TOWARDS INTERNET-SCALE TRAINING FOR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

The predominant approach for training web navigation agents gathers human demonstrations for a set of popular websites and hand-written tasks, but it is becoming clear that human data is an inefficient resource. We develop a pipeline to facilitate large-scale training for agents without laborious human annotations. In the first stage, an LLM generates tasks for 150k diverse websites. In the next stage, LLM agents complete tasks and produce trajectories. In the final stage, an LLM reviews the trajectories, and judges their success. Language models are competitive with human annotators, detecting and filtering out harmful content with an accuracy of 97%, generating tasks with a feasibility rate of 89%, and judging successful trajectories at 82.6% accuracy. Scaling the pipeline, agents based on *Llama 3.1 70B* solve 16.7% of tasks for 150k sites. Training on data generated by our pipeline is competitive with training on human demonstrations. In data-limited experiments derived from Mind2Web and WebLINX, we improve *Step Accuracy* by +89.5% and +94.5% respectively for agents trained on mixtures of human data and data from our pipeline. Our code is available at: data-for-agents.github.io.

1 INTRODUCTION

The predominant approach for training LLM-based web navigation agents is to collect human demonstrations across a set of manually curated websites and tasks Deng et al. (2023); Zhou et al. (2024b); Putta et al. (2024); Koh et al. (2024a); Liu et al. (2024); Lù et al. (2024); Rawles et al. (2023). Human data can be laborious to collect, and becomes costly to scale as the breadth of skills that users require from language model agents grows. There are more than three-hundred million sites on the internet The Common Crawl Foundation (2024), and the range of sites that researchers can manually prepare for human annotation represents a tiny fraction of the internet. The key problem is that human data can become unreliable at scale. Human-written web navigation tasks are highly effective for popular sites, but reliability drops for sites with lower popularity due to annotators' lack of familiarity. For these sites with lower popularity, which represent the majority of sites on the internet, human-written web navigation tasks are feasible just 40% of the time, requiring a costly manual verification step. For this same collection of sites, language models improve feasibility rates to more than 80%. There is a growing need to automate data pipelines for a next generation of agents trained at internet scale. We address a key challenge by reducing dependency on human data in the agent pipeline. We develop an automatic data pipeline that aims to facilitate Internet-Scale Training for Agents (shortened to InSTA)—a pipeline that relies on synthetic web navigation tasks proposed, attempted, and evaluated by language models.

Our method operates in three stages. In the first stage, we employ a language model to propose candidate web navigation tasks for an agent to perform across 150k live sites on the internet. Current works are limited to 200 popular sites Lù et al. (2024); Rawles et al. (2023); Deng et al. (2023) that humans annotators are likely to be familiar with. Language models help us scale to *1,000* times more sites than current efforts, with better coverage of real-world sites. One major consideration when scaling up training for agents is safety: building safe agents requires that we avoid sites with harmful, unsafe, or dangerous content. We evaluate the aptitude of language models at detecting such content, and aggressively filter out 85% of candidates from 1M initial sites down to 150k sites that are judged as safe by language models. These models succeed at detecting safe content with an accuracy of 97%, compared to 75% human accuracy. With tasks generated across a safe and diverse set of websites, we proceed to run language model agents to attempt the generated tasks.

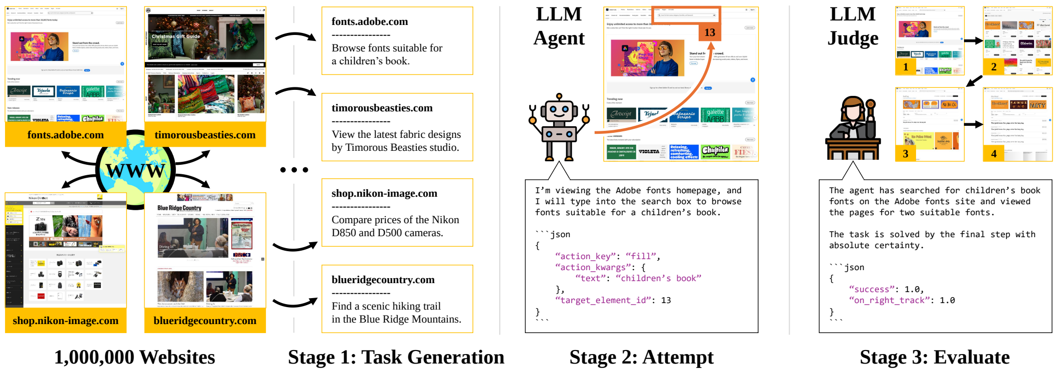

Figure 1: **Overview of the proposed agent pipeline.** We develop a pipeline for training web navigation agents at internet scale using tasks proposed, attempted, and evaluated by pretrained large language models. We generate 150k diverse tasks across 1M internet sites. Code for our data generation pipeline, and traces for agent rollouts will be available on our website: data-for-agents.github.io.

In the second stage of the pipeline, a language model agent attempts to complete tasks using a web browser. We provide the entire Playwright API to the agent, which operates the browser by generating function calls in the Playwright API. In the third stage of the pipeline, we scale evaluation using language models. We employ LLMs to judge Lightman et al. (2024) whether a task is solved by the final timestep, and obtain an accuracy up to 93.1% at detecting successful trajectories for the most confident predictions. Llama-3.1-70B-Instruct solves 16.7% of tasks zero-shot with a judge confidence of `conf` = 1. In a data-limited setting, training language model agents on data from our pipeline beats human demonstrations by up to +89.5% on Mind2Web, and up to +94.5% on WebLINX, highlighting the utility of our synthetic data for training LLM agents.

2 Internet-Scale Task Generation

Building internet-scale agents requires a diverse scaffold of tasks and environment configurations beyond what can be attained via manually curated examples annotated by humans. We develop a pipeline to efficiently harness vast quantities of sites on the internet that aims to facilitate Internet-Scale Training for Agents (InSTA). Our pipeline uses pretrained language models to generate, attempt, and evaluate synthetic web navigation tasks for a more diverse pool of sites than current efforts that rely on tasks manually curated by researchers Deng et al. (2023); Zhou et al. (2024b); Putta et al. (2024); Koh et al. (2024a); Liu et al. (2024); Lù et al. (2024); Rawles et al. (2023); He et al. (2024). Human data is a valuable yet finite resource, and we show that language models can be just as accurate. By removing human data from the agent pipeline, we can improve the safety and reliability of tasks, and efficiently scale task generation to 1M sites.

2.1 Language Model Task Proposer

In the first stage, we generate web navigation tasks using a **Language Model Task Proposer**. The task proposer is depicted in Figure 10, and serves two key functions in the pipeline: (1) filtering sites that cannot be safely annotated, especially those with harmful content, and (2) proposing realistic web navigation tasks that a hypothetical user might want to accomplish.

Model Details. We utilize pretrained and frozen language models that conform to a chat interface and accept a system prompt $\mathbf{x}_{\text{sys}}$, and a series of in-context examples via interleaved user and assistant prompts $\mathbf{x}_{\text{usr}}$ and $\mathbf{x}_{\text{ast}}$. The system prompt used for task generation is shown in Figure 10, and outlines all cases for which sites are considered unsafe for annotation. We consider the Llama 3.1 family of LLMs from Meta Grattafiori et al. (2024); Touvron et al. (2023b;a), the GPT family of LLMs from OpenAI, and the Gemini family of LLMs from Google. Inference is served using vLLM Kwon et al. (2023) for the Llama series of models. We employ a sampling temperature of 0.5, and a maximum budget of 64 newly generated tokens, all other parameters are kept as defaults in the OpenAI chat completions API, which is used to make inference calls to all LLMs.

Method	Acc.	Prec.	Recall
Llama 3.1 70B	85%	0.77	**1.00**
GPT-4o	95%	0.91	**1.00**
Gemini 1.5 Pro	**97%**	**0.96**	0.98
Human Baseline	75%	0.71	0.84

Figure 2: **Accuracy for detecting harmful sites.** We curate a set of 100 website domains, where 50 are safe, and 50 are unsafe based on filtering conditions in Figure 10. Pretrained language models exceed the accuracy and recall of human annotators at detecting harmful sites that are unsuitable for training agents.

Method	Feasibility Rate
Llama 3.1 70B	75%
GPT-4o	85%
Gemini 1.5 Pro	**89%**
Human Baseline	54%

Figure 3: **Expert feasibility of proposed tasks.** We propose web navigation tasks on 100 curated sites (listed in Appendix H), and measure the completion rates of human participants. Language models exceed the performance of human annotators at creating realistic web navigation tasks for LLM agents to perform.

Prompt Details. The goal of the task proposer is to accurately detect unsafe websites, and generate realistic web navigation tasks when suitable. We prompt the task proposer with the system prompt in Figure 10, a series of in-context examples (listed in Appendix H), and a final user prompt containing just the URL of the target website. We instruct the LLM via the system prompt to provide a task for the target website, or to return "N/A" and mark the website as not suitable for annotation. This format produces a throughput of 20 websites per second for *Llama 3.1 70B* served on 16 GPUs with vLLM, processing 1M sites in 14 hours. The efficiency of stage one aids in scaling to large numbers of sites on the internet, but we must not compromise safety and reliability for efficiency. To understand the trade-offs presented by our task proposal approach, we compare against typical human annotators at detecting safe websites for annotation, and creating realistic agent tasks.

2.2 Improving Safety

Language models *beat single pass human annotators* at detecting websites suitable for annotation. To evaluate detection performance, we employ the task proposer as a classifier, and consider sites where the task proposer returns "N/A" as the positive class. We curate 50 safe, and 50 unsafe domains, based on the filtering conditions outlined in the system prompt in Figure 10 (selected websites and their URLs are listed in Appendix H). We generate task proposals for each site, and measure the accuracy, precision, and recall of our safety filter compared to human annotators. The annotators are asked to classify each site as suitable or unsuitable for annotation based on the website URL, and the criteria listed in the system prompt, the same observations given to the task proposer to ensure a fair comparison. Results are presented in Table 2.

Understanding The Results. Language models outperform human annotators by 29.3% in accuracy, 35.2% in precision, and 31.0% in recall at detecting harmful sites. While larger models like *Gemini 1.5 Pro* show best overall accuracy, smaller models like *Llama 3.1 70B* display high recall with a minor drop in accuracy. Recall matters most for safety filters, and these results suggest *Llama 3.1 70B* is sufficient to detect most harmful sites with high confidence.

2.3 Improving Reliability

Language models are *more reliable than single pass human annotators* at creating realistic web navigation tasks. To evaluate reliability, we measure the rate that human workers are able to accomplish web navigation tasks generated by our pipeline. We select 100 safe website domains (different from the safety experiment, refer to Appendix H), generate task proposals using our pipeline, and measure the rate of self-reported task completion for human workers performing tasks. Workers start from the initial website URL in their browser, and navigate pages using their mouse and keyboard while staying on the original site, reporting once the task is complete, or once they believe the task is not feasible. We compare feasibility rates for tasks generated by our pipeline to tasks written by human annotators given the criteria for tasks listed in Figure 10. Results are shown in Table 3.

Understanding The Results. Language models outperform human annotators by 64.8% at creating feasible web navigation tasks. Larger models like *Gemini 1.5 Pro* display the best feasibility rates, but the smaller model *Llama 3.1 70B* still outperforms human annotators by 38.9%. To understand the relationship between the popularity of the site being annotated, and the reliability of human-written tasks, we conduct an experiment in Figure 4 comparing PageRank values Page et al. (1999) of sites according to the official June, 2024 host-level web graph from The Common Crawl Foundation (2024), versus the feasibility rates of proposed tasks from Table 3.

While human annotators match the reliability of LLMs at creating feasible web navigation tasks for popular sites, LLMs outperform human annotators by 157.1% for less popular sites with low PageRank values. As the obscurity increases, human annotators are less familiar with sites, and the reliability of their task proposals decreases by 55.7%, whereas the reliability of tasks generated by LLMs remains relatively constant. This difference suggests that we should employ language models to ensure reliable task proposals as we begin to scale agents to vast numbers of sites on the internet. But, where do we acquire this large and diverse set of websites to process for annotation?

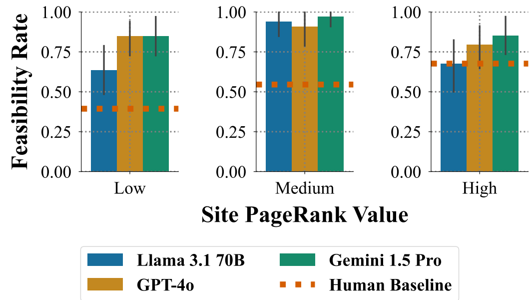

Figure 4: **Feasibility rates vs PageRank.** We visualize PageRank values, a useful proxy for the popularity of websites, versus the expert feasibility rates of proposed web tasks. Human-written tasks perform on par with LLMs for popular sites, but as target sites become less popular and annotators are less familiar with them, LLMs begin to outperform human annotators.

2.4 Scaling To 150,000 Sites

We propose to leverage open-source crawls of the internet for large-scale task generation. As of June, 2024, the web graph released by The Common Crawl Foundation (2024) contains more than 300 million unique hosts, which we adapt into a data source for agents. In particular, we sort hosts by their PageRank values, and select the top 1M sites for task generation. CommonCrawl is likely to contain many sites not suitable for annotation, and experiments in Section 2.2 illustrate the safety filter in the task proposer can effectively detect and remove them. In our configuration, task generation with *Llama 3.1 70B* takes 14 hours for 1M sites served with vLLM Kwon et al. (2023) on two 8-GPU nodes. Sections 2.2 and 2.3 show *Llama 3.1 70B* outperforms human annotators in safety and reliability, and we can serve it locally at significantly reduced cost versus proprietary LLMs with a marginal loss in quality. The distribution for tasks generated with *Llama 3.1 70B* for the top 1M sites in the CommonCrawl PageRank are visualized in Figure 5.

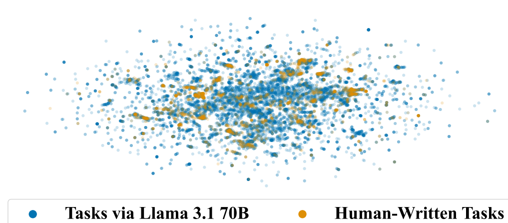

Figure 5: **Distribution of 150k tasks.** We compare the distribution of tasks generated by our pipeline (*blue* points) to the Mind2Web Deng et al. (2023) dataset (*orange* points) via textual features extracted by a sentence embedding model, and projected in 2D with UMAP McInnes et al. (2020). Our distribution is denser than human-written tasks, and has broad coverage of real-world sites, and diverse categories of tasks.

Understanding The Data. The task proposer filters out 85% of sites in CommonCrawl, resulting in 150k sites that can be safely assigned tasks for agents. Visualized in Figure 5, our distribution has broad coverage of real-world sites, and diverse categories of tasks. We automatically label task categories (procedure in Appendix H) and find that 89% of categories have fewer than the mean of 16.9 tasks per category. Top categories include *news search*, *recipe search*, *product lookup*, *tutorial search*, *event schedules*, *health information*, and many more. Refer to Appendix H for the top categories. Empowered by this large and diverse collection of tasks from across the internet, we can start to build internet-scale agents.

3 Internet-Scale Agents

In the next stage, we run agents on diverse web navigation tasks. Shown in Figure 6, we initialize a web browsing environment to the URL provided to the task proposer in Section 2, and run a language model agent to complete tasks by generating function calls in the Playwright API. For evaluation, current efforts typically use human-written constraints based on the final URL or page

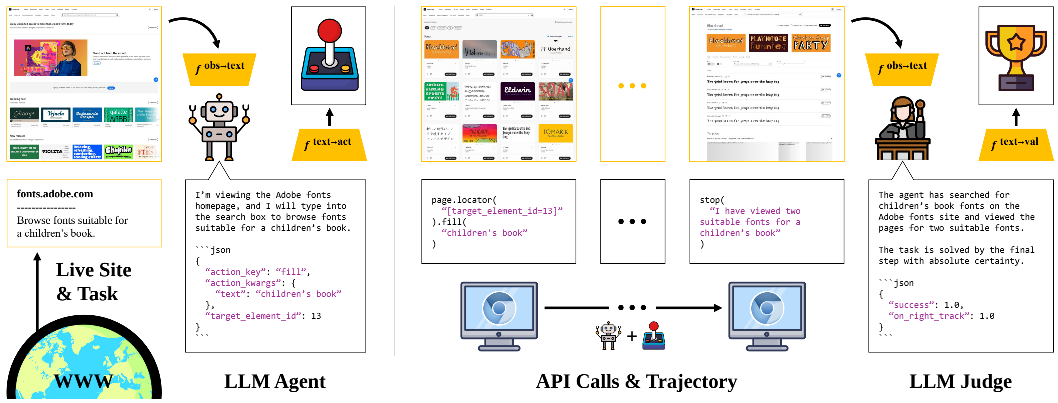

Figure 6: **Automatic evaluation for agents with language model judges.** Building on the large and diverse set of tasks generated by the pipeline, we employ pretrained language models to attempt and evaluate web navigation tasks. We dispatch language model agents to perform tasks by making calls to the Playwright API. We then employ language model judges to evaluate rollouts from agents.

state Zhou et al. (2024b); Koh et al. (2024a); Yao et al. (2023a); Drouin et al. (2024), but it can be difficult to scale these. Recall from Figure 4 that human annotators are less reliable for sites lower in the PageRank, where their familiarity is reduced. Results in section 2 showed that language models beat humans in safety and reliability for task generation. As we begin to scale agents to diverse internet tasks, can we replace human-written criteria with language model judgments for efficient evaluation? Their robustness remains an important unresolved question, as previous works have only considered language model judges for the limited set of popular websites from He et al. (2024). We begin by validating the robustness of language models for evaluating diverse internet tasks.

3.1 Evaluation With Language Models

Building on the sites used to measure reliability in Section 2.3, we conduct an experiment to measure the accuracy of language models for detecting successful web navigation trajectories. Experimental details are discussed in Appendix C, and results are shown in Figure 7.

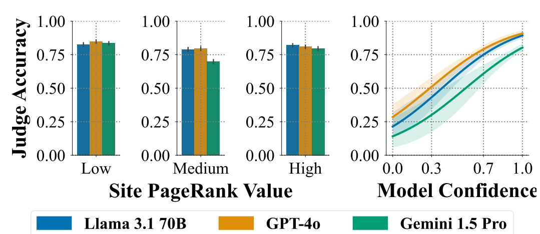

Figure 7: **Language models are robust evaluators.** We measure the accuracy of language models for detecting successful web navigation trajectories, and find that accuracy remains stable relative to PageRank values (*left plot*). As models become more confident, their accuracy improves (*right plot*), suggesting confidence is a useful proxy for the reliability of their predictions.

Language models are *robust evaluators for web navigation tasks*. Accuracy remains stable relative to PageRank values, suggesting that language models are effective for sites that typical human annotators are less familiar with. Best results are obtained with an evaluator based on *GPT-4o*, which attains an accuracy of 82.6%, compared to 81.7% for *Llama 3.1 70B*, and 78.0% for *Gemini 1.5 Pro*. While accuracy is robust to PageRank, the accuracy is highly informed by confidence. Language models show improved accuracy as their confidence improves, suggesting they can effectively determine when their predictions are reliable. When considering predictions with `conf` = 1, the *Llama 3.1 70B* evaluator displays a compelling 93.1% accuracy, 0.87 precision, and 0.82 recall for detecting successful web navigation trajectories. Now that we can efficiently and accurately judge trajectories, we can begin to scale language model agents to diverse internet tasks, and track their success. Harnessing this judge, we can study the current abilities and shortcomings of language model agents spanning 150k diverse live websites.

3.2 Scaling To 150,000 Agents

We scale language model agents to 150k lives sites in diverse domains across the internet, and attempt to complete 150k web navigation tasks generated by our pipeline. Shown in Figure 8, we evaluate trajectories using a *Llama 3.1 70B* judge, and run agents based on *Llama 3.1 70B*, selected because this model demonstrates high accuracy in Figure 7, and running currently available propriety models would be prohibitively expensive at this scale—see Appendix N for a cost analysis with different LLMs. We find that agents solve 16.7% of tasks with a model confidence of `conf` = 1. Furthermore, we observe that 35k tasks are judged to be on the right track with a confidence of `conf` = 1, suggesting these could be solved if a larger compute budget were allocated. The spread along the x-axis in both plots in Figure 8 suggests that our tasks cover a broad range of difficulties, and working to solve them presents an opportunity for improving the capabilities of LLM agents. We observe that, when judging success, our evaluator tends to prefer binary predictions with high confidence values, suggesting this subset of predictions is accurate based on Figure 7 results. Additional visualizations and analyses for the agents that produced Figure 8 are presented in Appendix I.

4 Training Agents

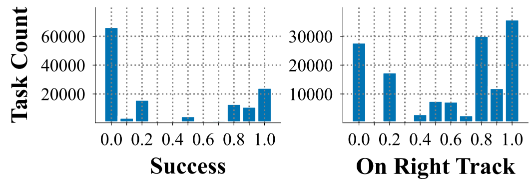

Figure 8: **Scaling agents to 150k live websites.** We run agents based on Llama 3.1 70B to complete tasks generated by our pipeline. We estimate the agent's success probability (*left plot*) using a language model evaluator, and estimate the probability the agent is on the right track (*right plot*). 16.7% of rollouts are estimated to be successful with `conf` = 1, and the spread of probabilities suggests the data spans many difficulties.

We compare agents trained on data from the InSTA pipeline to agents trained on human demonstrations from Mind2Web (Deng et al., 2023) and WebLINX (Lù et al., 2024), two popular benchmarks for web navigation agents. Recent work that mixes synthetic data with real data uses ratios from 50% to 80% real data (Trabucco et al., 2024), and we find a 50% ratio for the Mind2Web dataset, and an 80% ratio for the WebLINX dataset to work best. Shown by Figure 8, the distribution of our data has a broad performance spread, so we apply filtering rules to select high-quality training data. First, we require the evaluator to have returned `conf` = 1 that the rollout is a success, and that the agent was on the right track (this selects data where the actions are reliable, and directly caused the task to be solved). Second, we filter for data where the trajectory contains at least three actions. Third, we remove data where the agent encountered a server error, was presented with a captcha, or was blocked at any timestep in the trajectory. These filtering steps produce a set of $7,463$ synthetic demonstrations from our pipeline where agents successfully completed tasks generated by the InSTA pipeline. We uniformly at random select 500 demonstrations for our test set, and employ the remaining $6,963$ demonstrations for training.

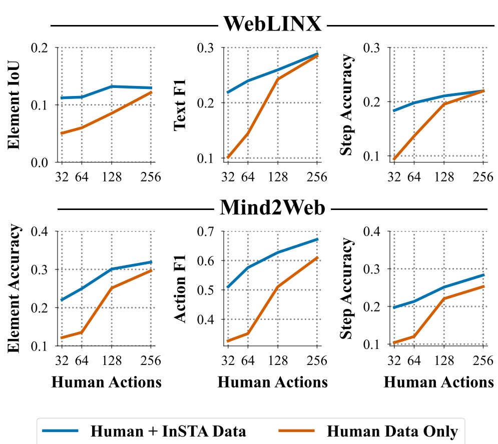

Figure 9: **Data-limited results with our data.** Language model agents trained on mixtures of our data and human demonstrations scale faster than agents trained on human data. In a data-limited setting with 32 human actions, mixing our data with human demonstrations improves *Step Accuracy* by +89.5% relative to human data for Mind2Web, and improves by +94.5% for WebLINX.

Understanding The Results. Language model agents trained with our data *scale faster with increasing data size* than agents trained with just human data. Without requiring human annotations when generating data, our method leads to improvements that range from +89.5% in *Step Accuracy* on the Mind2Web benchmark (the rate at which the correct element is selected, and the correct action is performed on that element) with 32 human examples, to +77.5% with 64 human examples, +13.8% with 128 human examples, and +12.1% with 256 human examples. Similarly, our data leads to improvements in *Step Accuracy* on the WebLINX benchmark that range from +94.5% with 32 human examples, to +44.8% with 64 human examples, +7.8% with 128 human examples, and +0.1% with 256 human examples. Our work reveals several exciting directions for future work. First, our work can be scaled further. The latest CommonCrawl release contains data for more than 300 million sites, suggesting another *1,000* times more data could be available by scaling the pipeline further. In addition, our judge was employed offline, and its high accuracy suggests that it could be used to guide an online algorithm. Finally, we considered only text-based agents in this work, and our pipeline could be extended to generate data for multimodal tasks.

REFERENCES

Jacob Andreas. Language models as agent models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 5769–5779, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.423. URL https://aclanthology.org/2022.findings-emnlp.423.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, Mar. 2024. doi: 10.1609/aaai.v38i16.29720. URL https://ojs.aaai.org/index.php/AAAI/article/view/29720.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023. URL https://arxiv.org/abs/2303.12712.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023. URL https://arxiv.org/abs/2310.05915.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. The browsergym ecosystem for web agent research, 2024. URL https://arxiv.org/abs/2412.05467.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL https://arxiv.org/abs/2306.06070.

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

Saumya Gandhi, Ritu Gala, Vijay Viswanathan, Tongshuang Wu, and Graham Neubig. Better synthetic data by retrieving and transforming existing datasets, 2024. URL https://arxiv.org/abs/2404.14361.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, and et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371. URL https://aclanthology.org/2024.acl-long.371.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2023. URL https://arxiv.org/abs/2312.08914.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama guard: Llm-based input-output safeguard for human-ai conversations, 2023. URL https://arxiv.org/abs/2312.06674.

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.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback, 2024. URL https://arxiv.org/abs/2309.00267.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *arXiv preprint arXiv: 2411.16594*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding?, 2024. URL https://arxiv.org/abs/2404.05955.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue, 2024. URL https://arxiv.org/abs/2402.05930.

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. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020. URL https://arxiv.org/abs/1802.03426.

Microsoft. Playwright. https://github.com/microsoft/playwright, 2024.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. Agentinstruct: Toward generative teaching with agentic flows, 2024. URL https://arxiv.org/abs/2407.03502.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale, 2024. URL https://arxiv.org/abs/2409.15637.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9781713871088.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford infolab, 1999.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks, 2024. URL https://arxiv.org/abs/2405.20309.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. REFINER: Reasoning feedback on intermediate representations. In Yvette Graham and Matthew Purver (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1100–1126, St. Julian's, Malta, March 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.eacl-long.67.

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

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023. URL https://arxiv.org/abs/2307.10088.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold, 2024. URL https://arxiv.org/abs/2406.14532.

Junhong Shen, Atishay Jain, Zedian Xiao, Ishan Amlekar, Mouad Hadji, Aaron Podolny, and Ameet Talwalkar. Scribeagent: Towards specialized web agents using production-scale workflow data, 2024. URL https://arxiv.org/abs/2411.15004.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. Fast best-of-n decoding via speculative rejection. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=348hfcprUs.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of llms should leverage suboptimal, on-policy data, 2024. URL https://arxiv.org/abs/2404.14367.

The Common Crawl Foundation. Common crawl, 2024. URL https://commoncrawl.org/.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a. URL `https://arxiv.org/abs/2302.13971`.

Hugo Touvron, Louis Martin, Kevin Stone, and et al. Llama 2: Open foundation and fine-tuned chat models, 2023b. URL `https://arxiv.org/abs/2307.09288`.

Brandon Trabucco, Kyle Doherty, Max A Gurinas, and Ruslan Salakhutdinov. Effective data augmentation with diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=ZWzUA9zeAg`.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench, 2024. URL `https://arxiv.org/abs/2409.13373`.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL `http://dx.doi.org/10.1007/s11704-024-40231-1`.

Junlin Xie, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. Large multimodal agents: A survey, 2024. URL `https://arxiv.org/abs/2402.15116`.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023a. URL `https://arxiv.org/abs/2207.01206`.

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, 2023b. URL `https://arxiv.org/abs/2305.10601`.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text, 2024. URL `https://arxiv.org/abs/2406.07496`.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 2023. URL `https://arxiv.org/abs/2310.12823`.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users, 2023. URL `https://arxiv.org/abs/2312.13771`.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction, 2024. URL `https://arxiv.org/abs/2408.15240`.

Tianyang Zhong, Zhengliang Liu, Yi Pan, and et al. Evaluation of openai o1: Opportunities and challenges of agi, 2024. URL `https://arxiv.org/abs/2409.18486`.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024a. URL `https://arxiv.org/abs/2310.04406`.

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

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator(pae): Autonomous skill discovery for foundation model internet agents, 2024c. URL `https://arxiv.org/abs/2412.13194`.

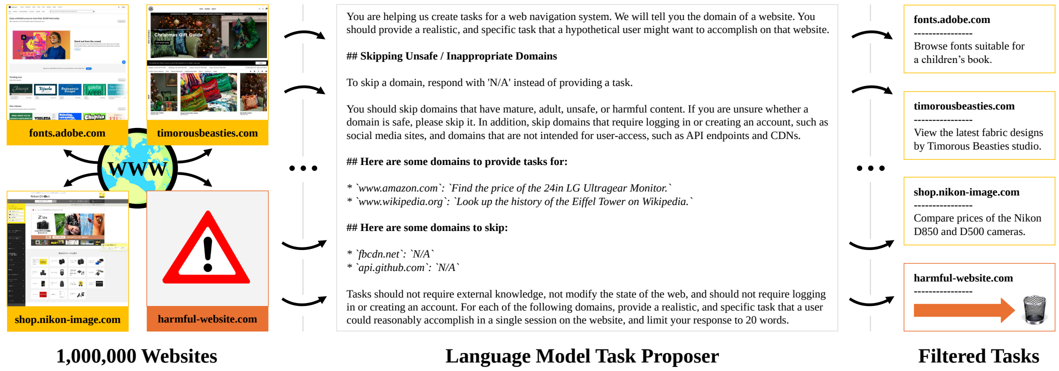

Figure 10: **Task proposal and filtering for 150k live websites.** Starting from 1,000,000 websites, we employ a pretrained language model that marks sites as safe/unsafe for annotation, and assigns a realistic task that a hypothetical user might want to accomplish on each site. The task proposer rejects 85% of websites from the pipeline, resulting in 150k safe websites annotated with realistic tasks.

A Related Works

Language Model Agents. There is an emerging paradigm in modern NLP using language models Radford et al. (2019); Brown et al. (2020); Touvron et al. (2023a;b) as backbones for agents Andreas (2022). These models display impressive reasoning capabilities Bubeck et al. (2023); Zhong et al. (2024); Valmeekam et al. (2024) that allow them to generalize to downstream applications, such as web navigation, where text formats differ significantly from their training data. Search algorithms provide a secondary axis to improve the reasoning capabilities of the language model agents Yao et al. (2023b); Besta et al. (2024); Koh et al. (2024b); Zhou et al. (2024a) by providing an explicit algorithmic scaffold, and allowing test-time compute to improve reasoning steps Snell et al. (2024); Zhong et al. (2024). While the majority of works focus on running language models as agents zero-shot, fine-tuning language models to improve their effectiveness as agents is becoming popular Putta et al. (2024); Zeng et al. (2023); Zhang et al. (2023); Hong et al. (2023); Xie et al. (2024); Wang et al. (2024) as target benchmarks are becoming more difficult for zero-shot language models.

Agent Pipelines. There are a growing number of agent pipelines aimed at fine-tuning language models to improve their effectiveness as agents Mitra et al. (2024); Zeng et al. (2023); Putta et al. (2024); Chen et al. (2023); Ou et al. (2024). However, driven by the limited data available, many such works train on data with significant overlap with their test environment—either with different tasks for the same environment configuration as the test setting Deng et al. (2023), or the same tasks Putta et al. (2024). We instead consider a setting where tasks and environment configurations are entirely separate between training and testing, creating a strong train-test split that follows recommended practice. This presents a challenge—web navigation data for training LLM agents is limited Deng et al. (2023); Lù et al. (2024). We address this challenge with scale, and better coverage of the distribution of real-world sites. We train on diverse tasks generated by our pipeline, and successfully transfer agents trained on our data to downstream benchmarks while maintaining a strong train-test split. Our training procedure resembles a modified FireAct Chen et al. (2023), where language models jointly propose and evaluate tasks for agents.

Agent Datasets. The majority of datasets for training web navigation agents rely on human annotators to create tasks Zhou et al. (2024b); Koh et al. (2024a); Rawles et al. (2023), and provide demonstrations Deng et al. (2023); Lù et al. (2024); Rawles et al. (2023); Shen et al. (2024). This approach has limits, as the breadth and diversity of tasks researchers can manually curate is dwarfed by the sheer quantity of sites on the internet. There are more than three-hundred million sites on the internet according to The Common Crawl Foundation (2024), and existing datasets are limited to about 150 popular sites that human annotators are already familiar with Deng et al. (2023); Lù et al. (2024); Shen et al. (2024). There is a hypothetical *1,000,000* times more data that could be available if we can efficiently harness this previously untapped resource. However, the majority of sites are relatively obscure, and human annotators are unreliable for sites they are not already familiar with. Finding suitable annotators becomes impractical at this massive scale, so we adapt language models to propose, attempt, and evaluate web navigation tasks. While we are not the first to consider synthetic data for training agents Gandhi et al. (2024); Ou et al. (2024); Setlur et al. (2024); Tajwar et al. (2024), we have developed a key approach to harness internet-scale data efficiently.

Language Model Judges. Core to our pipeline is a language model evaluator. Using language models to judge the correctness of responses is becoming popular to improve accuracy for LLMs Li et al. (2024), and applications include verifying reasoning steps Zhang et al. (2024), rejection sampling Snell et al. (2024); Sun et al. (2024), prioritizing frontier nodes in search algorithms Zhou et al. (2024a); Koh et al. (2024b), filtering out harmful responses Inan et al. (2023), providing feedback for response improvement Madaan et al. (2023); Paul et al. (2024); Patel et al. (2024); Yuksekgonul et al. (2024), and providing ratings for alignment Lee et al. (2024); Ouyang et al. (2024). Our use of language models to evaluate agent tasks is inspired by the generative verifier in Zhang et al. (2024), and modified from the multimodal verifier in He et al. (2024), where our language model predicts a confidence score that a task is solved, which is used to identify successful attempts.

B Language Model Agents

Language model agents are a class of decision-making agent represented by $\pi_{\text{LLM}}(\mathbf{a}_t|\mathbf{s}_t, \mathbf{c})$, a policy that processes multimodal observations $\mathbf{s}_t$, and predicts textual actions $\mathbf{a}_t$ in order to complete a task $\mathbf{c}$. Underneath this abstraction, a large language model (LLM) generates actions via next-token prediction, conditioned on a system prompt $\mathbf{x}_{\text{sys}}$.

$$\mathbf{a}_t = f^{\text{ text}\rightarrow\text{act}}\Big(\text{ LLM}([\ \mathbf{x}_{\text{sys}}, \mathbf{c}, \text{Enc}(\mathbf{s}_t)\ ])\ \Big) \tag{1}$$

Environment representations for observations and actions typically differ from the language model's expected format, and functions are introduced that map the observations into a multimodal prompt $\text{Enc}(\cdot)$, and parse actions from the language model's completion $f^{\text{ text}\rightarrow\text{act}}(\cdot)$. For web navigation, the environment state $\mathbf{s}_t$ is HTML DOM, and is often formatted as raw HTML code, an Accessibility Tree, Set-of-marks, or screenshots Zhou et al. (2024b); Koh et al. (2024a); Chezelles et al. (2024); Shen et al. (2024). Action formats vary between works, and we build on Schick et al. (2023)'s function-calling framework, where a language model generates code that is parsed into a function name, and corresponding arguments. Given a set of strings L, and a set of function argument values G, the set of actions $\mathcal{A}$ is:

$$\mathcal{A} = \Big(L_{\text{func}} \times (L_{\text{arg1}} \times G_{\text{arg1}}) \times (L_{\text{arg2}} \times G_{\text{arg2}}) \times \cdots\Big) \tag{2}$$

Where L_{func} is the set of function names in our API, and function arguments have a name and value $(L_{\text{arg1}} \times G_{\text{arg1}})$. We provide the agent access to the entire API for Playwright Microsoft (2024), a browser automation library developed by Microsoft that wraps around a Chromium web browser. The agent's goal is to complete a web navigation task specified via a natural language instruction $\mathbf{c} \in L$, starting from an initial URL, and operating the browser via function calls to the Playwright API until the task is complete, after which point the agent calls `stop` with an optional answer:

$$\mathbf{a}_{\text{stop}} = \Big(\text{"stop"}, (\text{"answer"}, \text{"I am done"})\Big) \tag{3}$$

We prompt the language model backbone to generate responses in a Markdown format, where desired actions are wrapped in a JSON code block for straightforward parsing. The action parser $f^{\text{ text}\rightarrow\text{act}}$ consists of a regex template that matches to the first JSON code block, such as the example in Figure 1, followed by JSON decoding on the string contents within the code block. When parsing fails due to invalid syntax, we generate a new response until parsing succeeds. Equipped with a language model agent that makes calls to the Playwright API, we face a crucial roadblock that impedes scaling—obtaining large and diverse data.

C Experimental Details For Judge Accuracy

We run agents on tasks generated by Llama 3.1 70B for the 100 sites in Section 2.3, and prompt language models to estimate the probability that tasks are solved by the final timestep $\mathbf{r}_T$. We then conduct human evaluations for the trajectories and manually assign binary success labels. Accuracy is calculated by applying a threshold to the predictions $\mathbf{r}_T > 0.5$ to assign classes, and tracking the rate that predictions agree with human labels. To understand robustness for sites of varying popularity, we report the accuracy of language models versus the PageRank of corresponding sites.

Similarly, to understand the ability of language models to judge their own uncertainty, we report their accuracy versus their prediction confidence, given by $\texttt{conf} = 2 \cdot |\mathbf{r}_T - 1/2|$ (twice the total variation distance from the uniform distribution to the predicted distribution).

D Limitations & Safeguards

Language model agents present unique challenges and risks when applied to live tasks on the internet. For instance, agents visiting shopping sites can influence the statistics produced by analytics tools, which can impact prices on products, and product decisions from companies. Furthermore, agents visiting harmful content can add such harmful content to datasets, and perpetuate harmful behaviors into the training data for future agents. We mitigate these risks by carefully designing the task proposal stage of the InSTA pipeline. We consider the risks posed to analytics tools by limiting the engagement between agents and sites. We generate only one task per website, and we limit agents to just 10 actions per site, which includes clicks, typing, dropdown selection actions, and more. By limiting the interaction between agents and sites, the change in website traffic generated by the InSTA pipeline is minimal (just 30 seconds of interaction per site on average). By utilizing data from the InSTA pipeline in an offline fashion, as in Section 4 of the main paper, no additional web traffic is generated when training agents. To ensure that agents do not modify the state of the web (i.e. avoid attempting to make purchases, avoid leaving comments on posts, avoid making accounts, etc), we provide an instruction in the system prompt of the task proposer (see Figure 10) to avoid writing tasks that require the agent to modify the state of the web.

The task proposer is instructed via the system prompt to filter out sites with harmful content, sites not intended for user access, and sites that require making an account to operate (such as social media, and forums). We explore the performance of the task proposer at filtering out unsuitable sites in Section 2.2, and find that all models detect unsuitable sites with a recall from 0.98 to 1.0, and accuracy up to 97%, suggesting our filter is reliable. Sites used to benchmark the performance of the safety filter are discussed in Appendix H, and include categories of harmful, and mature content.

E Ethical Considerations

One important ethical consideration when gathering data from the internet is to handle copyrighted, private, and sensitive materials carefully. The internet contains vast amounts of personal data created by users that includes personally-identifying-information that should not be included in public datasets. We address this ethical consideration in two ways. First, the task proposer is instructed to filter out social media sites and forums that are likely to contains personally-identifying-information. Second, we store and release only the prompts we used, and traces for agents' actions—importantly, we do not release any web source code that could be used to recover sensitive data. These steps significantly reduce, but do not completely eliminate the risk that private, and sensitive materials are included in our data, and methods for detecting, replacing, and removing such materials from datasets remains an important task for researchers working on safety.

F Broader Impacts

As their capabilities broaden, language model agents are being increasingly used to operate real-world systems and APIs. This shift comes with several benefits and risks. Agents that operate your computer to aid in work tasks can significantly boost productivity for certain workers, but can displace others whose jobs have been fully automated. Agents that operate web browsers to complete personal tasks for users can provide convenience, but expose a new attack vector where compromised agents perform unintended actions. Certain risks can be mitigated with proper safeguards, such as post-processing data to prevent jail-breaking, but other risks are existential in nature, and harder to address purely from a research perspective.

Our data pipeline aims to facilitate internet-scale training for agents using offline data. Training from offline data is important because current agent capabilities are low, and training them on live websites risks taking bandwidth away from real users, while polluting the internet with noisy LLM outputs. Due to these concerns, we advocate for researchers building internet-scale agents to follow the guidelines set forth in this work: (1) agents should not modify the state of the internet by making accounts, creating posts, or submitting content; (2) agents should be throttled to a maximum rate of

```
You are helping us create tasks for a web navigation system. We
    ↪ will tell you the domain of a website. You should provide a
    ↪ realistic, and specific task that a hypothetical user might
    ↪ want to accomplish on that website.

## Skipping Unsafe / Inappropriate Domains

To skip a domain, respond with 'N/A' instead of providing a task.

You should skip domains that have mature, adult, unsafe, or
    ↪ harmful content. If you are unsure whether a domain is safe,
    ↪  please skip it. In addition, skip domains that require
    ↪ logging in or creating an account, such as social media
    ↪ sites, and domains that are not intended for user-access,
    ↪ such as API endpoints and CDNs.

## Here are some domains to provide tasks for:

* `www.amazon.com`: `Find the price of the 24in LG Ultragear
    ↪ Monitor.`
* `www.wikipedia.org`: `Look up the history of the Eiffel Tower on
    ↪  Wikipedia.`

## Here are some domains to skip:

* `fbcdn.net`: `N/A`
* `api.github.com`: `N/A`

Tasks should not require external knowledge, not modify the state
    ↪ of the web, and should not require logging in or creating an
    ↪  account. For each of the following domains, provide a
    ↪ realistic, and specific task that a user could reasonably
    ↪ accomplish in a single session on the website, and limit
    ↪ your response to 20 words.
```

Figure 11: **System prompt for task generation**. We carefully design the system prompt for task generation to ensure that sites not suitable for inclusion in the training data for agents are detected and removed. This prompt ensures that proposed tasks are passive in nature, and only involve retrieving information—active tasks like making posts and creating accounts are explicitly not allowed.

interaction, and a maximum number of interactions; (3) copyrighted, private, and sensitive materials should be removed from training data.

G AGENTS.TXT & STANDARDS FOR INTERNET AGENTS

Akin to `robots.txt` directives, website creators should have a standard format to specify how internet agents are allowed to interact with their websites—if at all. Desireable controls include rate limits for interactions, limits for maximum numbers of interactions, restrictions to allow agents to interact with certain pages and not others, and restrictions on the kind of data on webpages that agents are allowed to observe (achieved via tagging elements to hide their content from agents). In addition to restricting the data available to agents, website creators should have the ability to specify locations for "playgrounds" that replicate certain key functions of their site with virtual tasks and simulated data that are intended to teach agents how to operate their site while directing traffic from agents away from their primary user-facing platform.

H More Details On Task Generation

We provide the system prompt used for task generation in Figure 11. This prompt was provided to Llama 3.1 70B, GPT-4o, and Gemini 1.5 Pro to generate tasks and filter sites unsuitable for annotation in Section 2. We carefully designed this system prompt to enforce that generated tasks are passive in nature, and do not modify the state of content on the internet. In addition to this system prompt, we employed a list of 100 hand-picked in-context examples of website URLs and appropriate tasks, which are provided in the following JSON list. When querying an LLM to generate tasks, we randomly sample 16 in-context examples from the following list, and provide only these examples to the LLM. This helps promote diverse tasks.


```
[
    {
        "domain": "archive.org",
        "task": "Identify the oldest book available in the public
            ↪ domain on this site."
    },
    {
        "domain": "arxiv.org",
        "task": "Retrieve the latest preprint paper on machine
            ↪ learning."
    },
    {
        "domain": "wikibooks.org",
        "task": "Find a freely available textbook on linear algebra
            ↪ ."
    },
    {
        "domain": "wiktionary.org",
        "task": "Get the definition and etymology of the word '
            ↪ serendipity'."
    },
    {
        "domain": "openlibrary.org",
        "task": "Locate an ebook about classic literature that is
            ↪ available for borrowing."
    },
    {
        "domain": "openculture.com",
        "task": "Find a free online course on ancient history."
    },
    {
        "domain": "theguardian.com",
        "task": "Retrieve an article discussing recent trends in
            ↪ renewable energy."
    },
    {
        "domain": "medium.com",
        "task": "Identify a highly rated blog post on productivity
            ↪ hacks."
    },
    {
        "domain": "goodreads.com",
        "task": "Find the most popular book related to neuroscience
            ↪ ."
    },
    {
        "domain": "wired.com",
```

```
        "task": "Retrieve an article about the latest advancements in wearable technology."
    },
    {
        "domain": "data.gov",
        "task": "Identify the latest government dataset on climate change."
    },
    {
        "domain": "kaggle.com",
        "task": "Find a well-documented data science competition on image recognition."
    },
    {
        "domain": "gov.uk",
        "task": "Locate the latest UK government report on healthcare."
    },
    {
        "domain": "unsplash.com",
        "task": "Find a high-resolution image of the Milky Way Galaxy."
    },
    {
        "domain": "pexels.com",
        "task": "Retrieve a popular photo tagged with 'nature'."
    },
    {
        "domain": "creativecommons.org",
        "task": "Find an article explaining Creative Commons licensing types."
    },
    {
        "domain": "pypi.org",
        "task": "Retrieve the most downloaded Python package for data analysis."
    },
    {
        "domain": "huggingface.co",
        "task": "Identify a popular machine learning model on this platform."
    },
    {
        "domain": "sciencenews.org",
        "task": "Find the most recent article on the health impacts of air pollution."
    },
    {
        "domain": "mit.edu",
        "task": "Retrieve a publicly available research paper on quantum computing."
    },
    {
        "domain": "springer.com",
        "task": "Identify the latest edition of a Springer book on robotics."
    },
    {
        "domain": "jstor.org",
```

```
        "task": "Find a research paper discussing the history of the
            ↪  Internet."
    },
    {
        "domain": "biorxiv.org",
        "task": "Retrieve the most recent bioRxiv preprint on CRISPR
            ↪  technology."
    },
    {
        "domain": "medrxiv.org",
        "task": "Find a public health preprint related to COVID-19."
    },
    {
        "domain": "commons.wikimedia.org",
        "task": "Retrieve a high-resolution image of the Eiffel
            ↪ Tower."
    },
    {
        "domain": "scholar.google.com",
        "task": "Find the most cited article by a specific
            ↪ researcher."
    },
    {
        "domain": "plos.org",
        "task": "Locate the latest research paper on gene editing
            ↪ published here."
    },
    {
        "domain": "flickr.com",
        "task": "Find a photo that has been released under a
            ↪ Creative Commons license."
    },
    {
        "domain": "datacite.org",
        "task": "Retrieve metadata for a dataset related to
            ↪ environmental studies."
    },
    {
        "domain": "orcid.org",
        "task": "Find the ORCID ID of a well-known researcher in AI
            ↪ ."
    },
    {
        "domain": "zotero.org",
        "task": "Retrieve an article discussing citation management
            ↪ tools."
    },
    {
        "domain": "github.com",
        "task": "Find the most starred repository on deep learning."
    },
    {
        "domain": "figshare.com",
        "task": "Retrieve an open dataset on climate patterns."
    },
    {
        "domain": "zenodo.org",
        "task": "Find the latest publication on open science
            ↪ practices."
```

```
    },
    {
        "domain": "worldcat.org",
        "task": "Locate a catalog entry for a rare book on botany."
    },
    {
        "domain": "biodiversitylibrary.org",
        "task": "Retrieve a scanned copy of an 18th-century
            ↪ botanical illustration."
    },
    {
        "domain": "genome.gov",
        "task": "Find the latest update on the Human Genome Project
            ↪ ."
    },
    {
        "domain": "merriam-webster.com",
        "task": "Retrieve the definition and usage of the word '
            ↪ quantum'."
    },
    {
        "domain": "stanford.edu",
        "task": "Find the most recent online lecture on artificial
            ↪ intelligence."
    },
    {
        "domain": "edx.org",
        "task": "Retrieve a TED Talk on leadership in technology."
    },
    {
        "domain": "ted.com",
        "task": "Find the latest ocean temperature data available."
    },
    {
        "domain": "noaa.gov",
        "task": "Retrieve a dataset related to consumer behavior."
    },
    {
        "domain": "data.world",
        "task": "Find a course on data visualization."
    },
    {
        "domain": "curious.com",
        "task": "Retrieve a well-cited article on the psychological
            ↪ impact of social media."
    },
    {
        "domain": "theconversation.com",
        "task": "Identify a recent research paper on biodiversity
            ↪ conservation."
    },
    {
        "domain": "nature.com",
        "task": "Retrieve the latest article on genomics research."
    },
    {
        "domain": "pnas.org",
        "task": "Find a science news article on robotics
            ↪ advancements."
```

```
    },
    {
        "domain": "sciencedaily.com",
        "task": "Identify the top story on global health issues."
    },
    {
        "domain": "bbc.com",
        "task": "Retrieve a recent podcast episode about space
            ↪ exploration."
    },
    {
        "domain": "npr.org",
        "task": "Locate the most recent update on the global
            ↪ biodiversity status."
    }
]
```

H.1 Details For Safety Experiment

This list of examples is also provided in our code release, alongside the script that we used to generate task proposals for the top 1M sites in the CommonCrawl PageRank The Common Crawl Foundation (2024). Using these prompts for task generation, we can filter our sites that are unsuitable for annotation, due to containing harmful content, or sensitive user data. To evaluate the performance of our filter, we employed a set of 100 curated websites, where 50 are manually verified as safe, and 50 are manually verified as unsafe based on the filtering conditions. These sites were chosen to span popular sites that typical annotators are likely familiar with, and less popular sites that annotators may not already be familiar with.

```
safe_sites_list = ['dhss.mo.gov', 'dizionari.corriere.it', '
    ↪ southgippsland.vic.gov.au', 'ds.iris.edu', 'lobbycontrol.de
    ↪ ', '4rsmokehouse.com', 'barnsleyfc.co.uk', 'wiwi.uni-
    ↪ wuerzburg.de', 'uplandca.gov', 'lsus.edu', 'wpcode.com', '
    ↪ webopedia.internet.com', 'tamko.com', 'premierchristian.news
    ↪ ', 'genome.jgi.doe.gov', 'burgerking.ca', 'thehugoawards.org
    ↪ ', 'radio.fm', 'thevinyldistrict.com', 'unilang.org', '
    ↪ raywhitegroup.com', 'grapevinetexas.gov', 'sanfrancisco.
    ↪ cbslocal.com', 'hyde-design.co.uk', 'breastcancerfoundation.
    ↪ org.nz', 'ludwigsburg.de', 'ignitionrobotics.org', '
    ↪ deliverit.com.au', 'kodokan.org', 'clickstay.com', '
    ↪ searchdatamanagement.techtarget.com', 'oceanario.pt', '
    ↪ wentworthpuzzles.com', 'catholicworldreport.com', 'quizlet.
    ↪ com', 'innovation.nhs.uk', 'synonyms.reverso.net', 'news.
    ↪ siemens.co.uk', 'readability-score.com', 'co.modoc.ca.us', '
    ↪ cityofmyrtlebeach.com', 'loire.gouv.fr', 'lawphil.net', '
    ↪ saem.org', 'parmigianoreggiano.it', 'engaging-data.com', '
    ↪ itf-tkd.org', 'aka.education.gov.uk', 'ub.uni-kl.de', '
    ↪ mottchildren.org']

unsafe_sites_list = ['refer to the code release']
```

We provide the list of safe sites in the preceding code block, and in the code release, but we provide the list of unsafe sites in just the code release to keep this manuscript clean of harmful content. Using these lists, we conduct an experiment measuring the accuracy, precision, and recall of our safety filter for detecting sites that are not suitable for training agents. In this experiment, we run the task proposer with the same prompts used to scale annotation the top 1M sites in the CommonCrawl PageRank, and we consider a site to be marked positive for unsafe content if the task proposer generates "N/A" rather than predicting a task. To calculate the human baseline for this experiment, we showed participants the system prompt observed by the task proposer, and asked participants to label sites as safe or unsafe using this per-example prompt:

```
human_safety_prompt = "Is this website safe to include?\n\n{
    ↪ host}\n\nenter 'y' for yes / safe, 'n' for no / unsafe: "
```

Human participants were not allowed to visit the URL shown, and had to determine whether the site is safe for annotation purely from their prior knowledge (the same conditions faced by the task proposer). The 100 sites for the safety experiment were shuffled into a uniformly random order to ensure the order of annotation did not bias the human annotators predictions. One human participant was used to obtain the human baseline result in Table 2.

H.2 Details For Reliability Experiments

Similar to the previous safety experiment, we employed human participants to obtain a human baseline for task feasibility. In particular, we showed human participants the system prompt in Figure 11 for the task proposer, and had them write a task for each of the following websites without visiting the URL (the same conditions faced by the task proposer). The following 100 sites were shuffled into a uniformly random order to ensure the participants were not influenced by the order in which sites were shown. After tasks were proposed by participants, and by LLMs, we evaluated the expert feasibility of tasks by manually attempting to complete the tasks proposed by each set of participants, and marking tasks as feasible, or not feasible based on our own ability to complete them. In total, we annotated 400 tasks, which required 8 hours of annotation. One human participant was used to obtain the human baseline result in Table 3.

```
reliability_sites_list = ['godaddy.com', 'chrome.google.com', '
    ↪ apple.com', 'support.cloudflare.com', 'support.apple.com', '
    ↪ edition.cnn.com', 'go.microsoft.com', 'google.de', 'w3.org',
    ↪  'yandex.ru', 'bfdi.bund.de', 'microsoft.com', 'apps.apple.
    ↪ com', 'networksolutions.com', 'support.mozilla.org', 'yelp.
    ↪ com', 'cnn.com', 'ec.europa.eu', 'developer.mozilla.org', '
    ↪ icann.org', 'books.google.com', 'globenewswire.com', '
    ↪ onlinelibrary.wiley.com', 'gnu.org', 'slideshare.net', '
    ↪ metacpan.org', 'porkbun.com', 'oag.ca.gov', 'spiegel.de', '
    ↪ linuxfoundation.org', 'help.opera.com', 'mayoclinic.org', '
    ↪ podcasts.apple.com', 'nhs.uk', 'addons.mozilla.org', 'google
    ↪ .fr', 'pewresearch.org', 'finance.yahoo.com', 'weforum.org',
    ↪  'g2.com', 'savethechildren.org', 'news.com.au', 'biblia.com
    ↪ ', 'yr.no', 'engadget.com', 'microsoftstore.com', 'ema.
    ↪ europa.eu', 'theintercept.com', 'princeton.edu', '
    ↪ foodandwine.com', 'sfgate.com', 'voguebusiness.com', '
    ↪ ourworldindata.org', 'livingwage.org.uk', 'cms.law', '
    ↪ msdmanuals.com', 'websitesetup.org', 'support.xbox.com', '
    ↪ treehugger.com', 'tripadvisor.com.pe', 'mondragon.edu', '
    ↪ greenparty.ca', 'aaojournal.org', 'restaurantpassion.com', '
    ↪ iwillteachyoutoberich.com', 'moneyconvert.net', '
    ↪ gesundheitsinformation.de', 'ovc.uoguelph.ca', 'zdnet.be', '
    ↪ oxfordamerican.org', 'snackandbakery.com', 'journals.uic.edu
    ↪ ', 'confused.com', 'standards.globalspec.com', '
    ↪ onlyinyourstate.com', 'ahsgardening.org', 'wyze.com', '
    ↪ nornickel.ru', 'viessmann.fr', 'benetton.com', 'firecomm.gov
    ↪ .mb.ca', 'executedtoday.com', 'eukn.eu', 'fraeylemaborg.nl',
    ↪  'verizon.com/about/news-center', 'orthodoxalbania.org', '
    ↪ cheapjoes.com', 'bake-eat-repeat.com', '
    ↪ plattformpatientensicherheit.at', 'hifinews.com', '
    ↪ cellsignal.com', 'thenotariessociety.org.uk', 'chosenfoods.
    ↪ com', 'westerndressageassociation.org', 'pridesource.com', '
```

```
You are a helpful scientific assistant categorizing tasks on the
    ↪ web. You will observe a domain and web navigation task, and
    ↪ you should provide a concise categorization of the task in 3
    ↪  words or less. For example, if the domain is "google.com"
    ↪ and the task is "find a recipe for mashed potato", you may
    ↪ categorize the task as "recipe search".

## Task Format

Here is the format for the task:

[domain]: [task]

Here is what each part means:

`[domain]`: The domain of the website you are observing.
`[task]`: The task a user is trying to accomplish on the website.

## Response Format

Respond with a category name for the task in 3 words or less, and
    ↪ provide only the category name, do not provide an
    ↪ explanation or justification for the categorization.

Here is the next task, please follow the instructions carefully.
```

Figure 12: **System prompt for task categorization**. We employ *Llama 3.1 70B* to automatically label task categories for our dataset of 150k web navigation tasks. We prompt the LLM to assign categories in 3 words or less, and set the sampling temperature to 0.5 to encourage predictions to use more consistent language. Using these categories, we seek to understand agent performance by category.

```
    ↪ northtacomapediatricdental.com', 'strade-bianche.it', '
    ↪ pvdairport.com', 'institute.sandiegozoo.org', 'raintaxi.com
    ↪ ']

human_reliability_prompt = "\n\n{host}\n\nenter a task, or respond
    ↪  with 'N/A' instead: "
```

H.3 Automatic Task Categorization

To better understand the statistics of generated tasks, we employ *Llama 3.1 70B* to assign task categories. We prompt *Llama 3.1 70B* with the system prompt in Figure 12 to assign a category in 3 words or less to encourage simple categories. Categories have 16.9 tasks on average, and 953 categories have more than the mean, while 7741 have less than the mean. There is occasional overlap between categories, which can be observed in Figure 13, but for the purposes of understanding performance by category, overlap is acceptable provided categories have sufficiently large numbers of tasks, and performance per category can be accurately calculated. We provide our task categorization script in the official code release.

I Understanding Agent Capabilities & Limitations

To complete the analyses presented in Section 3, we explore the categories of tasks that agents succeed at most frequently. Shown in Figure 14, we plot the average judge success probability prediction $\mathbf{r}_T$ versus task category for the top 70 most successful categories that have at least 100 tasks assigned to them. Based on the figure, top categories include search for *contact information*, finding

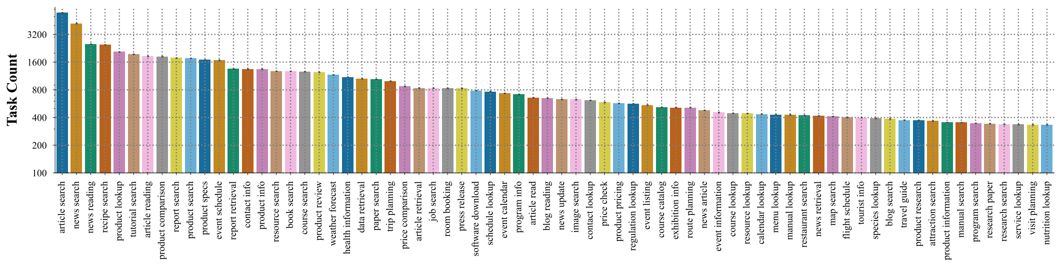

Figure 13: **Largest categories for internet-scale task generation**. We assign categories to 150k web navigation tasks generated by our pipeline in Section 2, and visualize the number of tasks for each of the largest 70 categories. Top categories include *article search*, *news search*, *recipe search*, *product lookup*, and more. The top 12 task categories have more than 1600 tasks assigned to each of them, the mean number of tasks per category is 16.9, and 89% of categories (7741 in total) have fewer than the mean number of tasks.

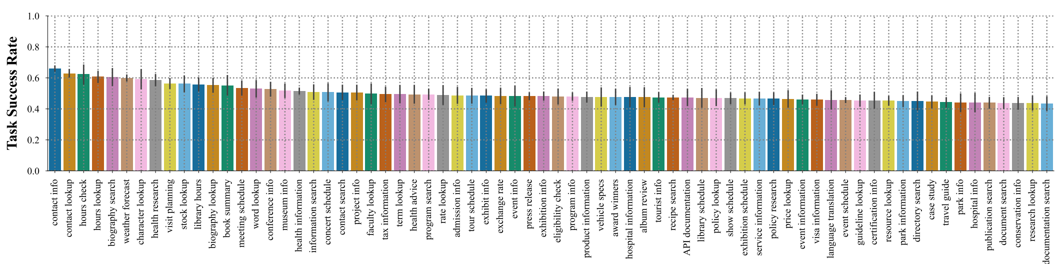

Figure 14: **Most successful categories for internet-scale task generation**. We explore the rates of task completion for the top categories of tasks generated by our pipeline. We restrict our focus to categories where at least 100 tasks are assigned, and plots the success rates for the top 70 of such categories. Results show that 22 categories are solved with more than a 50% rate with agents based on *Llama 3.1 70B*.

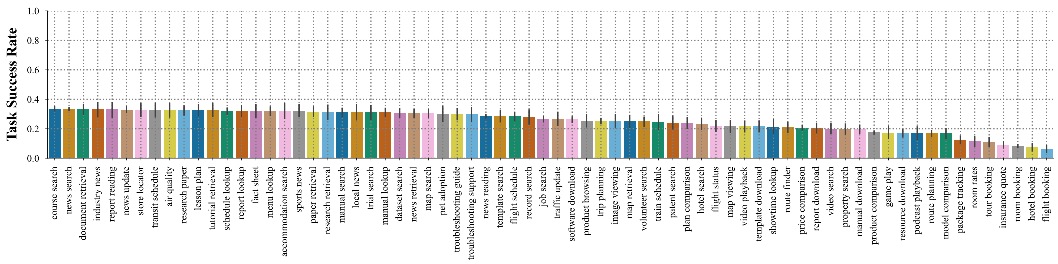

Figure 15: **Least successful categories for internet-scale task generation**. Similar to the previous figure, we explore the rates of task completion for the bottom 70 categories that have at least 100 tasks assigned to them. While the majority of the least successful categories have success rates greater than 20%, performance drops as low as 5%. Many of the categories shown in the plot above involve actions that are not feasible given the current limitations of the Playwright API, and may be possible in future work that extends agents to a fully-operable virtual computer environment. In addition, better LLM backbones are likely to improve performance.

hours of operation, looking up *biographical information*, obtaining current *weather forecasts*, and conducting *health research*. Based on these results, the top 22 categories are solved with more than a 50% rate using agents based on *Llama 3.1 70B* running zero-shot. As stronger models are developed, the success rates for agents running in our pipeline are likely to improve, and the quality of the data we generate will jointly improve.

In addition to studying the best-performing categories, we also explore the limitations of current agents via their least successful categories. Shown in Figure 15, we select the bottom 70 categories in terms of their average judged success probability for categories with at least 100 tasks assigned to them. Many of these categories require agents to remember and reason about previous interactions, such as the *product comparison* category. For this category, an agent must review several products, and compare their details from memory. In these cases, access to a note-taking app may improve performance. Additionally, certain task categories involve requests that are not feasible given the limitations of the Playwright API, including categories for *downloading reports / manuals*, and *opening and playing files*. While these tasks are not currently feasible, providing agents with a fully-operable virtual computer environment could unlock these abilities in future work.

J AGENT & JUDGE SYSTEM PROMPTS

We provide the system prompt used with our agent below. This prompt is released in our official code, alongside the observation processor that maps webpage DOM to a compact markdown format, referenced in the system prompt.

````
You are a helpful assistant operating my web browser. I will show
    ↪ you webpages formatted in markdown, and I want your help to
    ↪ complete a web navigation task. Read the webpage, and
    ↪ respond with an action in JSON to interact with the page,
    ↪ and help me complete the task.

## Formatting The Response

Respond with actions in the following JSON schema:

```json
{
   "action_key": str,
   "action_kwargs": dict,
   "target_element_id": int
}
````
````

````
```

Here is what each key means:

- `action_key`: The action to perform.
- `action_kwargs`: Named arguments for the action.
- `target_element_id`: The id of the element to perform the action on.

## Available Actions

I'm using playwright, a browser automation library, to interact with the page. I'm parsing the value assigned to `action_key` into a method call on the page object, or an element object specified by the value assigned to `target_element_id`. Here are the available actions:

### Click Action Definition

- `click`: Click on an element specified by `target_element_id`.

### Example Click Action

Suppose you want to click the link `[id: 5] Sales link`:

```json
{
   "action_key": "click",
   "action_kwargs": {},
   "target_element_id": 5
}
```

### Hover Action Definition

- `hover`: Hover over an element specified by `target_element_id`

### Example Hover Action

Suppose you want to hover over the image `[id: 2] Company Logo image`:

```json
{
   "action_key": "hover",
   "action_kwargs": {},
   "target_element_id": 2
}
```

### Fill Action Definition

- `fill`: Fill an input element specified by `target_element_id` with text.
   - `value`: The text value to fill into the element.

### Example Fill Action
````
````

````
Suppose you want to fill the input `[id: 13] "Name..." (Enter your name text field)` with the text `John Doe`:

```json
{
   "action_key": "fill",
   "action_kwargs": {
      "value": "John Doe"
   },
   "target_element_id": 13
}
```

### Select Action Definition

- `select`: Select from a dropdown element specified by `target_element_id`.
   - `label`: The option name to select in the element.

### Example Select Action

Suppose you want to select the option `red` from the dropdown `[id: 67] "blue" (select a color dropdown)`:

```json
{
   "action_key": "select_option",
   "action_kwargs": {
      "label": "red"
   },
   "target_element_id": 67
}
```

### Go Back Action Definition

- `go_back`: Go back to the previous page (`target_element_id` must be null).

### Example Go Back Action

```json
{
   "action_key": "go_back",
   "action_kwargs": {},
   "target_element_id": null
}
```

### Goto Action Definition

- `goto`: Navigate to a new page (`target_element_id` must be null).
   - `url`: The URL of the page to navigate to.

### Example Goto Action

Suppose you want to open google search:
````

````
```json
{
   "action_key": "goto",
   "action_kwargs": {
      "url": "https://www.google.com"
   },
   "target_element_id": null
}
```

### Stop Action Definition

- `stop`: Stop the browser when the task is complete, or the answer is known.
   - `answer`: Optional answer if I requested one.

### Example Stop Action

```json
{
   "action_key": "stop",
   "action_kwargs": {
      "answer": "I'm done!"
   },
   "target_element_id": null
}
```

Thanks for helping me perform tasks on the web, please follow the instructions carefully. Start your response with an explanation in 50 words, and choose exactly one action you would like to perform.
````

We also provide the system prompt used with out LLM judge. The system prompt instructs the judge to predict a json-formatted dictionary that contains a "success" key, and an "on_right_track" that represent the estimated probability that the task is successful, and that the agent is on the right track towards solving the task, respectively. These distinctions are adapted from Koh et al. (2024b), and help us filter for high-quality training data by distinguishing trajectories that were solved by the agent's own actions from trajectories that were solved by chance.

````
You are a helpful assistant providing feedback on a web automation script. I will show you a list of previous actions, the current webpage formatted in markdown, and the proposed next action. I want your help evaluating the proposed action, to determine if the desired task is complete, or if we are on the right track towards future completion.

## Reading The Action Schema

You will see actions in the following JSON schema:

```json
{
   "action_key": str,
   "action_kwargs": dict,
   "target_element_id": int
}
```
````

~~~
Here is what each key means:

- 'action_key': The action to perform.
- 'action_kwargs': Dictionary of arguments for action.
- 'target_element_id': The id of the element to perform the action on.

## Available Actions

I'm using playwright, a browser automation library, to interact with the page. I'm parsing the value assigned to 'action_key' into a method call on the page object, or an element specified by the value assigned to 'target_element_id'. Here is an example action:

### Example Click Action

Here is an example where the script clicked the link '[id: 5] Sales link':

```json
{
  "action_key": "click",
  "action_kwargs": {},
  "target_element_id": 5
}
```

### Example Select Action

Here is an example where the script selected the option 'red' from the dropdown '[id: 67] "blue" (select a color dropdown)':

```json
{
  "action_key": "select_option",
  "action_kwargs": {
    "label": "red"
  },
  "target_element_id": 67
}
```

### Example Goto Action

Here is an example where the script opened google search:

```json
{
  "action_key": "goto",
  "action_kwargs": {
    "url": "https://www.google.com"
  },
  "target_element_id": null
}
```

### Example Stop Action
~~~

```
Here is an example where the script stopped with the message "I'm
    ↪ done!":

```json
{
   "action_key": "stop",
   "action_kwargs": {
      "answer": "I'm done!"
   },
   "target_element_id": null
}
```

Formatting The Response

Think step by step, and start your response with an explanation of
    ↪ your reasoning in 50 words. Then, provide an evaluation in
    ↪ the following JSON schema:

```json
{
   "success": float,
   "on_right_track": float,
}
```

Here is what each key means:

- `success`: What is the probability the desired task has been
    ↪ completed successfully, rated from 0.0 (not possible) to 1.0
    ↪ (absolutely certain)?
- `on_right_track`: What is the probability the script is on the
    ↪ right track towards a future success, rated from 0.0 (not
    ↪ possible) to 1.0 (absolutely certain)?

Thanks for helping me evaluate the script, please follow the
    ↪ instructions carefully. Start your response with a step by
    ↪ step explanation. Then, provide an evaluation in the JSON
    ↪ schema above.
```

K DETAILS FOR TRAINING AGENTS

To understanding the utility of the generated data for training agents, we filter the data, and compare our filtered data to human demonstrations on the Mind2Web benchmark Deng et al. (2023). In particular, we sweep over different sizes of random subsets of human actions, from 32 to 256, which helps us understand the value of synthetic data generated from the InSTA pipeline versus different scales of human data. We then fine-tune models based on `google/flan-t5-large` from HuggingFace. We employ identical training hyperparameters to those used in Deng et al. (2023) to ensure that our results are directly comparable to previous work. Results in Section K report performance on the official `test_website` split of Mind2Web, where agents are tested on previously unobserved websites.

In order to prepare our data, we employ three filtering rules. In the first rule, we filter for data where the agent was predicted to have succeeded at the task with `conf` = 1, and was predicted to be on the right track with `conf` = 1. This filtering rule is motivated by our findings in Section 3, where we found that our LLM judge based on *Llama 3.1 70B* has an accuracy up to 93.1% at detecting successful trajectories for its predictions with `conf` = 1. Filtering based on both "success" and "on_right_track" conditions is essential to obtain data where the agent directly caused the task to be solved, rather than the task being solved by external conditions. The next filtering rule we use is to select trajectories with at least three actions, which helps create training data that is not too easy (i.e. not solved after just one or two actions). Finally, we select tasks where the agent did not encounter any errors during execution. These include being presented with server errors such as 404 Not Found, and 403 Forbidden, encountering a captcha, and being blocked, even if just temporary, from the target website. These filtering steps produce an automatically curated set of $7,463$ demonstrations from our pipeline where agents successfully completed tasks generated by the InSTA pipeline. We reserve 500 demonstrations from this pool for our test set, and the rest for training agents in Figure 9. The original Mind2Web dataset contains $2,350$ tasks.

L Additional Related Works

While writing this paper, concurrent work was released that introduces a Proposer-Agent-Evaluator framework for web navigation agents Zhou et al. (2024c). There are several key differences between our work and theirs, and the most important difference is scale. We generate tasks for 1M sites on the internet, whereas their work considers just 85 real-world sites, 5 sites from WebArena Zhou et al. (2024b), and 13 sites from WebVoyager He et al. (2024). The second difference is evaluation. Safety and reliability play crucial roles when gathering data, and we conduct an analysis on the safety and reliability of data generated by our method on 100 real-world sites. Another major difference pertains to offline learning. Offline learning should be used when scaling agents because current agent capabilities are low, and training them online risks polluting the internet with noisy LLM outputs, while taking bandwidth away from real users. The final difference pertains to the train-test split. We train agents on diverse internet data, and transfer to target benchmarks, while the agents presented in Zhou et al. (2024c) train on sites from target benchmarks using synthetic tasks. Our train-test split is stronger, and evaluates the ability for agents trained on our synthetic data to generalize to novel websites, domains, and tasks.

M Hyperparameters

We provide a list of the hyperparameters used in this work in Table 1. Values are selected to mirror prior work in synthetic data Trabucco et al. (2024), and to employ standard hyperparameters for training agents on Mind2Web Deng et al. (2023).

N Cost Analysis For Llama 3.1 70B

To better contextualize why using *Llama 3.1 70B* is important for a project at this scale, we analyze the number of tokens processed by the LLM, and compute an expected cost if this were served using proprietary models. As the analysis shows, using *Llama 3.1 70B* is most feasible option for running agents at this large scale, and results in the paper show that this choice of LLM backbone does not compromise in accuracy and performance. We have deep gratitude for the Llama team at Meta working to make developments in language modeling available to the research community.

Hyperparameter Name	Value
OpenAI API Model Name	`gpt-4o`
Google API Model Name	`gemini-1.5-pro`
Llama HuggingFace Model Name	`meta-llama/Llama-3.1-70B-Instruct`
CommonCrawl PageRank Revision	`cc-main-2024-apr-may-jun-host-ranks.txt.gz`
Number of sites before filtering	$1,000,000$
Number of tasks after filtering	$146,746$
Max Tokens Per Observation	$4,096$
Max Tokens Per Action	$2,048$
Max Tokens Per Judgement	$2,048$
Max Tokens Per Task	64
Max Observations Per Agent Context	5
Max Actions Per Agent Context	5
Max Observations Per Judge Context	1
Max Actions Per Judge Context	5
OpenAI Inference API Sampling Temperature	0.5
OpenAI Inference API Sampling Top P	1.0
Mind2Web HuggingFace Model Name	`google/flan-t5-large`
Mind2Web Training Epochs	5
Mind2Web Batch Size	32
Mind2Web Learning Rate	`5e-5`
Mind2Web Filtering Success Threshold	1.0
Mind2Web Filtering On Right Track Threshold	1.0
Mind2Web Filtering Similarity Threshold	0.5
Mind2Web Synthetic Data Mixing Ratio	50%

Table 1: **Hyperparameters used in our paper.** We organize hyperparameters into five sections, including names of language model backbones, parameters of the data generation pipeline, sampling parameters for the OpenAI inference API, training parameters used by the Mind2Web benchmark, and filtering parameters used to prepare our data for the Mind2Web benchmark.

Variable Name	Value
Number of tasks	$146,746$
Max tokens per observation	$4,096$
Max observations per agent context window	5
Typical agent / judge response size	128
Max tokens per system prompt	$1,024$
Max steps per task	10
Tokens processed by the agent	$146,746 * ((4,096 * 5 + 1,024 + 128) * 10) = 31,744,094,720$
Tokens processed by the judge	$146,746 * (4,096 + 1,024 + 128 * 10) = 939,174,400$
Total tokens processed	$32,683,269,120$
Expected API cost for *GPT-4o*	$ $163,416.35$
Expected API cost for *Gemini 1.5 Pro*	$ $228,782.88$
Expected AWS compute cost for serving *Llama 3.1 70B* (14 days for two 8-gpu v100 spot instances)	$ $6,622.56$
Percent saved using *Llama 3.1 70B*	$[95.9, 97.1]$ %

Table 2: **Cost analysis for different LLM models in the fully-scaled pipeline.** This table provides statistics for the number of tokens that were processed by our pipeline, and why serving using a local LLM engine like vLLM is important for bringing down costs.