BrowserArena: Evaluating LLM Agents on Real-World Web Navigation Tasks

Sagnik Anupam[*], **Davis Brown**[1], **Shuo Li**[1], **Eric Wong**[1], **Hamed Hassani**[1], **Osbert Bastani**[1]
[1]University of Pennsylvania

Abstract

LLM web agents now browse and take actions on the open web, yet current agent evaluations are constrained to sandboxed environments or artificial tasks. We introduce BrowserArena, a live open-web agent evaluation platform that collects user-submitted tasks, runs Arena-style head-to-head comparisons, and uses step-level human feedback to surface failure modes. Collecting and analyzing step-level annotations on the agent traces, we identify three consistent failure modes: captcha resolution, pop-up banner removal, and direct navigation to URLs. By constructing targeted datasets to further study these tasks, we discover variations in how different language models navigate these failure modes. We find, for example, that o4-mini deploys a wider variety of strategies to circumvent captcha resolution than other models and DeepSeek-R1 consistently misleads users about pop-up banner closure. Our findings surface both the diversity and brittleness of current web agents. More broadly, our benchmarking methodology provides an approach to evaluating and understanding web agent failure modes at scale.

1 Introduction

Recently, with the advent of web agents such as Manus and OpenAI's Operator [21], there has been significant interest in the ability of large language models (LLMs) to interact and complete tasks on diverse websites. As a result, several benchmarks have been developed to evaluate the performance of various LLMs and agent frameworks on web browsing tasks [26]. Some of these benchmarks focus on agent interaction with self-hosted websites, with success on tasks being measured using custom execution-based evaluation procedures [14]. However, "closed" benchmarks have limited task diversity [27] because they are restricted to only a few websites, so current benchmarks cannot serve as good tests of real-world web agents.

Limitations of current open-web evaluations: Recently, researchers have built systems that allow agents to browse the open web [7, 22], given the significant success of open-ended environments for agent evaluation in other domains such as software engineering [22] and general computer use [1, 24]. However, such approaches still suffer from four major drawbacks. First, in such benchmarks, tasks are described using highly specific instructions to the agent, which is unlikely to mirror how real-world users describe and perform tasks on the open web. Second, significant engineering effort is often required to incorporate new tasks into these systems because they often require ground-truth success criterion for measuring task performance [7]. This need for ground-truth success criteria limits the types of tasks that can be evaluated within these approaches. Third, since these success criteria are often evaluated using programs, they also serve as an entry barrier preventing non-technical users from contributing new tasks to these benchmarks. Due to this entry barrier, most benchmarks developed on top of such open-web environments are static, ground-truth-based benchmarks with detailed task descriptions. Finally, existing ground-truth based benchmarks can be accessed by a diverse range of LLMs with different levels of tool access and reasoning frameworks as long as the

[*]Correspondence to: Sagnik Anupam <sanupam [at] seas.upenn.edu>.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

final system produces the correct ground truth result. While this flexibility is helpful for comparing across a wide range of systems, it obscures the differences in performance due to the usage of different language models.

Our approach: A live evaluation platform using user-submitted tasks and pairwise comparison between agents. We introduce BrowserArena, a live evaluation platform for evaluating LLM performance on user-submitted open-ended web agent tasks which builds off the Chatbot Arena [9] framework. In BrowserArena, users are requested to enter a task description, which is then submitted to two randomly-selected LLMs that utilize the BrowserUse library [20] to interact with and navigate different websites. BrowserArena uses a similar evaluation approach to other platforms for open-ended tasks, such as Chatbot Arena [8] and Copilot Arena [9]: pairwise comparisons between different agents to develop models for human preferences. This approach allows for the evaluation of tasks with ambiguous specifications and allows users to rank agent outputs according to criteria that may be difficult to evaluate in a ground-truth-based benchmark (such as whether the intermediate steps taken by the agent were reasonable).

Can VLMs model human preferences on agent performance? After collecting the user-submitted votes, we ask a new set of users to evaluate a subset of the original user-submitted tasks to measure the variance in user preference while evaluating the same agents on the same task. We observe that there is broad agreement with the original user-submitted preferences while taking a majority vote among the new users' submissions. However, despite previous work demonstrating multimodal-LLM-as-a-judge capabilities on evaluating pair comparisons on other image-based datasets [6], our experiments show that there is still a significant gap between human preferences and the preferences exhibited by vision-language models (VLMs).

Identification of agent failure modes through user-submitted step-level feedback: To overcome VLMs' limited capabilities for evaluating agents, we present an alternative methodology using human step-level feedback for identifying "failure modes" [4, 15], which are recurring situations across different tasks where users report that LLM agent behavior did not meet their expectations. Our approach is as follows: in our study, after a user submits a task on our evaluation platform, we ask the same user to annotate the steps produced in both agents' output traces to understand where the agent may have fallen short of user expectations. Intermediate steps in agent traces contain LLM-generated stepwise goals as well as descriptions of the actions taken during that step. We ask users to either mark the step's actions as correct with respect to its goal or mark it as incorrect and explain why it is contrary to their expectations of a successful step. This approach helps us collect more granular insights into agent behavior when compared to the simple voting mechanism present in prior work [9]. By analyzing user-submitted annotations, we identify three failure modes occurring within our system (captcha resolution, pop-up banner removal, and direct navigation to URLs). We then construct targeted datasets of tasks which reproduce these failure modes with a high frequency, and present our conclusions on the differences in language model behavior on these failure modes. We currently plan to open-source the BrowserArena platform codebase for collecting preference data to help identify new agent failure modes. We have open-sourced our codebase at `https://github.com/sagnikanupam/browserarena`.

Our key contributions are as follows:

1. We present an evaluation platform, BrowserArena, for pairwise comparison between models for user-submitted web-browsing tasks (Section 3).

2. We collect user preference data on 109 user-submitted tasks, using which we construct a language model leaderboard and demonstrate a gap in existing VLMs' ability to model human preferences (Section 4).

3. Given VLM preference labeling unreliability, we describe a new methodology for evaluating language model performance in web browsing by collecting step-level user annotations on agent traces and analyzing them to identify common failure modes, which are then studied separately (Section 5). We find, for example, that DeepSeek-R1 consistently misrepresents its ability to close pop-up banners, despite being unable to even identify such banners (due to its lack of multimodal capabilities).

2 Related Work

Question Answering Benchmarks: Several popular web agent benchmarks formulate their tasks as text or multimodal inputs to question-answering systems since they can be evaluated using reference ground truth strings. AssistantBench [27] presents a dataset of user-submitted domain-specific text-only QA tasks which only accept strings, numbers, and dictionaries as ground truth. WebQA [5] comprises of multi-image and complex single-image questions presented to the model alongside a set of positive sources and distractor sources. GAIA [16] presents a QA benchmark with more difficult tasks, several of which either require web browsing, code execution, and diverse filetype reading capabilities. BrowseComp [23] comprises of even harder QA tasks which take humans several hours of browsing to solve since the correct answers to the questions satisfy several constraints that are difficult to evaluate. While these benchmarks can evaluate agents' ability to search the web for information that may be very difficult to find or reason about data discovered via web search, they do not accurately represent how most human users would use these models for web navigation tasks on an everyday basis and does not measure several abilities valued by humans while navigating the web, such as navigating and taking actions on dynamic websites.

Self-Hosted and Simulated Benchmarks: Mind2Web [10] uses real-world webpage snapshots that include raw HTML code, DOM snapshots, and the network traffic for replaying an interaction, but formulates the web navigation task as an action selection or element selection task, restricting their measure of success to successfully replicating human-generated trajectories. Other approaches have formulated the web navigation problem as Partially-Observable Markov Decision Processes (POMDPs) with various reward mechanisms. For example, WebShop [25] introduced a simulated environment for executing search tasks defined in natural language on a shopping website containing products listed on Amazon, with agents only allowed to take click and search actions with ground truth rewards based on product attributes. WebArena [29] introduced a benchmark for executing natural language tasks on four self-hosted clones of popular websites with a larger action set, using both ground-truth answers and LLM-guided fuzzy matching for evaluating agent success. WebArena has been extended for evaluating agents on visually-grounded tasks in VisualWebArena [14], on tasks involving learning from long-context video understanding in VideoWebArena [12], and on complex tasks requiring mathematical reasoning and memory in WebChoreArena [19] using similar evaluation procedures. However, these benchmarks assign rewards based on the final output produced by the trajectory, making it difficult to assess if the intermediate steps taken by the agent would be considered reasonable by humans (as they assign equal rewards to two agents even if they take different approaches to reaching the same terminal state). They also do not provide methods for evaluating partial progress on tasks that are grounded in human preferences, instead relying on fuzzy matching to reward models whose outputs resemble the ground truth at the end of their trajectory.

Open Web Benchmarks: Certain popular benchmarks have adapted their evaluation methodology for evaluating web agents that can browse on the open web. WebVoyager [11] introduces a benchmark comprising tasks from 15 websites, omitting websites requiring CAPTCHA or login, developing tasks by sampling and rewriting tasks from Mind2Web [10] and prompting LLMs to generate new tasks. However, they then have to annotate tasks with sets of possible answers, with only 22.3% of tasks having "golden" answers that they expect not to change in the short term. MMInA [28] converts tasks from the WebQA dataset [5] into multimodal multi-hop problems, annotating them with instructions, examples of other QA tasks, and a "universe" of websites the model is allowed to visit while solving the tasks. While single-hop tasks are evaluated using ground-truth and fuzzy-matching based evaluations, multi-hop tasks are evaluated by marking tasks as completed only if each hop was completed correctly (by either visiting the correct link or by collecting the desired information). Thus, despite evaluating on dynamic, changing websites, the benchmark is restricted to evaluating tasks with respect to either ground-truth information or human-defined trajectories, making it difficult to scale the benchmark construction methodology to new tasks and websites (especially given the benchmark's dependence on dynamic web content not breaking the ground-truth human trajectories). SearchArena [18] is an extension of ChatbotArena's user-preference guided leaderboard system that allows for users to evaluate tasks on two randomly selected LLMs augmented with search capabilities. However, their framework is restricted to web search tasks for retrieving and summarizing information, and is unable to evaluate web agent behavior on browser-based tasks involving taking actions on websites. Additionally, SearchArena does not provide agent traces describing the sequence of websites visited and actions taken on each website, making it difficult to compare partial progress on each task and analyze step-level feedback.

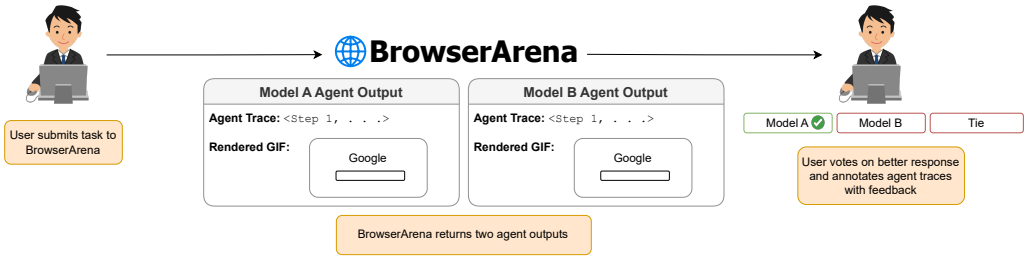


Figure 1: An overview of the study procedure showing how users interact with BrowserArena. We include examples of user submitted tasks in Appendix M.

3 BrowserArena Evaluation Platform

We develop the BrowserArena website by equipping ChatBot Arena's open-source codebase [9] with the capabilty of submitting a task to BrowserUse [20] and visualizing the results. On visiting the website, users are presented with a text box in which to enter a description of the task (examples of user-submitted tasks are in Appendix M). Once the user submits their task, two LLMs are chosen at random with uniform probability for creating the BrowserUse agents. These models are then used to construct BrowserUse agents, which utilize independent Playwright [17] instances for automating a Chromium browser. The LLM is permitted to choose an action from the set of actions pre-defined by the BrowserUse controller (for a full list, see Table 1 in Appendix C). The BrowserUse agents accept the task, previous steps, current URL, open tabs, and a list of HTML elements with associated numeric indices, where the indices of interactive elements are distinguished from the other elements. If the model has multimodal capabilities (all our tested models except DeepSeek-R1), it also receives a screenshot of the current browser with an overlay labelling the rendered HTML elements with their indices. The LLMs then output a JSON object describing the current state of the task, containing four properties: a self-evaluation of whether the previous goal was completed, a memory property describing what has been done so far, a goal property describing the next immediate objective, and a sequence of actions to take.

The user-submitted task prompt is then submitted to the two BrowserUse agents, each using one of the sampled LLMs as the model backend. Once both models finish, we present the user with the agent outputs of the models, as well as a GIF rendering each step that the agent took on the Playwright Chromium browser instance. Once these agent outputs are rendered on the website, users are provided with an option to vote on which response is better.

4 Experimental Evaluation

For collecting tasks on BrowserArena, we first design a user study (details described in Section 4.1) asking users to submit tasks, vote for the agent that best completed the task, and annotate the generated agent traces. Then, using user votes, we construct a leaderboard of models . We present our results in Section 4.2. Then, we run a study to measure human evaluator agreement on a subset of the user-submitted tasks (detailed in Section 4.3), and demonstrate a significant gap between VLM preferences and human preferences based on agent outputs in Section 4.4.

4.1 User Study Design

For our experimental study, we solicit tasks and feedback on agent performance via a survey on Prolific. We recruit users on Prolific from United Kingdom, United States, Australia, Canada, and New Zealand with response approval rates between 90–100%. We approved a total of 213 valid responses, ultimately keeping 109 responses from 98 users due to system outages, logging issues, and invalid responses. We collected responses in 3 batches, with the average of the batch median completion times being 35:10 minutes, and payments being made at an average hourly rate of $8.01/hr.

We provide further details about the each batch's compensation and median completion times in Appendix H.

We ask Prolific users to submit tasks that involve clicking and interacting with different websites (which we call "interactive tasks") and to explicitly avoid submitting tasks which either can be answered by analyzing Google search result links and descriptions or can be answered by a language-model chatbot without searching for an answer. We also caution users not to enter tasks where the answer is easily provided within the Google search results without clicking on a website or is an open-ended question that a chatbot can answer without clicking on any website (which we call "search tasks"). We provide some examples to help users differentiate between the two, which we have listed in Appendix A.

Since the goal of each step is LLM-defined, we ask users to use the agent traces and the generated GIFs to identify steps that were executed correctly with respect to their goals, and describe where "incorrect" steps fell short. With the help of user feedback, we analyze the agent traces and construct a mapping between each step generated by the agent and whether it was perceived to be successfully executed. This information is then used to identify the failure modes explored in our case study in Section 5. Finally, users are asked to vote between the two LLM models. Unlike the original ChatBot Arena website, we only accept "Left", "Right", and "Tie" votes and ignore "Both models are bad" votes, since we are interested in measuring partial progress if both agents fail.

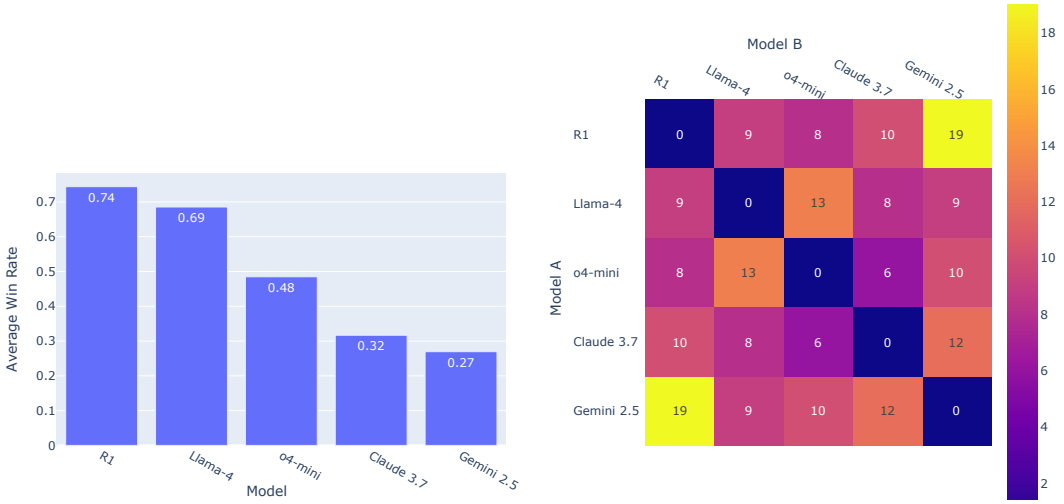
We then utilize the user votes to construct a model leaderboard of voter preferences. We evaluate the performance of five models on the BrowserArena platform: DeepSeek **R1**, Anthropic **Claude 3.7** Sonnet:Thinking, Meta **Llama-4**-Maverick, OpenAI **o4-mini**, and Google **Gemini 2.5**-Pro-Preview-03-25 using the OpenRouter API platform. In our subsequent discussions, we will refer to each of these models by the bolded portion of their names. We note that while the BrowserUse platform supports submitting image screenshots of the webpage alongside search results and web page structure in API calls to the model, R1, being a language model without multimodal capabilities does not utilize the image screenshot provided.

4.2 Ranking Results

By estimating the Bradley-Terry coefficients of each model [2] based on the user votes, we compute the ranks of different models using the ranking methodology described in Chatbot Arena [9]. We provide a more detailed summary of leaderboard construction in Appendix B. We present our leaderboard from 109 valid battles alongside our win fraction heatmap, average win rate bar, confidence interval calculations, and a heatmap of the battlecounts in Figure 2. Based on the user-submitted tasks, the LLM agent with the highest ELO rating is based on R1, which surprisingly is the only model evaluated that does not have multimodal capabilities.

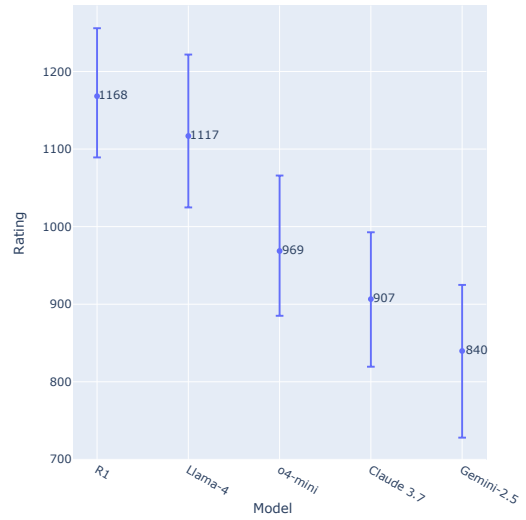
4.3 Human Evaluator Agreement

We evaluate how consistently humans judge head-to-head browser-agent runs on 25 randomly selected task submissions, and find modest-to-strong agreement. For each task, annotators are shown the same agent trace and GIF comparison used for the original task submissions, and are asked to select between Agent 1, Agent 2, and Tie. 165 new human annotations are collected from Prolific; we use two screening questions and participants take on-average 57 seconds to provide a selection on a task. We compare these human aggregates to the label from the original task submission with inter-annotator agreement, which measures how often different human evaluators make the same choice when comparing two agent trajectories, with higher agreement indicating clearer differences in performance between the agents. We find that the majority vote of the new human annotators has modest agreement with the baseline labels (63.2% of questions) and modest inter-annotator agreement (57.6%). Lower agreement is largely explained by the lack of consistency between labelers when voting 'tie'; the majority vote agreement goes up to 100% agreement when 'tie' votes are removed and we force a majority selection between Agent 1 and 2. Similarly, the inter-annotator agreement goes up to 83% when ties are filtered. These results suggest that differences between human agent judgements reflect differing decision thresholds more than differing rank orderings of agents.

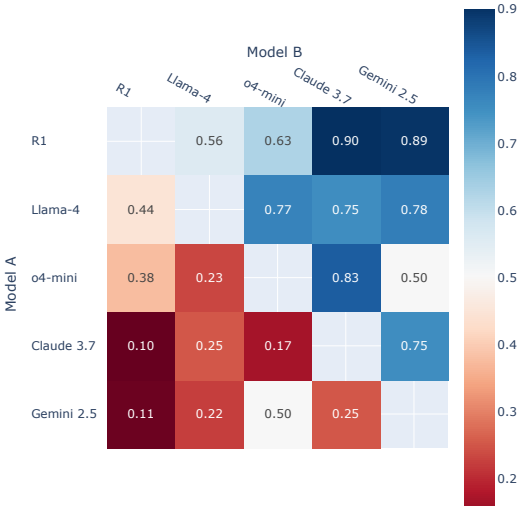


(a) Average Win Rate across models

(b) Battle Count Heatmap



(c) Bootstrapped ELO Rating



(d) Pairwise Win Fraction Heatmap

Figure 2: We compute the average win rate, battle counts, bootstrapped ELO ratings, and pairwise win fractions from 109 user-submitted tasks and evaluations on Prolific. For the ELO-based leaderboard, we simply sort the models from highest to lowest bootstrapped ELO rating in Figure 2(c).

4.4 VLM-as-a-Judge

For VLM evaluation, we use the same 25 randomly selected task submissions we use for measuring human evaluator agreement. The original human task labels are compared to two vision-language model judges (GPT-4o, o4-mini) that are prompted with the same input (the agent trace and GIFs) and asked to choose between select between Agent 1, Agent 2, and Tie. As shown in Figure 3, GPT-4o has relatively high agreement with the human annotation baseline (68%), o4-mini only 58%. Interestingly, we find that the GIFs showing the agent computer seem to be hurt GPT-4o agreement: in input ablations, trace-only evaluation improves GPT-4o's agreement with the baseline annotations by 10 percentage points (79% vs. 68% with GIFs and traces), while GIF-only input collapses performance to 48% agreement despite an increased self-reported confidence. These results indicate that multimodality can hurt judge reliability in this setting. In summary, we find a sizable gap in labeler agreement between VLMs and humans.

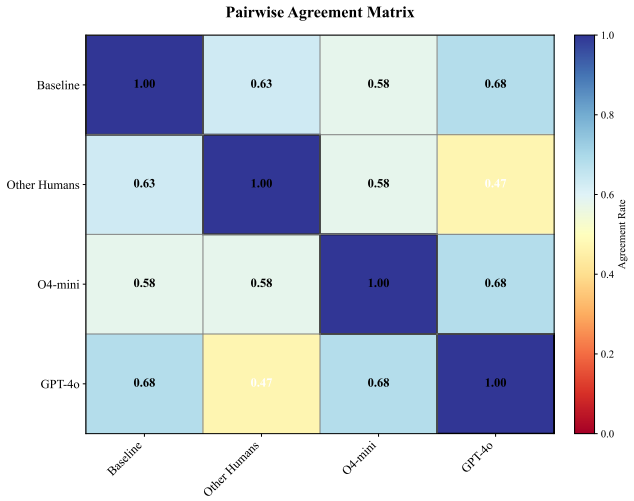


Figure 3: Pairwise agreements between the baseline labels, the new annotators, and two vision-languages models (GPT-4o and o4-mini; we take the majority @5).

5 Prominent Agent Failure Modes

We use the agent traces and human feedback collected in our benchmark to surface and study three prominent failure modes in current agents. After collecting the step-level feedback as a part of our initial Prolific user study, we cluster and summarize the step-level annotations as described in Section 5.1. Using these clusters, we identify three failure modes where agents fail to complete tasks which we investigate in greater detail: Captcha Solving (Section 5.2), Pop-Up Banner Closure (Section 5.3), and Direct Navigation (Section 5.4).

To study variations in model behavior on occurrence of each of these failure modes, we use the following general pipeline. We first construct a new larger dataset of tasks which reproduce the failure mode scenario with a high probability when an agent attempts to complete the task. Then, once we execute these tasks for each language model, we use o4-mini as a judge to evaluate the traces generated by the agent and determine if the specific failure mode occurred while the agent was executing the task. We then report aggregate statistics on how often each language model ran into specific scenarios while executing the tasks.

5.1 Discovering Common Failure Modes

We use our step-level human labels on the BrowserArena agent tasks to automatically find 'failure modes' [15, 4], consistent mistakes an agent makes while performing the user-submitted tasks. Three of our discovered failure modes are explored in detail in Sections 5.2-5.4. To automatically find these common failure modes, we use two methods (dataset featurization [3] and an API-only method, Docent [15]) that first cluster the step-level labels in an embedding feature space, and then use auxiliary LLMs to summarize these clusters; these cluster summaries, which pick out consistent agent behavior across tasks, are the failure modes. The methods find very similar failure modes; we give the full set of discovered failure modes found via dataset featurization in Table 2 and Docent in Figure 4(a). Our cluster and summarization hyperparameters are described in Appendix I.

We then select the following three failure modes for a more detailed investigation from the list of failure modes we have constructed:

1. **Captcha Solving:** On encountering a captcha puzzle, agents can get stuck while attempting to solve the puzzle since the individual components of the puzzle may not be clickable elements in the webpage's DOM. We thus seek to study the different strategies used by different language models to evaluate if specific models prefer different captcha-avoidance methods to others.

2. **Pop-Up Banner Closure:** On encountering a pop-up banner obscuring a part of the website, agents can be preventing from making progress on the remainder of the task due to being unable to close the pop-up-banner. We thus study how often a language model identifies that a pop-up banner is blocking its access to the website and successfully closes the banner and moves ahead with its task.
3. **Direct Navigation**: Sometimes, agents choose to directly navigate to a website URL (hereby referred to as the starting website) that they believe is integral for solving the task as opposed to conducting a Google Search to collect relevant links first. This can lead to delays in completing the task if navigating the starting website is more complex for the agent compared to the websites which may have been selected had the model conducted a Google search first.

5.2 Captcha Solving

Dataset Construction: We first identify `www.expedia.com` as a website that is reliably blocked by a captcha on our system when an agent attempts to visit it while solving a user-submitted task. We then construct a dataset of 220 tasks which require interacting with or visiting the Expedia website. 20 of these tasks are constructed from human written task templates, and 200 of them are generated by GPT 4.1 using a task generation prompt (for template and prompt details, see Appendix D).

Scenarios: We first construct a set of captcha circumvention strategies by manually examining LLM agent traces produced by different models on some of the 20 template-based tasks. We additionally have an LLM (o4-mini) also analyze all of these traces and identify if any other strategies have been used for captcha navigation in these traces. We add the new strategies detected by LLM to our existing set of strategies. Finally, we use o4-mini to identify if any strategy from our strategy set was used in the agent traces of each of the 220 tasks that each LLM attempted to solve. For the detailed prompt used for o4-mini to judge all the agent traces and the description of each strategy provided in the prompt, see Appendix F.

Results: We present our results measuring the percentage of times each particular strategy was deployed by a model in Table 3 in Appendix J. We observe that most language models show a clear preference for the "Direct Link", "Google Search", and "New Tab" strategies. However, Claude 3.7 prefers the Switch Websites method much more than other LLMs, while both it and Gemini-2.5-Pro use the "New Tab" tactic less than other LLMs (and in fact prefer the "Switch Websites" method to it). On the other hand, o4-mini uses all the listed strategies at least once, and uses some strategies not used at all by other language models, such as the "Text-only Rendering", "Public Proxy", and "Internet Archive". It also uses tactics such as "Cache", "Mobile", and "Internal Navigation" and "Country Domain" at much higher rates than other LLMs, suggesting that it is better at getting around captcha challenges in the event of their presence disrupting the search than other language models as it is able to try a wider range of strategies.

5.3 Pop-up Banner Closure

Dataset Construction: We first identify `www.bbc.com` as a website that reliably generates a privacy policy banner when an agent attempts to visit it while solving a user-submitted task. We construct a dataset of 80 tasks which require interacting with or visiting the BBC website by prompting GPT 4.1 using a task generation prompt (for template and prompt details, see Appendix E).

Scenarios: We consider three scenarios that the LLM agent may find itself in: either it did not detect a pop-up banner while evaluating the task, it did discover a pop-up banner and successfully closed it, and it marked the task as being completed (independent of whether it managed to progress past the pop-up banner). We then use o4-mini to identify if any of these scenarios occurred in the agent traces of each of the 80 tasks that each LLM attempted to solve. For the detailed prompt used for o4-mini to judge all the agent traces and the description of each strategy provided in the prompt, see Appendix G. For complete results of how frequently each scenario occurs for specific LLMs, please refer to Table 4 in Appendix K.

Results: Notably, R1 seems to have never realized that a part of the website is blocked by a privacy policy pop-up in all the times it attempts to complete the BBC agent tasks, indicating that multi-modal reasoning ability is required for detecting the privacy policy pop-up. However, R1 marks the task as completed at the highest rate of all the LLMs, suggesting that without multimodal capabilities, it is

unable to reason that its task remains incomplete without closing the cookie banner. On the other hand, o4-mini and Llama-4 manage to close pop-up banners at a higher rate than the remaining multimodal LLMs, although only o4-mini marks a similar percentage of tasks as completed as compared to the percentage of tasks for which the LLM judge determines that it closed the pop-up banner.

5.4 Direct Navigation

Dataset Construction: We focus on a knowledge-intensive question answering task to investigate whether agents opt to directly answer, directly navigate to relevant websites, such as Wikipedia, or instead invoke the Google Search API. To this end, we sample 100 questions from the TriviaQA dataset [13], which comprises naturally occurring questions posed by trivia enthusiasts.

Scenarios: We consider two distinct scenarios that the language model (LLM) agents may encounter: (1) the agent recognizes the question and directly answers or navigates to the corresponding Wikipedia page; or (2) the agent lacks sufficient knowledge and first queries the web using Google Search. For each question, we collect the agent's execution trajectory and manually annotate the scenario it conforms to. A summary of the distribution of scenarios across models is provided in Table 5 in Appendix L.

Results: We observe that the most frequent behavior involves invoking the Google Search API to retrieve relevant information using extracted keywords. In some instances—more commonly observed with Llama-4—the agent navigates to Google.com and inputs search queries manually, rather than using the API. In contrast, direct answering or navigation to Wikipedia pages is relatively rare. These findings suggest that, in general, agents tend to follow the instruction and leverage Google as the primary information source when responding to knowledge-intensive queries.

6 Conclusions

In this paper, we have presented a web agent evaluation platform, BrowserArena, for pairwise comparison between various language models on user-submitted web browsing tasks. After collecting user preference data on 109 user-submitted tasks, we first construct a language model leaderboard to demonstrate user preferences between various models. Then, we demonstrate a gap between VLM agreement and human evaluator agreement on user preferences.

This gap motivates our development of a new methodology for evaluating language model performance by collecting step-level user annotations on agent traces and analyzing them to identify common failure modes. We then provide methods to construct three targeted datasets to further study these failure modes, and report our results on differences in model behavior when encountering these failure modes.

7 Limitations

Our approach for standardizing language model agents involves equipping models with BrowserUse [20], which provides all models with a standard format in which to output their goals and the action to be taken in each step. However, equipping models with different or more powerful capabilities may help improve agent capabilities in solving tasks, which makes our results and evaluation method dependent on the browser agent system connected to the LLM.

Additionally, another drawback is that the failure modes we discover may be system specific. We believe that it is still useful to identify failure modes and construct targeted datasets to analyze model behavior under similar circumstances. However, the specific tasks that trigger the failure mode may be different depending on the system configuration - for example, it may be possible to reduce the likelihood of encountering captchas on a particular website by using rotating proxies.

References

[1] Bonatti, R., Zhao, D., Bonacci, F., Dupont, D., Abdali, S., Li, Y., Lu, Y., Wagle, J., Koishida, K., Bucker, A., et al. (2024). Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*.

[2] Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

[3] Bravansky, M., Kubon, V., Hariharan, S., and Kirk, R. (2025). Dataset featurization: Uncovering natural language features through unsupervised data reconstruction. *arXiv preprint arXiv:2502.17541*.

[4] Brown, D., Balehannina, P., Jin, H., Havaldar, S., Hassani, H., and Wong, E. (2025). Adaptively profiling models with task elicitation. In *The 2025 Conference on Empirical Methods in Natural Language Processing*.

[5] Chang, Y., Narang, M., Suzuki, H., Cao, G., Gao, J., and Bisk, Y. (2022). Webqa: Multihop and multimodal qa. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16495–16504.

[6] Chen, D., Chen, R., Zhang, S., Wang, Y., Liu, Y., Zhou, H., Zhang, Q., Wan, Y., Zhou, P., and Sun, L. (2024). Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark. In *Forty-first International Conference on Machine Learning*.

[7] Chezelles, D., Le Sellier, T., Gasse, M., Lacoste, A., Drouin, A., Caccia, M., Boisvert, L., Thakkar, M., Marty, T., Assouel, R., et al. (2024). The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*.

[8] Chi, W., Chen, V., Angelopoulos, A. N., Chiang, W.-L., Mittal, A., Jain, N., Zhang, T., Stoica, I., Donahue, C., and Talwalkar, A. (2025). Copilot arena: A platform for code llm evaluation in the wild. *arXiv preprint arXiv:2502.09328*.

[9] Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhu, B., Zhang, H., Jordan, M., Gonzalez, J. E., et al. (2024). Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*.

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

[11] He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., and Yu, D. (2024). Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

[12] Jang, L., Li, Y., Zhao, D., Ding, C., Lin, J., Liang, P. P., Bonatti, R., and Koishida, K. (2024). Videowebarena: Evaluating long context multimodal agents with video understanding web tasks. *arXiv preprint arXiv:2410.19100*.

[13] Joshi, M., Choi, E., Weld, D., and Zettlemoyer, L. (2017). TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Barzilay, R. and Kan, M.-Y., editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

[14] Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M. C., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., and Fried, D. (2024). Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

[15] Meng, K., Huang, V., Steinhardt, J., and Schwettmann, S. (2025). Introducing docent. `https://transluce.org/introducing-docent`.

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

[17] Microsoft (2025). Playwright.

[18] Miroyan, M., Wu, T.-H., King, L., Li, T., Pan, J., Hu, X., Chiang, W.-L., Angelopoulos, A. N., Darrell, T., Norouzi, N., and Gonzalez, J. E. (2025). Search arena: Analyzing search-augmented llms.

[19] Miyai, A., Zhao, Z., Egashira, K., Sato, A., Sunada, T., Onohara, S., Yamanishi, H., Toyooka, M., Nishina, K., Maeda, R., et al. (2025). Webchorearena: Evaluating web browsing agents on realistic tedious web tasks. *arXiv preprint arXiv:2506.01952*.

[20] Müller, M. and Žunič, G. (2024). Browser use: Enable ai to control your browser.

[21] OpenAI (2025). Introducing operator. Research preview announcement.

[22] Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., et al. (2024). Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

[23] Wei, J., Sun, Z., Papay, S., McKinney, S., Han, J., Fulford, I., Chung, H. W., Passos, A. T., Fedus, W., and Glaese, A. (2025). Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

[24] Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. (2024). Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

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

[26] Yehudai, A., Eden, L., Li, A., Uziel, G., Zhao, Y., Bar-Haim, R., Cohan, A., and Shmueli-Scheuer, M. (2025). Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

[27] Yoran, O., Amouyal, S. J., Malaviya, C., Bogin, B., Press, O., and Berant, J. (2024). Assistantbench: Can web agents solve realistic and time-consuming tasks? *arXiv preprint arXiv:2407.15711*.

[28] Zhang, Z., Tian, S., Chen, L., and Liu, Z. (2024). Mmina: Benchmarking multihop multimodal internet agents. *arXiv preprint arXiv:2404.09992*.

[29] Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. (2023). Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Example Tasks Presented to Users

Examples of Valid Interactive Tasks:

1. What are today's top 20 headlines from CNN?
2. Compare the bus prices for one-way tickets from Boston to New York next Saturday on different ticket purchasing websites.
3. Create a list of the top-ranked chess players on chess.com from Belgium.

Examples of Invalid Search Tasks *(Alongside Why They are Invalid)*

1. How do I increase my concentration while working? *(This is invalid because it can be answered using a chatbot and does not require clicking on a specific website.)*
2. What is the weather today? *(Google will output this answer in a box displayed at the top of search results, again does not require clicking on a specific website.)*
3. Who are the members of the Beatles? *(Google provides a lot of links with the text containing the answer to this question under those links, so you do not need to click on a website to answer this question.)*

B Ranking Methodology

We use a similar approach to other pairwise-comparison evaluation procedures for ranking models. Here, we present an overview of the procedure in the binary preference case for M models. As defined in [8, 9], in a sequential setting, at time $t \in \mathbb{N}$, we first formally define our comparative data set $\mathcal{A} = \{(m, m') : m < m' \text{ and } m, m' \in [M]\}$. Then, for a pair of models $A_t = (i, j) \in \mathcal{A}$, we model the human preference $H_t \in \{0, 1\}$, where H_t is 1 if i is preferred over j and 0 if j is preferred over i. We then define the score function to be the vector of Bradley-Terry coefficients $\beta \in \mathbb{R}^M$ [2]. Under the Bradley-Terry model, the probability of model i beating model j i.e. $\mathbb{P}(H_t = 1)$ is given as shown:

$$\mathbb{P}(H_t = 1) = \frac{e^{\beta_i}}{e^{\beta_i} + e^{\beta_j}} \tag{1}$$

The rank of a model m is then calculated as follows:

$$\text{rank}(\beta)_m = 1 + \sum_{m' \in [M]} \mathbb{1}\{\beta_{m'} > \beta_m\} \tag{2}$$

The BT coefficients are then estimated via maximum likelihood estimation, with 95% confidence intervals being calculated by bootstrapping for 100 rounds. After determining the confidence interval, the rank of each model is estimated by computing the number of models whose lower bound is less than its upper bound [8]. This model can then be extended to cases when H_t is not binary by estimating the BT score from a nonparametric extension of the Bradley-Terry model [9].

C BrowserUse Permitted Actions

Action Name	Action Description
Complete Task	Mark task as completed with success=True if successfully completed and success=False if at last step.
Search Google	Search the query in Google on the current tab.
Go to URL	Visit the specified URL in the current tab.
Go Back	Go back in history to the previous website visited.
Wait	Wait for x seconds where $x = 3$ by default
Wait for element to be visible	Wait for an element specified by the CSS Selector to become visible within the specified timeout.
Click element by Index	Click the HTML element specified by its numeric index
Click element by Selector	Click the HTML element specified by its CSS Selector.
Click element by XPath	Click the HTML element specified by its XPath path expression.
Click element with Text	Click the HTML element containing the provided text.
Input Text	Input the provided text into the specified input interactive element.
Save as PDF	Save the current page as a PDF file.
Switch Tab	Switch to a different browser tab.
Open URL in New Tab	Open the specified URL in a new tab.
Close Tab	Close the specified browser tab.
Extract Page Content	Extract page content using an LLM prompted with the specified goal.
Save as HTML	Save the raw HTML content of current page as an HTML file.
Scroll Down	Scroll down by a specified pixel amount, by default scroll down one page.
Scroll Up	Scroll up by a specified pixel amount, by default scroll up one page.
Send Special Keys	Send special key commands (Esc, Backspace, keyboard shortcuts) to the current page.
Scroll to Text	Scroll until the specified text is visible on the current page.
Get Dropdown Options	Get all options from a dropdown element.
Select Dropdown Option by Text	Select dropdown option using the specified text
Drag and Drop	Drag and drop elements, or between coordinates of the page.

Table 1: BrowserUse actions available to LLMs, as specified in [20].

D Prompt Used for LLM-Generated Captcha Solving Tasks

Note: While we prompt the language model to generate a 1000 tasks in most of our prompts, we usually get between 80-100 tasks in a single response. For generating our final list tasks, we prompt the model repeatedly and remove any duplicate tasks generated.

Model Used: GPT 4.1

System Prompt: You are a helpful task-generating agent.

User Prompt: In JSON format, generate a list of 1000 tasks for a web agent that involve navigating to the Expedia website and searching for information. Here are some examples of the desired tasks:

1. "Find a list of activities on Expedia to do in Chicago on December 3."
2. "Find the cheapest hotel in Houston on Expedia from August 8-12."
3. "Find the cheapest round-trip flights between Houston and Los Angeles on Expedia from November 10-15."

Make sure to randomize the cities, dates, and types of tasks. Do not generate any tasks that require the agent to take any action on the website except for searching and retrieving information. Make sure the tasks include the phrase "on Expedia" so that the agent searches for the Expedia website.

E Prompt Used for LLM-Generated Pop-Up Banner Tasks

Model Used: GPT 4.1

System Prompt: You are a helpful task-generating agent.

User Prompt: In JSON format, generate a list of 1000 tasks for a web agent that involve navigating to the bbc.com website and searching for information. Here are some examples of the desired tasks:

1. "Find the top 10 headlines of the day on bbc.com."
2. "What is the top story today in the Culture section on bbc.com."
3. "List all the headlines from three days ago in the innovation section on bbc.com."

Make sure to randomize the number of days and the sections of the bbc.com website in the task. You must only choose sections from the following list: News, Sport, Business, Innovation, Culture, Arts, Travel, Earth, Audio, Video. Do not generate any tasks that require the agent to take any action on the website except for searching and retrieving information. Make sure the tasks include the phrase "on bbc.com" so that the agent searches for the bbc.com website.

F Prompt Used for Evaluating Captcha Analysis Agent Traces

Model Used: o4-mini

System Prompt: You are an LLM judge evaluating the output of an agent attempting to traverse different websites to solve a task. The task involves navigating Expedia, a travel website which often uses captchas to block robots from traversing the website. Your goal is to identify the captcha avoidance strategies used by the agent. The list of captcha avoidance strategies are as follows:

1. `cache` - attempts to use Google's cache to retrieve an unblocked version of the website.
2. `mobile` - attempts to use the mobile version of the website to retrieve an unblocked version of the website.
3. `direct_link` - attempts to open the website by navigating directly to a link that may have the correct website instead of searching for it on Google.
4. `google_search - attempts to conduct a Google search to identify alternative links to the same website (without using any cache terms - if the Google search has cache terms, then the `cache` strategy was used).
5. `randomized_interaction` - attempts to wait random amounts of time before completing an interaction to circumvent bot detection algorithms.
6. `reloads` - reloads the website in an attempt to remove the captcha.
7. `new_tab` - attempts to open the website in a new tab to avoid any session cookies being associated with its search.
8. `switch_websites` - switches to a non-Expedia website to solve the task instead of trying to navigate to Expedia.
9. 'internal_navigation' - attempts to go to the home webpage of Expedia, and navigate to the correct webpage from the home webpage.
10. `country_domain` - attempts to use a different country domain of Expedia to retrieve an unblocked version of the website.
11. `text-only rendering` - attempts to perform a text-only render or retrieve the plaintext version of the website by using a proxy such as Textise.
12. `public proxy` - attempts to use a public proxy such as AllOrigins to avoid bot protection mechanisms.
13. `internet_archive` - attempts to use the Internet Archive's Wayback Machine to retrieve an unblocked version of the website.
14. `google_travel_integration` - attempts to use the Google Travel integration to solve the task. This strategy is only applicable if the term "Google Flight Integration" or "Google Travel Integration" is mentioned in the agent's output.

You should output a JSON string in the format `strategy_name`: `bool` which has all the keys as strategy_names and the values as "true" or "false" depending on whether the strategy was detected in the agent output. For example, if only the `reloads` and `new_tab` strategies were detected in the output, your output should be:

```


{
"cache": false,
"mobile": false,
"direct_link": false,
```

"google_search": false,
"randomized_interaction": false,
"reloads": true,
"new_tab": true,
"switch_websites": false,
"internal_navigation" false,
"country_domain": false,
"text-only rendering": false,
"public_proxy": false,
"internet_archive": false,
"google_travel_integration": false
}
```

Make sure that all the keys are present in the output, even if their value is false. Only output the JSON string, do not include any other text, and make sure that the only keys in the JSON string are the ones listed above. Do not include any additional keys or values in the output.

**User Prompt:** `{Agent Trace}`

# G Prompt Used for Evaluating Pop-Up Banner Agent Traces

**Model Used:** o4-mini

**System Prompt:** You are an LLM judge evaluating the output of an agent attempting to traverse different websites to solve a task. The task involves navigating to a website with a cookie banner, privacy pop-up, or pop-up ad and completing the assigned task. You are provided with the task and the agent's output, and your goal is to identify if the agent was successful in closing the cookie banner or pop-up ad and progressing with the task.

You should output a JSON string in the format {"banner_detected": `bool`, "banner_closed": `bool`, "task_successfully_completed": `bool`} where the "banner_detected" key indicates whether the agent found a cookie banner, privacy pop-up, or pop-up ad at any point while trying to complete the task, and the "banner_closed" key indicates whether the agent successfully closed it. If no cookie banner or pop-up ad was detected, both values should be false. The "task_successfully_completed" key should be set to True if the agent states it successfully completed the task at the end of the trace.

Make sure that all the keys are present in the output, even if their value is false. Only output the JSON string, do not include any other text, and make sure that the only keys in the JSON string are the ones listed above. Do not include any additional keys or values in the output.

**User Prompt:** `{Agent Trace}`

# H Prolific User Study Compensation

We collect tasks from users in 3 batches: the 6 participants in the first batch of the pilot study were paid $1.50 per response based on a projected median response time of 11:00 minutes for an hourly cost of $8.17/hr. The second and third batches were paid at an hourly rate of $8.01/hr, with the 28 responses in the second batch paid at a rate of $5.40 per response based on the calculated median response time of 40:28 minutes, while the 179 responses in the third batch were paid at a rate of $4.69 per response based on the calculated median response time of 35:09 minutes.

# I Failure mode discovery details

## I.1 Dataset Featurization

We apply *Dataset Featurization* [3] to surface common failure modes from our step-level agent task labels, following the unsupervised, two-stage pipeline of (i) feature proposal via contrastive data–reconstruction prompts and (ii) forward selection under a reconstruction–perplexity objective. Concretely, for each target goal string $x$, we draw $C{=}5$ contrastive strings $\{r_c\}_{c=1}^{5}$ from the corpus
```

and prompt GPT-4o to propose $K{=}4$ short ($\leq$20 words) binary predicates that are true of x while (ideally) not holding for the $\{r_c\}$. This contrastive step forces candidates to be discriminative rather than generic. Pooling across $N{=}218$ goal–feedback examples yields 872 initial feature hypotheses. We embed each candidate (and associated step text) with `text-embedding-3-small`, standardize embeddings, and perform K-means with target granularities chosen to achieve interpretable coverage yielding $15, 10, 5$ clusters across sweeps. From each cluster we retain one representative phrasing. We then assign binary truth values by asking GPT-4o (temperature $= 0$) to evaluate every (goal string, clustered feature) pair, producing a $N \times K'$ binary matrix (labels "Y/N").

The final failure modes are selected from these clusters by testing how well they allow a language model model (Llama-3-8B) to reconstruct the step-by-step labels. Namely, we treat active features for a text as a newline-delimited context and compute mean per-text perplexity

$$\mathrm{PPL}(D \mid \phi) \;=\; \frac{1}{N}\sum_{n=1}^{N} \mathrm{PPL}\big(x^{(n)} \,\big|\, \texttt{ctx}(\phi(x^{(n)}))\big),$$

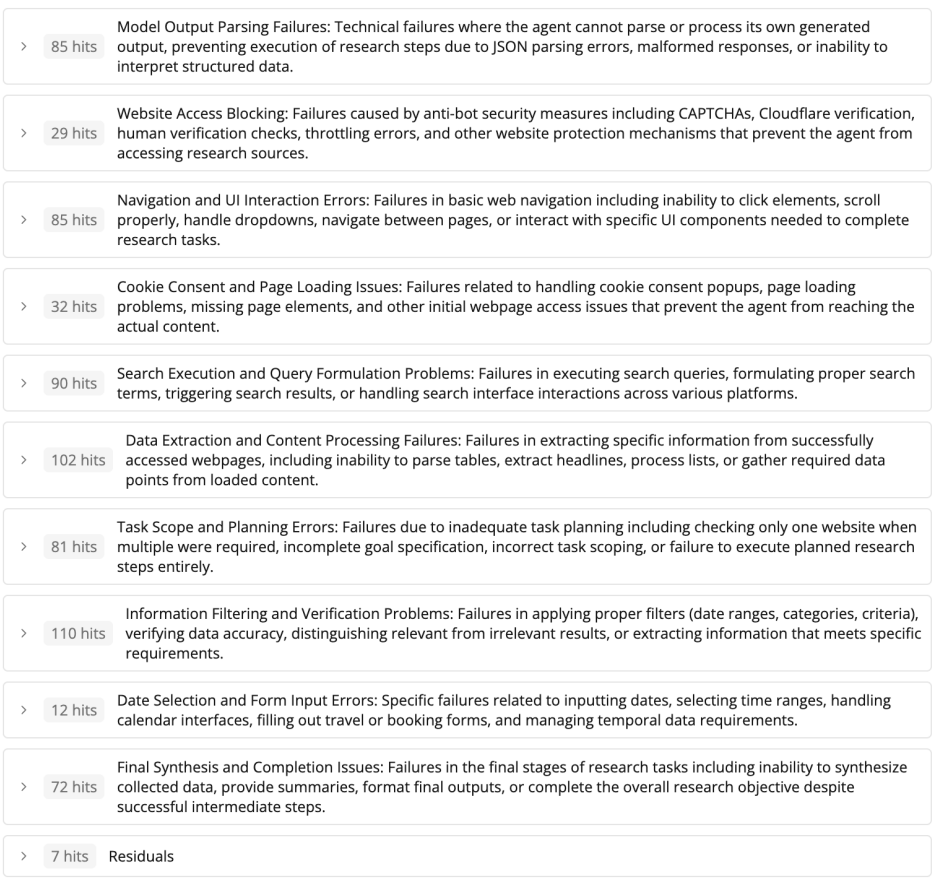
then greedily append the feature F that most reduces perplexity, i.e.,

$$F \;=\; \arg\min_{F'} \; \mathrm{PPL}\big(D \,\big|\, \phi \cup \{F'\}\big),$$

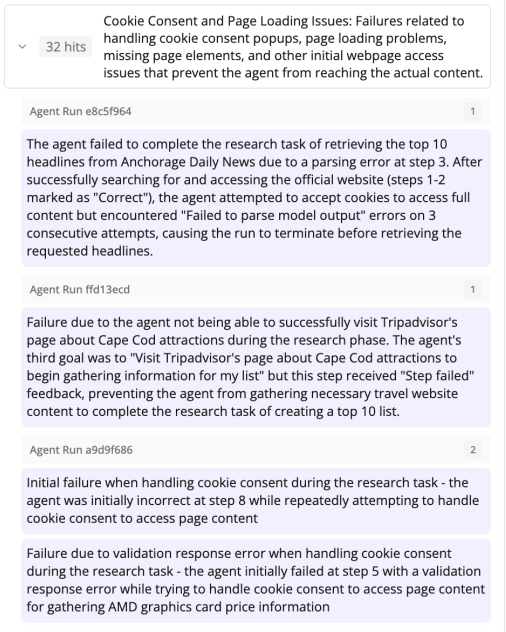
stopping when no candidate yields a further drop (or a feature budget is reached). Following DF, we use a static reconstruction prompt and cache log-probabilities for texts where a feature evaluates to FALSE to avoid redundant computation. The resulting cluster summaries instantiate the final failure modes.

I.2 Docent

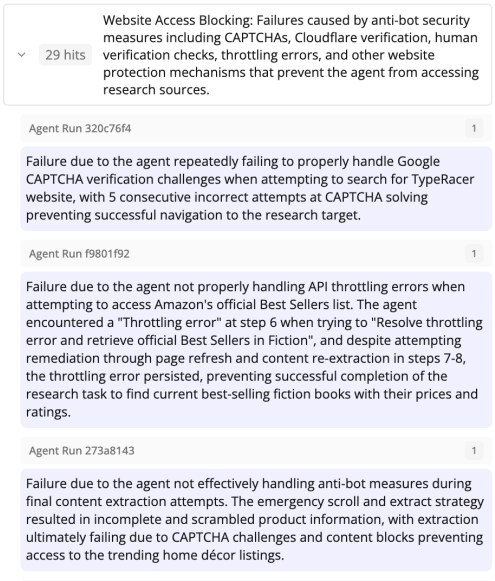
We also use an API-only method, Docent [15], to help confirm the consistency of our clusters and summaries across featurization methods. We pass the human-step level labels of each agent goal, along with the prompt: `Based on the step-by-step feedback metadata on each agent step, find the failure modes where the agent fails to complete research tasks. Be granular, e.g. not just "failure" but "failure due to the agent not properly handling x in case y.".` The failure modes are displayed in Figure 4(a); we find significant overlap between our dataset featurization failure modes and the Docent failure modes. To featurize the dataset, Docent uses Claude Sonnet 4 to produce natural language summaries of our human step-level labels, two of these (for the cookie and captcha failure modes) are presented in Figure 4(b) and (c).



(a) Docent summarization of our human step-level labels



(b) Example individual datapoint captioning (in blue), from the Claude Sonnet 4, for the failure mode dealing with cookies.



(c) Example individual datapoint captioning (in blue), from the Claude Sonnet 4, for the failure mode dealing with cookies.

Figure 4: Failure mode identification with Docent [15].

Table 2: Agent failure modes found form the human step-level labels via dataset featurization [3], under different granularity k. Bolded rows correspond to the failure modes we explore in detail via generated tasks in Section 5.

k	Failure mode	Count	Share (% of total 220)
$k = 5$			
	Complex tasks with multiple steps	185	84.9
	Navigation to specific website sections	116	53.2
	Straightforward task sequences	65	29.8
	Repeated parsing errors	54	24.8
	Task completion execution errors	22	10.1
	Cookie consent handling failures	12	5.5
$k = 10$			
	Specific list extraction tasks	148	67.9
	Direct URL navigation attempts	77	35.3
	Goal completion without failures	68	31.2
	Repeated parsing errors	63	28.9
	Concise task structure	58	26.6
	High frequency unsuccessful attempts	57	26.1
	Technical errors (non-navigation)	44	20.2
	Product category focus	37	17.0
	Cookie consent success	20	9.2
	Inadequate human feedback	8	3.7
$k = 15$			
	Navigation to specific sections	157	72.0
	Repeated task completion attempts	109	50.0
	Parsing failure feedback	99	45.4
	Concise task structure	79	36.2
	Detailed extraction from tables	74	33.9
	Goal completion without errors	67	30.7
	Multiple information location attempts	65	29.8
	Repeated parsing errors	51	23.4
	Technical errors (non-navigation)	49	22.5
	URL error references	44	20.2
	Task completion execution errors	37	17.0
	Travel-related task focus	36	16.5
	Cookie consent success	21	9.6
	CAPTCHA/verification failures	15	6.9
	Inadequate human feedback	8	3.7

J Captcha Solving Strategy Preferences

Captcha-Solving Strategy	Gemini 2.5	o4-mini	R1	Llama-4	Claude-3.7
Cache	0.00	45.45	1.82	0.00	0.00
Mobile	0.00	58.64	5.91	0.00	0.00
Direct Link	25.45	97.73	81.82	69.55	60.91
Google Search	42.73	100.00	94.55	61.36	77.27
Randomized Interaction	0.00	0.45	25.91	3.18	0.00
Reloads	9.09	3.64	27.73	12.27	1.82
New Tab	4.55	60.45	52.27	69.55	12.27
Switch Websites	15.45	5.00	31.82	22.73	60.00
Internal Navigation	0.91	40.00	22.73	0.00	0.45
Country Domain	0.45	29.09	0.91	0.00	0.00
Text-only Rendering	0.00	7.27	0.00	0.00	0.00
Public Proxy	0.00	1.36	0.00	0.00	0.00
Internet Archive	0.00	3.64	0.00	0.00	0.00
Google Travel Integration	0.00	0.45	1.36	0.00	0.91

Table 3: Percentage of times a particular captcha avoidance strategy was deployed by a model while solving tasks in the Expedia task dataset

K Pop-Up Banner Closure Scenarios

Pop-Up Banner Scenarios	Gemini 2.5	o4-mini	R1	Llama-4	Claude-3.7
Banner Detected	53.75	91.25	0.00	98.75	**100.00**
Banner Closed	4.65	**17.81**	0.00	17.72	7.5
Marked as Completed	7.5	23.75	**53.75**	3.75	2.5

Table 4: Percentage of times a particular pop-up banner scenario was observed in an agent's trace while executing tasks from the BBC task dataset. We note that the percentage in the Banner Closed row is determined with respect to the number of tasks where the agent determines that there is a banner as per the LLM judge. The other two rows (Banner Detected and Marked as Completed) are computed with respect to the total number of tasks in the BBC dataset.

L Direct Navigation Actions Taken

	Google API	Google Site	Wiki	Direct Answer	Failed
Claude-3.7	97	3	0	0	0
R1	98	2	0	0	0
Gemini 2.5	50	0	0	0	50
Llama-4	74	26	0	0	0
o4-mini	76	2	1	9	12

Table 5: Count of times agents taking different actions when asked questions from TriviaQA dataset.

M Examples of User-Submitted Tasks

Task Prompt 1: Find me the last available train from Cardiff Central to Barry Docks station today on trainline.

Task Prompt 2: Compare flight prices from washington DC to Paris france to flights from washington DC to Kyoto Japan