AI Agents for Web Testing: A Case Study in the Wild

Naimeng Ye[*], **Xiao Yu**[*], **Ruize Xu**[*], **Tianyi Peng**, **Zhou Yu**
Columbia University
{ny2336, xy2437, rx2246, tp2845, zy2461}@columbia.edu

Abstract

Automated web testing plays a critical role in ensuring high-quality user experiences and delivering business value. Traditional approaches primarily focus on code coverage and load testing, but often fall short of capturing complex user behaviors, leaving many usability issues undetected. The emergence of large language models (LLM) and AI agents opens new possibilities for web testing by enabling human-like interaction with websites and a general awareness of common usability problems. In this work, we present WebProber, a prototype AI agent-based web testing framework. Given a URL, WebProber autonomously explores the website, simulating real user interactions, identifying bugs and usability issues, and producing a human-readable report. We evaluate WebProber through a case study of 120 academic personal websites, where it uncovered 29 usability issues—many of which were missed by traditional tools. Our findings highlight agent-based testing as a promising direction while outlining directions for developing next-generation, user-centered testing frameworks.

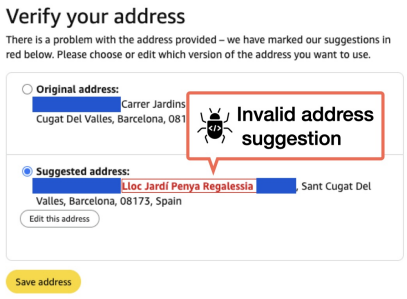
1 Introduction

The modern web hosts billions of websites (Chakarov, 2023), offering rich services and content that span nearly every aspect of daily life. Common web applications include e-commerce websites such as Amazon, social media platforms like Facebook, information portals like Wikipedia along with a vast number of personal websites.

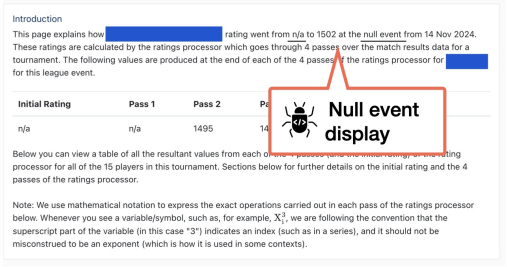
To ensure the quality and reliability of these web applications, automated web testing has become a critical component of modern web development cycles. Traditional web testing approaches, such as static and dynamic analysis (Ricca & Tonella, 2001), have been crucial in mitigating common issues and vulnerabilities such as layout and functional bugs. These traditional approaches mainly rely on verifying code paths, automating scripted UI interactions, and measuring load performance using established tools like Cypress, Puppeteer, and JMeter (Cypress.io, 2025; Google Chrome Developers, 2025; Apache Software Foundation, 2025). Despite these efforts, such approaches face significant challenges in detecting real-world, user-facing issues. Since real users' actions are highly diverse and context-dependent, software-based methods often fail to cover test-cases that capture the full spectrum of user behavior. This results in many undetected bugs and missing features that degrade user experience (see Figure 1 for real-world examples).

We introduce **WebProber**, a highly extensible web testing framework that leverages AI agents to simulate complex human behaviors on the web. Unlike existing approaches (Le et al., 2025; Wang et al., 2025; Lu et al., 2025) that use large language models to generate test cases or interact with post-processed HTML files, WebProber employs powerful visual language models (VLMs) (Bordes et al., 2024) to interact directly with visual webpages like human testers. Given a URL, WebProber explores the webpage for common user-side bugs by performing actions such as clicking, typing, and scrolling. It generates a comprehensive report of unexpected website behaviors based on its interaction history. We illustrate this workflow in Figure 2, which consists of three stages.(1) a proposal module that suggests error-prone features to investigate, guided by a bug database, (2) an interaction module that simulates user experience guided by VLMs, and (3) a report generation module that examines the full interaction history to identify user-side bugs and suggest potential UI/UX improvements.

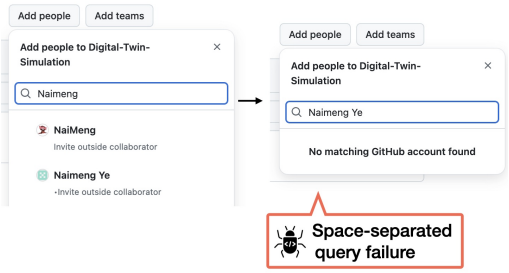
[*]Equal contribution.



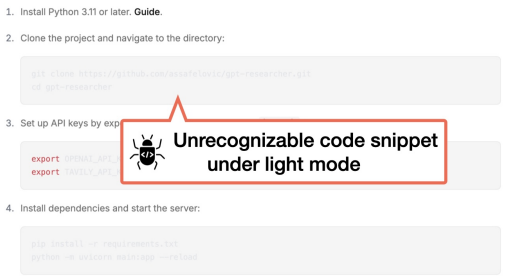
(a) Amazon



(b) USA Table Tennis (USATT) Official Website



(c) GitHub



(d) README Code Snippet

Figure 1: Website usability bugs that are not easily detected by traditional web testing techniques. (a) On the Amazon Spain website, during a purchase, the system suggests a non-existent and unclickable address. (b) The USATT website displays null event text for a league event. (c) In a GitHub organization repository, the user search function does not support queries with spaces when adding users. (d) On certain MCP server pages, code snippets in the README file are illegible in light mode due to poor color contrast.

As a case study, we deployed our framework on 120 personal websites in the wild and found that our framework is able to identify 29 usability issues that impact user experience. Many of these issues—such as textual errors and misdirected links—were not detected by traditional automated testing tools, highlighting the unique strengths of agent-based testing in uncovering subtle, human-centric problems. Our empirical study on personal websites presents a first step towards building a scalable web testing framework based on AI agents, and we hope that this work can serve as a foundation for future research in this direction.

In summary, our contributions are:

1. We introduce WebProber, a highly extensible web testing framework that leverages AI agents to simulate human behavior on the web.
2. We present a case study on 120 personal websites in the wild, on which WebProber found 29 usability issues.
3. We release our code and our human-annotated bug database for future research.[1]

2 Related Work

Browser-Use Agents With the advent of visual language models (VLMs), many works have explored the use of powerful models like GPT-4o and Claude-3.7 for web navigation tasks (Liu

[1]The github repository is available at https://github.com/TianyiPeng/WebProber. The database url is available at https://webbugvid.netlify.app/. Due to privacy concerns with some personal websites, we included only a subset of the bugs in the database.

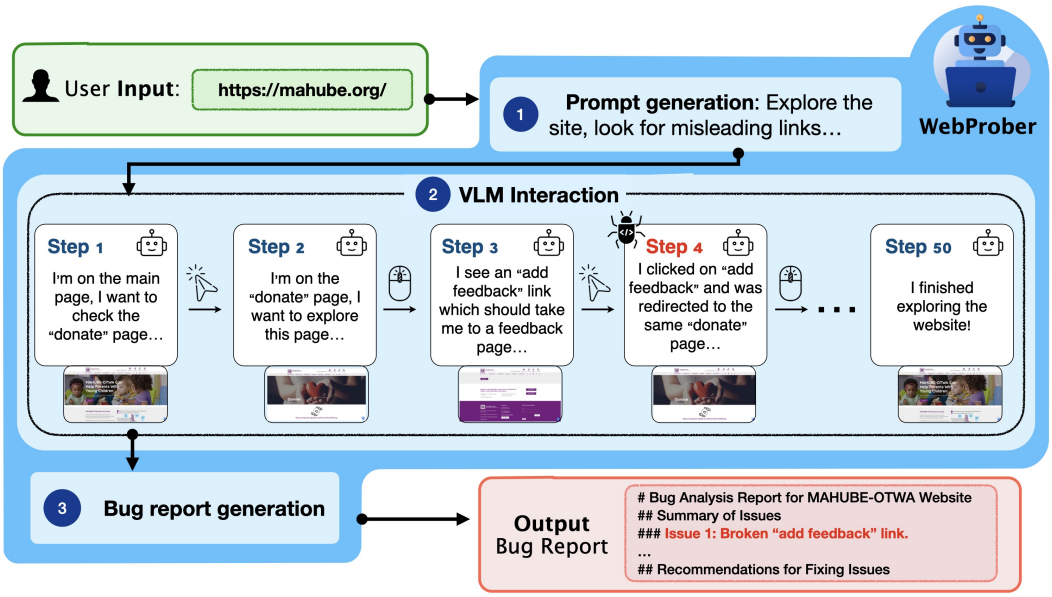


Figure 2: Workflow of WebProber. Given a user-provided URL, the agent generates a comprehensive bug report through three stages: (1) testing prompt generation, (2) VLM-guided interaction, and (3) bug report generation. For more details, refer to section 3.

et al., 2023a; Zhou et al., 2024; Koh et al., 2024a). Earlier efforts used the accessibility tree or a screenshot of the webpage as input to the VLM, and prompted it to generate actions such as clicking and typing (Yang et al., 2023; Koh et al., 2024a). Recent works have explored various strategies to further improve an agent's decision-making process, such as iteratively prompting the model to improve its own output (Madaan et al., 2023; Shinn et al., 2023), or augmenting the agent's decision process using search algorithms such as breadth- or depth-first search (Yao et al., 2023), best-first search (Koh et al., 2024b), and Monte Carlo tree search (Yu et al., 2023; 2025). However, these works typically focus on solving pre-defined tasks such as finding a specific item on a shopping website, or navigating to a specific webpage. Our work aims to use agents to *discover* bugs missed by existing automated testing tools on real-world websites.

Automated Web Testing Automated web testing emphasizes systematically testing web applications with minimal human intervention. Traditional approaches aim to generate action trajectories and can be broadly categorized into three classes: (1) randomized testing, where action sequences are generated stochastically Android Developers (2022); (2) model-based methods, which construct a state graph of the application and use graph traversal algorithms such as depth-first search to explore it Mesbah et al. (2012); Stocco et al. (2023); Liu et al. (2025b); and (3) techniques based on reinforcement learning, which generate action sequences while maximizing a reward signal Zheng et al. (2021); Sherin et al. (2023).

More recently, the field has begun to incorporate LLMs in automated web testing. They are used to expand the test action space Liu et al. (2023b; 2024b); Wang et al. (2024), and to guide navigation and interaction Alian et al. (2025); Shahbandeh et al. (2024); Liu et al. (2025a). In parallel, similar trends have emerged in mobile app testing Liu et al. (2024a); Yoon et al. (2023); Lee et al. (2024); Wen et al. (2024); Chen et al. (2025). Our work differs from prior literature by emphasizing the simulation of realistic user behaviors powered by VLMs, and by targeting contextual bugs often overlooked by traditional techniques, such as critical typographical errors or misdirected links.

3 WebProber

We present WebProber, a web testing system based on AI agents. Given a website URL, WebProber returns a detailed report enumerating user-side bugs and UI/UX issues found during its interaction with the website. WebProber operates through a three-stage pipeline: (1) generating testing prompts

that target common vulnerabilities for the particular class of website given, (2) simulating human-like web interactions, and (3) analyzing the interaction trajectory to generate comprehensive bug reports. We present an overview of this process in Figure 2, and describe this process in detail below.

Prompt generation Prompts guide an AI agent to look for common usability issues for different classes of web pages, enabling more targeted and efficient exploration by focusing on typical usability issue patterns. In this work, we created our testing prompts through an iterative refinement process and release the final high-quality prompt template in our repository. For each website type (e.g., personal websites), we begin with a preliminary prompt instructing the VLM on which features to test and what issues to detect. We then refine both the prompt and bug set through iterative cycles: applying WebProber to discover new bugs, manually verifying their reproducibility, and using a VLM to generate improved prompts based on the expanded bug set. This process continuously develops our prompt instructions while building a diverse collection of web usability bugs valuable for future evaluation. An example of prompt refinement is provided in appendix B.1. While we demonstrate one effective approach to prompt generation, any method that produces high-quality, targeted prompts for usability testing would be compatible with our framework.

Interaction simulation Using the generated testing prompts from the previous stage, WebProber employs VLM-based agents to systematically interact with the website. Building on the Browser-Use Python package (Müller & Žunič, 2024), our system iteratively (1) prompts a VLM for an action based on a website screenshot (e.g., clicking a button or entering text), (2) executes the action on the website, and (3) repeats until either the maximum step limit is reached or the target feature has been tested. Throughout this process, we preserve the complete interaction trajectory, including screenshots, reasoning traces, and actions.

Bug report generation Finally, we generate detailed bug and usability reports by analyzing the full interaction trajectory with a VLM. Since usability issues typically emerge during interactive use, the complete interaction history is important for an accurate diagnosis. Detailed prompts for report generation are provided in appendix B.2.

In our implementation, we used Claude-3.7 Sonnet (Anthropic (2025)) as the VLM for each stage of WebProber's pipeline. Each stage can be independently configured to use a different VLM, though exploring that is left as future work.

4 Experiments

To demonstrate the effectiveness of WebProber, we conducted a case study on real-world academic personal websites crawled from OpenReview author profiles. We collected 120 personal websites and applied WebProber on this dataset to detect usability issues. We then manually inspected the generated reports to analyze the detected bugs, specifically evaluating whether they represented genuine usability issues or false positives. The results are presented in Section 4.1.

In addition to measuring the capabilities of WebProber, we also investigated the coverage of bugs detectable by our framework. Since the total set of bugs on a website is unknown *a priori*, we manually inspected a representative subset of 80 websites to identify all potential bugs as a proxy for ground truth. We then ran WebProber on the same subset of websites to investigate both detected and undetected issues. The results and analysis are presented in the following sections.

4.1 Results

Our approach effectively identifies usability issues that impact user experience Across our dataset of 120 academic personal websites, WebProber successfully identified 29 usability issues (verified by the authors). In addition to bugs detectable by traditional techniques, e.g. rendering issues with images, our agent is also able to identify contextual bugs that are often overlooked by these methods. These issues span several categories, including link mistakes, rendering issues etc. We give a couple of representative examples of these usability issues as follows.

- **Broken or misdirected links** The most common class of bugs detected is broken or misdirected links. We present an example in Figure 3a: the agent identified that a project description was inconsistent with the paper linked through the "Read more here" button.
- **Logical inconsistencies** Finally, we find our WebProber is also able to detect logical inconsistencies in website contents, typically resulting from typographical errors. These errors sometimes lead to factual inaccuracies or user confusion. For example, in Figure 3b, the agent identified a spring course syllabus (determined by calendar dates) that incorrectly scheduled a "Fall break" week.

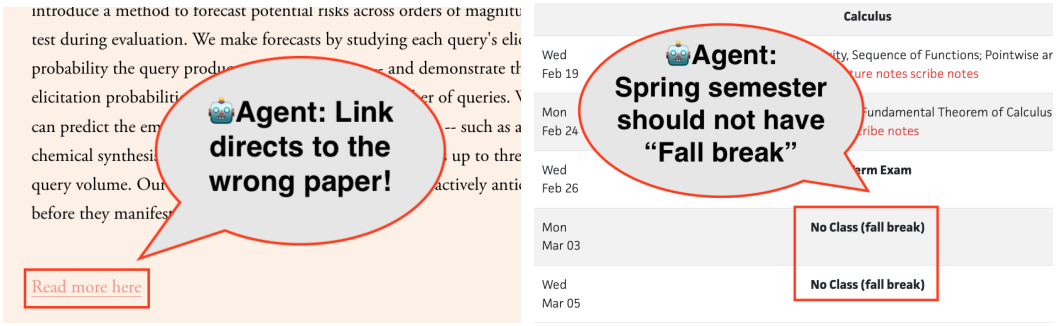
These results illustrate the capabilities of VLM-based web testing and provide insights into what types of issue can be automatically detected.

4.2 Discussion

While WebProber is able to identify real bugs and UI/UX issues in the wild, we also find numerous cases where human oversight is still needed for bug discovery. We present our findings below.

False positives While WebProber successfully discovered 29 usability issues, we found that 85% of all reported bugs across the 120 websites were false positives. The majority of these problems stemmed from technical limitations of the browser automation framework (the framework through which the agent applies actions on the webpage) rather than actual website issues. One common example is PDF access problems, which is often caused by the automation framework's security settings. However, the agent often incorrectly attributes these failures to website defects rather than automation constraints. Additionally, a small portion of false positives resulted from reasonable but incorrect assumptions of the agent, particularly when the agent lacked sufficient temporal or domain context to properly interpret website content. Figure 4 illustrates one such example.

Undetected bugs On our representative subset of 80 websites, we manually identified 32 bugs, of which WebProber successfully detected 19, achieving a coverage of 59.4%[2]. The undetected bugs fell in two primary failure modes. First, and most frequently, bugs were often located deep within the website hierarchy, requiring navigation through multiple pages. Since we executed our pipeline only once per website, the agent often terminated exploration before encountering these deeply embedded issues. Second, certain pages containing bugs were inaccessible due to dynamic content rendering issues that our current implementation cannot handle effectively. These results suggest that effective bug detection requires improved exploration strategies capable of performing systematic, long-horizon traversals of website hierarchies and handling dynamic content. We defer these enhancements to future work.



(a) Misdirected link (b) Typographical error

Figure 3: Examples of WebProber bug detection results. (a) A "Read more here" link for one research project incorrectly leads to a different paper. (b) A spring course syllabus mistakenly lists "fall break."

[2]Since the authors may not have found all possible bugs, the actual coverage may be lower.

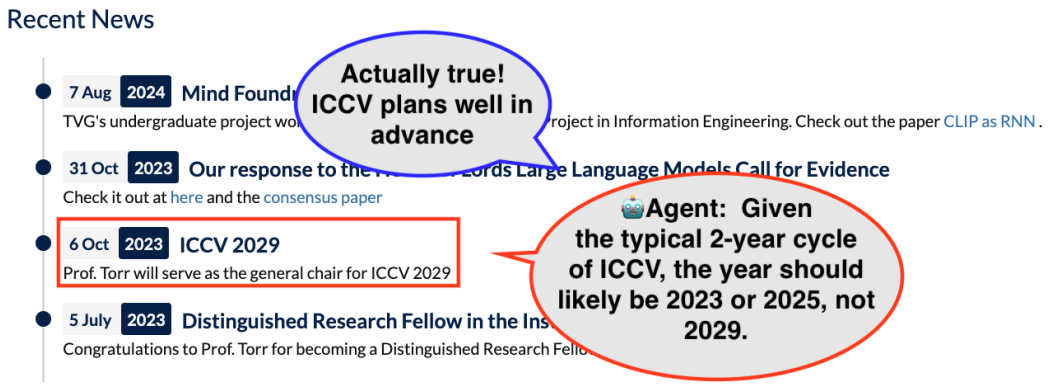


Figure 4: Example of false positive: an announcement referencing a "2029" conference is flagged as a typo, but in reality, conferences such as ICCV do plan leadership roles and organizational details several years in advance.

5 Conclusion and Future Work

We introduced WebProber, an agent-based web testing framework. Applied to 120 academic personal websites, WebProber uncovered 29 usability issues—many missed by traditional tools—demonstrating the potential of agent-driven testing. This case study also revealed several challenges and future directions:

Agent-Browser Interaction. Agent interactions remain unreliable—misclicks, erratic navigation, and poor performance on complex sites contribute to false positives. Enhancing browser control fidelity is a key priority.

Bug Coverage and Training. Current agents are not optimized for bug discovery. Reinforcement learning and hybrid approaches incorporating traditional automated web testing tools may improve coverage and effectiveness.

Lack of Benchmarks. Progress is hindered by the absence of a standardized benchmark for web usability issues. Curating datasets like SWEBench would support training and evaluation.

Web Testing in Other Domains. Vibe-coded websites, startup landing pages, and non-profit websites often involve quick prototyping with limited budgets for thorough quality assurance. AI-generated sites, in particular, may contain bugs that their creators—often without professional development expertise—are unable to detect. While we believe AI agent-based web testing could significantly benefit these cases, we leave a rigorous field study for future work.

References

Parsa Alian, Noor Nashid, Mobina Shahbandeh, Taha Shabani, and Ali Mesbah. Feature-driven end-to-end test generation. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 678–678. IEEE Computer Society, 2025.

Android Developers. Monkey. https://developer.android.com, 2022. Accessed: 2025-06-25.

AI Anthropic. Claude 3.7 and claude code, 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet.

Apache Software Foundation. Apache jmeter: Load testing for web applications. https://jmeter.apache.org/, 2025. Accessed: 2025-06-26.

Florian Bordes, Richard Yuanzhe Pang, Anurag Ajay, Alexander C Li, Adrien Bardes, Suzanne Petryk, Oscar Mañas, Zhiqiu Lin, Anas Mahmoud, Bargav Jayaraman, et al. An introduction to vision-language modeling. *arXiv preprint arXiv:2405.17247*, 2024.

Radoslave Chakarov. How many websites are there? how many are active in 2023? https://webtribunal.net/blog/how-many-websites, 2023.

Mengzhuo Chen, Zhe Liu, Chunyang Chen, Junjie Wang, Boyu Wu, Jun Hu, and Qing Wang. Standing on the shoulders of giants: Bug-aware automated gui testing via retrieval augmentation. *Proceedings of the ACM on Software Engineering*, 2(FSE):825–846, 2025.

Cypress.io. Cypress: Testing frameworks for javascript. https://www.cypress.io/, 2025. Accessed: 2025-06-26.

Google Chrome Developers. Puppeteer: Headless chrome node.js api. https://pptr.dev/, 2025. Accessed: 2025-06-26.

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, 2024a. URL https://arxiv.org/abs/2401.13649.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents, 2024b. URL https://arxiv.org/abs/2407.01476.

Nguyen-Khang Le, Quan Minh Bui, Minh Ngoc Nguyen, Hiep Nguyen, Trung Vo, Son T. Luu, Shoshin Nomura, and Minh Le Nguyen. Automated web application testing: End-to-end test case generation with large language models and screen transition graphs, 2025. URL https://arxiv.org/abs/2506.02529.

Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 1119–1133, 2024.

Chenxu Liu, Zhiyu Gu, Guoquan Wu, Ying Zhang, Jun Wei, and Tao Xie. Temac: Multi-agent collaboration for automated web gui testing. *arXiv preprint arXiv:2506.00520*, 2025a.

Chenxu Liu, Junheng Wang, Wei Yang, Ying Zhang, and Tao Xie. Judge: Effective state abstraction for guiding automated web gui testing. *ACM Transactions on Software Engineering and Methodology*, 2025b.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2023a. URL https://arxiv.org/abs/2308.03688.

Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. Fill in the blank: Context-aware automated text input generation for mobile gui testing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1355–1367. IEEE, 2023b.

Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024a.

Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Zhilin Tian, Yuekai Huang, Jun Hu, and Qing Wang. Testing the limits: Unusual text inputs generation for mobile app crash detection with large language model. In *Proceedings of the IEEE/ACM 46th international conference on software engineering*, pp. 1–12, 2024b.

Yuxuan Lu, Bingsheng Yao, Hansu Gu, Jing Huang, Zheshen Jessie Wang, Yang Li, Jiri Gesi, Qi He, Toby Jia-Jun Li, and Dakuo Wang. Uxagent: An llm agent-based usability testing framework for web design. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Ali Mesbah, Arie Van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB)*, 6(1): 1–30, 2012.

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

F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pp. 25–34, 2001. doi: 10.1109/ICSE.2001.919078.

Mobina Shahbandeh, Parsa Alian, Noor Nashid, and Ali Mesbah. Naviqate: Functionality-guided web application navigation. *arXiv preprint arXiv:2409.10741*, 2024.

Salman Sherin, Asmar Muqeet, Muhammad Uzair Khan, and Muhammad Zohaib Iqbal. Qexplore: An exploration strategy for dynamic web applications using guided search. *Journal of Systems and Software*, 195:111512, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Andrea Stocco, Alexandra Willi, Luigi Libero Lucio Starace, Matteo Biagiola, and Paolo Tonella. Neural embeddings for web testing. *arXiv preprint arXiv:2306.07400*, 2023.

Dakuo Wang, Ting-Yao Hsu, Yuxuan Lu, Hansu Gu, Limeng Cui, Yaochen Xie, William Headean, Bingsheng Yao, Akash Veeragouni, Jiapeng Liu, Sreyashi Nag, and Jessie Wang. Agenta/b: Automated and scalable web a/btesting with interactive llm agents, 2025. URL https://arxiv.org/abs/2504.09723.

Siyi Wang, Sinan Wang, Yujia Fan, Xiaolei Li, and Yepang Liu. Leveraging large vision-language model for better automatic web gui testing. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 125–137. IEEE, 2024.

Hao Wen, Shizuo Tian, Borislav Pavlov, Wenjie Du, Yixuan Li, Ge Chang, Shanhui Zhao, Jiacheng Liu, Yunxin Liu, Ya-Qin Zhang, et al. Autodroid-v2: Boosting slm-based gui agents via code generation. *arXiv preprint arXiv:2412.18116*, 2024.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023. URL https://arxiv.org/abs/2310.11441.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Juyeon Yoon, Robert Feldt, and Shin Yoo. Autonomous large language model agents enabling intent-driven mobile gui testing. *arXiv preprint arXiv:2311.08649*, 2023.

Xiao Yu, Maximillian Chen, and Zhou Yu. Prompt-based monte-carlo tree search for goal-oriented dialogue policy planning, 2023. URL https://arxiv.org/abs/2305.13660.

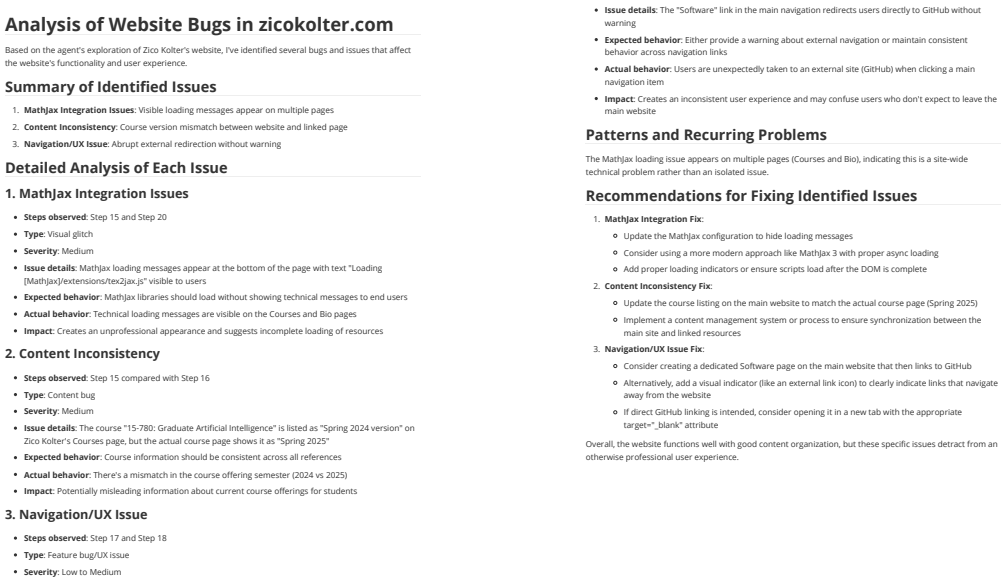
Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning, 2025. URL https://arxiv.org/abs/2410.02052.

Yan Zheng, Yi Liu, Xiaofei Xie, Yepang Liu, Lei Ma, Jianye Hao, and Yang Liu. Automatic web testing using curiosity-driven reinforcement learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 423–435. IEEE, 2021.

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, 2024. URL https://arxiv.org/abs/2307.13854.

A Example bug report generated by WebProber

Figure A1 shows an example of bug report by our WebProber. Within 20 steps, the model correctly found the inconsistency between the dates of a course on the personal website and the course page. Via prompting, the report is well-formatted with bug descriptions and details such as common patterns or recurring issues, and recommendations for fixing the bugs.



Analysis of Website Bugs in zicokolter.com

Based on the agent's exploration of Zico Kolter's website, I've identified several bugs and issues that affect the website's functionality and user experience.

Summary of Identified Issues

1. **MathJax Integration Issues**: Visible loading messages appear on multiple pages
2. **Content Inconsistency**: Course version mismatch between website and linked page
3. **Navigation/UX Issue**: Abrupt external redirection without warning

Detailed Analysis of Each Issue

1. MathJax Integration Issues

- **Steps observed**: Step 15 and Step 20
- **Type**: Visual glitch
- **Severity**: Medium
- **Issue details**: MathJax loading messages appear at the bottom of the page with text "Loading [MathJax]/extensions/tex2jax.js" visible to users
- **Expected behavior**: MathJax libraries should load without showing technical messages to end users
- **Actual behavior**: Technical loading messages are visible on the Courses and Bio pages
- **Impact**: Creates an unprofessional appearance and suggests incomplete loading of resources

2. Content Inconsistency

- **Steps observed**: Step 15 compared with Step 16
- **Type**: Content bug
- **Severity**: Medium
- **Issue details**: The course "15-780: Graduate Artificial Intelligence" is listed as "Spring 2024 version" on Zico Kolter's Courses page, but the actual course page shows it as "Spring 2025"
- **Expected behavior**: Course information should be consistent across all references
- **Actual behavior**: There's a mismatch in the course offering semester (2024 vs 2025)
- **Impact**: Potentially misleading information about current course offerings for students

3. Navigation/UX Issue

- **Steps observed**: Step 17 and Step 18
- **Type**: Feature bug/UX issue
- **Severity**: Low to Medium
- **Issue details**: The "Software" link in the main navigation redirects users directly to GitHub without warning
- **Expected behavior**: Either provide a warning about external navigation or maintain consistent behavior across navigation links
- **Actual behavior**: Users are unexpectedly taken to an external site (GitHub) when clicking a main navigation item
- **Impact**: Creates an inconsistent user experience and may confuse users who don't expect to leave the main website

Patterns and Recurring Problems

The MathJax loading issue appears on multiple pages (Courses and Bio), indicating this is a site-wide technical problem rather than an isolated issue.

Recommendations for Fixing Identified Issues

1. **MathJax Integration Fix**:
   - Update the MathJax configuration to hide loading messages
   - Consider using a more modern approach like MathJax 3 with proper async loading
   - Add proper loading indicators or ensure scripts load after the DOM is complete
2. **Content Inconsistency Fix**:
   - Update the course listing on the main website to match the actual course page (Spring 2025)
   - Implement a content management system or process to ensure synchronization between the main site and linked resources
3. **Navigation/UX Issue Fix**:
   - Consider creating a dedicated Software page on the main website that then links to GitHub
   - Alternatively, add a visual indicator (like an external link icon) to clearly indicate links that navigate away from the website
   - If direct GitHub linking is intended, consider opening it in a new tab with the appropriate target="_blank" attribute

Overall, the website functions well with good content organization, but these specific issues detract from an otherwise professional user experience.

Figure A1: An Example Bug Report on a Personal Website

B Implementation Details

B.1 Proposing Potential Bugs

To expand the database of historical issues, we follow an iterative prompting pipeline. Specifically, given the bugs found with current prompts, we pick up highly representative and domain-specific ones. Then, these bug descriptions are fed into powerful models to iteratively improve the current prompts. The updated prompts contain potential bug proposals that encourage the model to explore areas that are often more likely to be buggy. Table A1 shows the initial prompt and the prompt iterated with ten historical bugs in the personal website domain.

Table A1: Initial and Iterated Prompts

Iteration	Prompt
Initial	Go to the website [URL], a personal website. Explore the content, click on links, and occasionally pause to assess whether what is shown and linked on the website is coherent and appropriate. Unreasonable or problematic issues include, but are not limited to: broken or mismatched images/links, UI glitches/incapabilities, illogical or unfunctional web design, or textual errors, etc.
One Iteration	Go to the website [URL], a personal website. Systematically explore all accessible content, including menus, links, embedded media, interactive elements, and downloadable materials for WEBSITE BUGS. You should prioritize areas that are often more likely to contain issues. At each stage, critically evaluate whether the displayed information, layout, and behavior align with expectations for a functional and professional web experience. Carefully inspect for issues such as, but not limited to: (1) Broken elements: dead/missing links, 404 pages, failed image or video loads. (2) Interaction failures: non-responsive buttons, malfunctioning forms or filters, non-working download or redirect actions. (3) UI/UX flaws: lack of visual feedback, missing tooltips/ESC buttons, layout inconsistencies, uncustomized templates, poor mobile compatibility. (4) Content inconsistencies: outdated or contradictory data (e.g., dates or names), mismatched references or external links, typos or formatting errors. (5) Domain-specific bugs: for instance, broken external links to publications, projects, GitHub, Google Scholar, etc. Incorrect anchor links (e.g., internal navigation like #about or #projects not working). Outdated or dead email links (e.g., mailto: pointing to deprecated addresses). Missing or malformed citation info (e.g., BibTeX files, DOI links not rendering or downloading properly). Mismatched thumbnails or missing alt-text on research project previews. Videos or talks not embedded properly (e.g., iframe blocked by CORS). For each identified issue, consider its impact, repeatability, and specific trigger (e.g., ”clicking X under condition Y leads to error Z”)

B.2 Generating Bug Reports

Given the trajectories, we then prompt powerful models such as Claude-3.7 Sonnet to analyze the trajectories and generate a formatted bug report. We prompt the model to contain a summary of bugs, the steps where bugs occurred, short descriptions, and then more details such as common patterns or recurring issues, and recommendations for fixing the bugs. The full prompt is shown in Table A2.

Table A2: Prompt for Generating Bug Reports

Prompt
Please analyze the following agent run trajectory and identify any potential bugs or glitches in the website being tested. Consider both feature bugs (missing or incorrect functionality) and glitch-like bugs (visual or behavioral anomalies). Note that the type of bug is not always obvious, so don't be afraid to make an assumption. For example, if the website does not support certain features that the agent is trying to use, that is a bug (e.g. the agent is trying to use the "add to cart" feature, but the website does not have a cart, or that the agent is searching in some language that the website does not support). For each step, I'll provide: 0. The screenshot of the current browser state 1. The agent's evaluation of the step 2. The next goal 3. The action taken Please analyze the entire sequence of steps and identify: 1. Any unexpected behaviors or errors of the website itself (*note: not the agent's actions*) 2. Missing or incorrect functionality 3. Visual glitches or UI inconsistencies 4. Any other anomalies that might indicate bugs Here's the step-by-step trajectory: [Trajectory] Based on the above trajectory, please provide: 1. A summary of any bugs or glitches identified 2. The specific steps where issues occurred 3. The nature of each issue (feature bug, visual glitch, etc.) 4. Any patterns or recurring problems 5. Recommendations for fixing the identified issues For each identified issue, please specify: - The step number where it occurred - Whether it's a feature bug or visual glitch - The severity of the issue - The expected behavior vs actual behavior