications, such as unexpected pop-ups, to evaluate the agent's ability to handle unreliable conditions it may encounter in the real world. We elaborate on these two key functionalities below.

2.1 EFFICIENCY LOGGING VIA NETWORK INTERCEPTION

WABER measures two types of efficiency metrics: *network-level* and *application-level*. *Network-level* metrics include end-to-end task completion latency and the number of requests made to a specific service, such as a remote LLM service. These metrics can be computed at the network layer of the communication, without examining the content of the agent's requests and responses. They indicate whether an agent can complete a task within a reasonable timeframe and with an acceptable number of remote API calls. The latter is particularly important when the cost of a remote service is based on the number of API calls, or if the service throttles or blocks calls when a large number of requests are made within a short period. To measure network-level metrics, the WABER proxy intercepts each request and each response. Users can configure WABER with a script to log requests and responses for specific remote addresses (e.g., to an LLM service); otherwise, WABER outputs request counts for all remote addresses.

If the communication is encrypted (e.g., when the agent uses HTTPS), the proxy acts as a "man in the middle" between the agent and remote servers. To decrypt encrypted traffic, we install the proxy's built-in Certificate Authority (CA) on the sandbox, allowing the proxy to decrypt and encrypt traffic to and from any client running inside the sandbox.

Application-level metrics, on the other hand, in the context of efficiency measurement, require parsing the content of requests and responses to extract specific details, such as the token counts of LLM prompts and completions involved in task execution.

Note that the above mechanisms treat the agent as a black box running inside the sandbox. This ensures that efficiency evaluations remains independent of the agent's internal architecture, making the WABER framework suitable for evaluating third-party or custom-built agents.

WABER can report aggregated efficiency metrics (e.g, total latency) for a sequence of executed tasks. To achieve that, a user configures WABER with a signature of the first network request that WABER can use to identify the beginning of each task.

2.2 RELIABILITY EVALUATION VIA UNRELIABILITY INJECTION

WABER evaluates an agent's reliability by introducing controlled unreliable scenarios that mimic real-world challenges and then measuring its success rate despite the unreliability.

To do this, WABER intercepts and modifies responses at the *application level*, altering the actual content seen by the agent without changing the agent or the benchmark itself. The WABER proxy, acting as a "man in the middle," enables this capability by parsing and rewriting response payloads in real time. WABER supports the following three types of unreliable scenarios that are common in the real world:

- **Client-side Popups**: This simulates unexpected UI elements that require user interaction such as dismissing the popup before processing the page. WABER introduces popups by modifying the response HTML from benchmark websites.

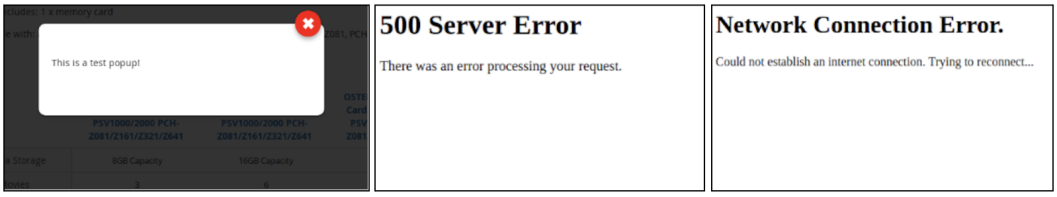


Figure 2: Unreliable Scenarios: Pop-up (left), 500 server error (center), Network error with Timed Delay (right)

- **Network Errors**: This involves introducing client-side connectivity issues, such as slow loading and connection timeouts. WABER proxy introduces such issues by adding a delay at the proxy.
- **Transient Server-side Errors**: These represent transient server-side failures that are usually handled by retrying the request. The failures include those that generate HTTP errors such as 408 (Request Timeout), 429 (Too Many Requests), 502 (Bad Gateway), 503 (Service Unavailable), and 504 (Gateway Timeout). These failures are implemented by the proxy by replacing the original response (which usually has the HTTP status code 200) with a new empty response containing an HTTP error code.

Figure 2 shows the client-side user experience under some unreliable scenarios.

Failure injection policies: Users can configure the frequency and mode of injected failures. The space of policies is large, but for ease of use, WABER includes the following two built-in policies for how often a failure mode is introduced: (1) only once, on the k'th loaded page ($k = 1$ means the first loaded page), (2) randomly n times throughout a task ($n = 1$ means the failure is injected once throughout the task). Users can select one of these policies for a given mode of failure or define their own policies with a mix of different types of failures.

3 IMPLEMENTATION

We have implemented a prototype of WABER, consisting of the following two key components.

Network proxy: We leverage `mitmproxy`(Cortesi et al., 2010–) to transparently capture network interactions and modify traffic in real-time. Mitmproxy is an open-source HTTP(S) proxy that allows for decrypting encrypted traffic on the fly. By default, Mitmproxy listens on port 8080 to intercept and process requests. We install Mitmproxy's built-in Certificate Authority (CA) as a trusted certificate within WABER sandbox. This ensures that Mitmproxy can intercept, decrypt, and re-encrypt HTTP(S) traffic by acting as a "man in the middle" between any client (e.g., the agent) inside the sandbox and a server outside it.

To introduce controlled unreliable behavior into a benchmark, we leverage Mitmproxy's addon mechanism.[2] The addon mechanism allows injecting custom logic to hook into and modify Mitimproxy's behavior on how it forwards/blocks/manipulate traffic. We create a WABER addon that implements the techniques described in Section 2.

WABER addon for Mitmproxy operates in conjunction with a `config.json` file, which allows users to specify the types of unreliable conditions they wish to simulate.

Sandbox: Our prototype uses a Linux sandbox that can run any Linux-based agent. Note that some agents run within their own sandboxes. For example, agents such as Agent Workflow Memory (AWM) Wang et al. (2024c) and Stacked LLM Policy (SteP) Sodhi et al. (2024b), use Playwright (Microsoft, 2020) to launch a Docker container as their execution environment. In such cases, we need to ensure that all network traffic made within the container, is captured by `mitmproxy`. To achieve this, we configure Playwright to use an explicit HTTP proxy by adding the following setting: `proxy={"server": "http://{your_server_hostname}:{port_number}"}`

[2] `https://docs.mitmproxy.org/stable/addons-overview/`

This routes all Playwright-driven network interactions, including remote service requests, through Mitmproxy.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

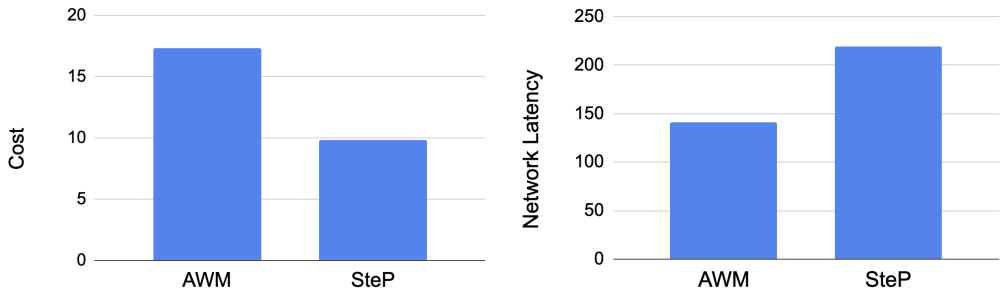
We used WABER to evaluate two existing agents, AWM (Wang et al., 2024c) and SteP (Sodhi et al., 2024b), on an existing benchmark WebArena (Zhou et al., 2023). We used all these existing systems without any modificaiton to them, highlighting a key benefit in adopting WABER for efficiency and reliability evaluation. WebArena is a standalone, self-hostable benchmark that is widely used for evaluating web agents. AWM and SteP are two of the top-performing web agents on the WebArena leaderboard. We choose these two agents because of the diversity of their underlying techniques. AWM constructs reusable task workflows by identifying and leveraging common action patterns from past tasks to optimize future performance. In contrast, SteP approaches web tasks as a Markov Decision Process, employing a dynamic stack of policies for flexible, adaptive decision-making. AWM reasons about the current state of a website using both vision and a text-based LLM, analyzing the page's screenshot and HTML. SteP relies solely on a text-based LLM to process the page's DOM tree.

We configured the agent to use the GPT-4o model via Azure OpenAI. All experiments were conducted on x86-64 Linux Virtual Machines. For prompt generation, we used the default templates provided in `webarena/agents/legacy/dynamic_prompting.py` Wang et al. (2024b) and `src/browser_env/prompts/step_fewshot_template.py` Sodhi et al. (2024a). We ran the AWM agent using the suggested `pipeline.py` file, which combines the steps of running inference on a set of tasks, evaluates agent-generated tasks and trajectory, and integrates the trajectory workflows to agent memory. We ran the SteP agent using the command `python scripts/evaluate/eval_webarena.py --config configs/webarena/eval_openai_agent.yml`.

4.2 AGENT EFFICIENCY

WebArena consists of 812 tasks in six different domains such as Reddit, a Shopping website (OneStopShop), an online store content management system (CMS), Gitlab, a Map, and English Wikipedia. In our experiments, we could not configure WebArena to execute tasks that involve Maps. Wikipedia primarily functions as a supplementary resource, often in conjunction with Maps and GitLab. Given that Wikipedia and GitLab were used together in only five tasks, we opted to focus our experiments on the remaining four environments: Reddit, Shopping, CMS, and GitLab. Overall, we use 655 tasks for evaluating efficiency of the agents with WABER. Figure 3 shows two efficiency metrics measured by WABER: average cost (in US cents) and latency (in sec) of a task. WABER reports cost of a task by logging the number of input and output tokens to GPT-4o, and translating that to the equivalent dollar value by using the pricing information in OpenAI website.[3] Figure 3 also shows the average success rate of the two agents for the tasks we considered with the GPT-4o model.

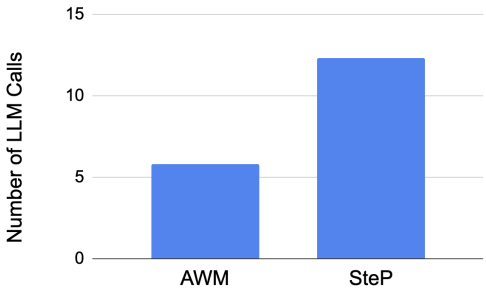
The efficiency results indicate that AWM is generally more expensive than SteP (Figure 3(a)), even though it completes tasks faster (Figure 3(b)) and with a higher success rate (Figure 3(c)) than SteP. AWM's higher cost comes from its use of longer prompts (an average of 66945.44 prompt tokens compared to SteP's 34591.30). The longer prompts are due to the additional HTML that AWM sends to the LLM. Despite sending long prompts to the LLM, AWM is on average 35.8% faster than SteP in completing a task, mainly because it makes 52.96% fewer interactions with the LLM (Figure 3(d)). These findings suggest that AWM achieves higher efficiency by leveraging longer prompts and reducing the need for multiple interactions, whereas SteP, despite lower token usage per request, requires more API calls to complete tasks.



(a) Cost (in US cents) per task

(b) Time (in seconds) per task

(c) Task success rate



(d) Number of LLM Calls per task

Figure 3: (a) Cost, (b) Latency, (c) Accuracy, and (d) Number of LLM calls per task. All 655 successfully and unsuccessfully completed tasks are considered.

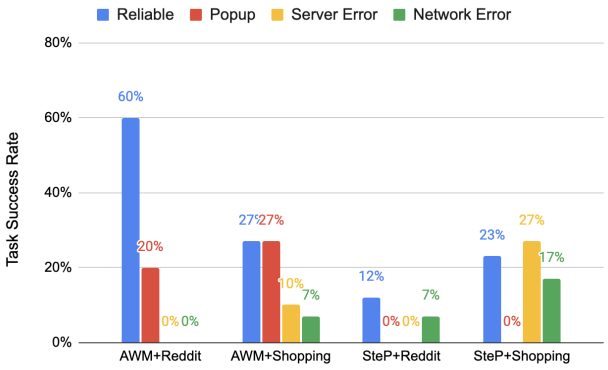


Figure 4: Accuracy of agents under unreliable scenarios

(a) Cost (in US cents) per task

(b) Time per task

Figure 5: (a) Cost and (b) Latency per task for Reliable Scenarios. All successfully and unsuccessfully completed tasks are considered.

4.3 Agent Reliability

We now report our results of using WABER to systematically test agent robustness by introducing real-world failures such as popups, server errors, and network disruptions. Testing for various failures is time consuming; we therefore perform our experiments on a subset of WebArena tasks. Specifically, we use the first 30 tasks of the Reddit and Shopping domains in WebArena. Figure 4 and Figure 5 show the success rates, cost, and latency of the agent for these tasks under various unreliable scenarios. Overall, both the agents struggle to successfully complete tasks under unreliable scenarios, even when they extensively use LLMs (shown by the increase in LLM cost and latency). Comparatively, AWM's vision-based approach provides better resilience than SteP's DOM-based approach.

Handling Client-side Popups: Popups introduce significant challenges for both agents, but AWM demonstrates a greater ability to recover. In Reddit, AWM's success rate drops from 60% in a reliable setting to 20% when popups appear, while SteP's success rate falls from 12% to 0%. After further analysis, we were able to deduce that if page HTML is not passed in as part of the observation at every step, neither agent is able to overcome the pop-up. The HTML provides information that there is an overlay added. Even though the DOM structure shows a clickable (x) and is added as a new button, and even though a screenshot clearly shows (to a human) that a pop-up is displayed, the agent cannot infer without explicitly being provided the page HTML that it must click the (x) to proceed with the original task. This underscores an important dilemma in agent design: while people want to design agents which are affordable and efficient (low in prompt tokens per LLM request), we also want to design agents which are reliable in the real world. Additionally, this highlights the importance of agents adopting adaptive strategies, as hard-coded policies struggle in dynamic environments Kapoor et al. (2024). Although AWM was able to dismiss the pop-up using the page HTML, its accuracy still drops significantly. This shows that strong performance in the containerized WebArena benchmark, a controlled environment, does not necessarily translate to success in the real world, where unreliable scenarios may arise. Agents also incur additional latency and LLM cost in handling popups. For example, in Reddit tasks, AWM experiences a 34.3% increase in average latency and a 2.58× increase in cost.

Response to Server Errors: Both agents demonstrate limitations when confronted with transient server failures, though their handling strategies diverge significantly. When encountering server errors, AWM often informs the user of the disruption and ceases task execution. SteP, in contrast, occasionally exhibits more proactive behavior, such as employing the 'go_back'action or refreshing the page by revisiting the same URL. This behavior mirrors human-like attempts to recover from errors but lacks consistency across different environments. For instance, while SteP demonstrates this adaptive approach in Shopping scenarios, it fails to replicate similar recovery strategies on Reddit.

[3] https://openai.com/api/pricing/

Success rates underscore these behavioral discrepancies. AWM's accuracy drops to 0% in Reddit and 10% in Shopping, whereas SteP completely fails in Reddit (0%) but shows resilience in Shopping with an accuracy of 27%. These findings suggest that while SteP's policy allows for occasional recovery, it lacks the robustness required for consistent performance across varied platforms. Conversely, AWM's straightforward error-reporting strategy contributes to its stable but suboptimal performance.

Response to Network Errors: Network errors similarly impact agent's success rates. AWM tends to wait during network errors, informing the user of the issue. While this improves human-agent interaction, AWM doesn't consistently reload the page, which a human would typically do to resolve connectivity issues. However, it is also common for network connections to recover without manual intervention, explaining AWM's passive approach. In Reddit, AWM's accuracy falls from 60% to 0%, and SteP's falls from 12% to 7%. In Shopping, AWM reaches 7%, and SteP outperforms it with a success rate of 17%. These results indicate that while agents attempt to recover, they do not do so effectively. AWM tends to wait for errors to resolve on their own, while SteP inconsistently attempts page refreshes.

4.4 WABER Accuracy and Overhead

Accuracy: We would like to emphasize that introducing WABER does not lead to any noticeable decline in agent accuracy. To demonstrate this, we conducted experiments over 30 Reddit and 30 Shopping tasks, totaling 60 tasks, tested with both SteP and AWM agents, and tested in both scenarios, where WABER was introduced vs. where it was never used. The results show that introducing WABER does not result in a significant difference in accuracy. In some cases, accuracy is slightly higher with the proxy in place. Specifically, when using WABER, the measured accuracy decreases by 1.67% (-1 task successes) for SteP but increases by 5% for AWM (+3 tasks successes). This minor variation highlights that WABER does not interfere with an agent's performance, while still providing transparent network-level measurements.

Overhead: We evaluated WABER's overhead using the same experimental setup as our accuracy measurement. We would like to note that latency can be measured in two ways with WABER: client latency, recorded as the time the agent begins to set up the environment to start the task until the task ends and the environment is ultimately closed, and network latency, which is specifically measured by WABER. Network latency is calculated as the time difference between the first LLM request for a task and the last LLM response for the task. It is important to note that network latency ignores the time a client spends before sending the first LLM request and after receiving the last response, and therefore differs from the latency a client experiences in completing a task. This difference in start time is evident in our calculations. Because task completion is I/O-intensive and network communication constitutes a significant portion of the total completion time, WABER's reported network latency remains representative of real-world client-side latencies.

To evaluate the overhead of WABER itself, we focused on client latency, as network latency can only be calculated when WABER is introduced. Our experiments show that while the difference in average client latency per task using WABER is relatively small (a 13.7% increase on average for SteP and a 10.5% increase for AWM), it is still a significant increase. This may suggest that, in a real-world environment where experiments are conducted on hundreds of tasks, this difference could scale and become more apparent. In future work, we hope to address this issue and provide a more comprehensive evaluation of accuracy and overhead across more tasks, benchmarks, and agents.

5 Conclusion

We have introduced WABER, a network proxy-based solution for evaluation the reliability and efficiency metrics of existing web agents on existing benchmarks. WABER can highlight the weakness and strength of an agent in handling real-world, unreliable scenarios. A key feature of WABER is that it does not require any modifications to existing agents and benchmarks, significantly lowering the efforts required to evaluate efficiency and reliability of web agents. Our evaluation of two existing agents on the WebArena benchmark using WABER demonstrates the need for developing more robust and efficient web agents. We hope that WABER will serve as a valuable tool for the

research community, enabling the development of web agents that are better suited for real-world applications.

REFERENCES

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024. URL https://doi.org/10.48550/arXiv.2409.08264.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023. URL https://doi.org/10.48550/arXiv.2309.17288. Presented at IJCAI 2024.

Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy, 2010–. URL https://mitmproxy.org/. [Version 11.1].

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023. URL https://doi.org/10.48550/arXiv.2306.06070. NeurIPS 2023 Spotlight.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024. URL https://doi.org/10.48550/arXiv.2403.07718.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. URL https://doi.org/10.48550/arXiv.2411.04468.

Nitesh Goyal, Minsuk Chang, and Michael Terry. Designing for human-agent alignment: Understanding what humans want from their agents. *arXiv preprint arXiv:2404.04289*, 2024. URL https://doi.org/10.48550/arXiv.2404.04289.

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. *arXiv preprint arXiv:2401.13919*, 2024. URL https://doi.org/10.48550/arXiv.2401.13919. Accepted to ACL 2024.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024. URL https://doi.org/10.48550/arXiv.2407.01502.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. URL https://doi.org/10.48550/arXiv.2401.13649. Accepted to ACL 2024.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. St-webagentbench: A benchmark for evaluating safety and trustworthiness in web agents. *arXiv preprint arXiv:2410.06703*, 2024. URL https://doi.org/10.48550/arXiv.2410.06703.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024. URL https://doi.org/10.48550/arXiv.2402.05930.

Microsoft. Playwright. https://playwright.dev, 2020. Accessed: 2024-02-02.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024. URL https://doi.org/10.48550/arXiv.2408.07199.

Segev Shlomov, Ben Wiesel, Aviad Sela, Ido Levy, Liane Galanti, and Roy Abitbol. From grounding to planning: Benchmarking bottlenecks in web agents. *arXiv preprint arXiv:2409.01927*, 2024a.

Segev Shlomov, Avi Yaeli, Sami Marreed, Sivan Schwartz, Netanel Eder, Offer Akrabi, and Sergey Zeltyn. Ida: Breaking barriers in no-code ui automation through large language models and human-centric design. *arXiv preprint arXiv:2407.15673*, 2024b. URL https://doi.org/10.48550/arXiv.2407.15673.

Paloma Sodhi, S.R.K. Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions. https://github.com/asappresearch/webagents-step, 2024a.

Paloma Sodhi, S.R.K. Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions. *arXiv preprint arXiv:2310.03720*, 2024b. URL https://doi.org/10.48550/arXiv.2310.03720.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhi-Yuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 0(0): 1–42, 2024a. doi: 10.1007/s11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. https://github.com/zorazrw/agent-workflow-memory.git, 2024b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://doi.org/10.48550/arXiv.2308.08155.

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) Workshops*, 2023.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-judge: Benchmarking safety risk awareness for llm agents. *arXiv preprint arXiv:2401.10019*, 2024. URL https://doi.org/10.48550/arXiv.2401.10019. EMNLP Findings 2024.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. *arXiv preprint arXiv:2408.15978*, 2024. URL https://doi.org/10.48550/arXiv.2408.15978.

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. *arXiv preprint arXiv:2307.13854*, 2023. URL https://doi.org/10.48550/arXiv.2307.13854.

6 APPENDIX

We note that SteP's default prompt instructions are configured to "draft" posts rather than fully "submit" them, which likely resulted in lower accuracy when evaluated using WebArena's output comparison methods. Since WebArena expects completed actions that match the final output, SteP's partial task completion may have negatively impacted its performance metrics. We intentionally did not modify this behavior to maintain the integrity of the agent's original design. Additionally, we note that since AWM agent relies on memory of past experiments in order to learn common workflow patterns overtime, the subset of thirty tasks may not be sufficient for its optimal performance. Again, we would like to urge the reader keep in mind that the goal of the paper is not to assess the accuracy of the agents but encourage agent developers to use the WABER framework to report the efficiency and reliability of their agents and experiment with their own unreliable scenarios when evaluating their agents' performance on real-world tasks.