CowPilot: A Framework for Autonomous and Human-Agent Collaborative Web Navigation

Faria Huq Zora Zhiruo Wang Frank F. Xu Tianyue Ou Shuyan Zhou
Jeffrey P. Bigham† **Graham Neubig**†
School of Computer Science, Carnegie Mellon University
{fhuq, jbigham, gneubig}@cs.cmu.edu
†Equal Supervision

Abstract

Web agents capable of conducting web tasks on user behalf have gained attention for their task automation potential. However, they often fall short on complex tasks in dynamic, real-world contexts, making it essential for users to work collaboratively with the agent. We present CowPilot, a framework supporting both autonomous and human-agent collaborative web navigation, and evaluation across task success, user experience, and task efficiency. CowPilot eases human effort by starting with agents proposing next steps, meanwhile allowing humans to execute, pause, or reject agent-proposed steps and take alternative actions instead; supporting seamless action-taking between agent and human participants. We conduct case studies on five websites and find that human-agent collaborative mode achieves the highest 95% success rate while requiring humans to perform only 15.2% of the total steps. Even with human interventions during task execution, the agent successfully drives up to half of task success on its own. CowPilot serves as a useful tool for data collection and agent evaluation across websites, which we hope will facilitate further advances. Video demonstrations are available at https://oaishi.github.io/cowpilot.html.

1 Introduction

Agents supported by large language models (LLMs) have become increasingly capable of automating digital tasks such as web navigation (Zhou et al., 2023; Deng et al., 2024). While existing frameworks for web agents mostly focus on solo, autonomous agents (Zheng et al., 2024b; Iong et al., 2024; Drouin et al., 2024), we identify the need for users to interact with the LLM agent for varied purposes such as supervision and collaboration, i.e., the *copilot* mode. While existing frameworks (Lù et al., 2024; Drouin et al., 2024; Wang et al., 2024a; Zheng et al., 2024b) mainly support users communicating with agents via natural language (NL) feedback, or recording actions of human users alone (Pan et al., 2024b), they do not support dynamic human-agent collaboration during a task, where humans and LLM agents alternate actions to recover from mistakes. We ask: *How can we enable human-agent collaborative task-solving?* and further, *How do agents perform under such dynamic settings?*

To help answer these questions, we introduce CowPilot (§2), a lightweight framework that can be seamlessly integrated into user web activities as a Chrome extension. CowPilot starts with the LLM agent *suggest*ing actions for human's approval, meanwhile allowing human to *pause* or reject the agent-suggested actions and take alternative ones to drive the process; human can also choose to *resume* the agent-driven process at any time to ease the effort (§2.1). To systematically evaluate this collaborative process, we propose several metrics for task accuracy, user experience, and efficiency aspects (§2.2).

Beyond agent web automation, CowPilot enables a wide range of use cases (§3), including: web automation (§3.1), data collection including agent trajectory and user feedback (§3.2) as well as evaluations for single or multiple agents (§3.3).

We conduct studies on five websites across shopping, social, and technical domains (§4), and show CowPilot collaborative mode achieves higher success rates over autonomous agents (by 47%) and even human-only settings (by 6%), with the LLM agent taking 84.8% of the steps and drive up to half of the task successes. These results suggest the great potential for accuracy and efficiency improvement with copilot agents.

Overall, CowPilot showcases the great potential of human-agent collaborative web navigation, and serves as a useful tool for future web automation, data collection, and evaluation research.

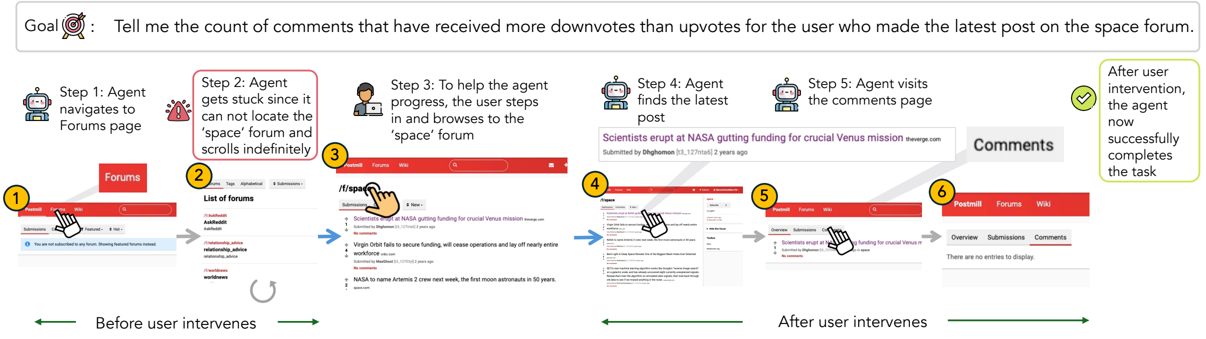

Goal : Tell me the count of comments that have received more downvotes than upvotes for the user who made the latest post on the space forum.
Step 1: Agent navigates to Forums page
Step 2: Agent gets stuck since it can not locate the 'space' forum and scrolls indefinitely
Step 3: To help the agent progress, the user steps in and browses to the 'space' forum
Step 4: Agent finds the latest post
Step 5: Agent visits the comments page
After user intervention, the agent now successfully completes the task
1
2
3
4
5
6
Forums
Postmill
Forums
Wiki
List of forums
/f/AskReddit
AskReddit
/f/relationship_advice
relationship_advice
/f/worldnews
worldnews
/f/space
Scientists erupt at NASA gutting funding for crucial Venus mission theverge.com
Submitted by Dhghomon [t3_127nta6] 2 years ago
No comments
Virgin Orbit fails to secure funding, will cease operations and lay off nearly entire workforce cnbc.com
NASA to name Artemis 2 crew next week, the first moon astronauts in 50 years.
Comments
Overview
Submissions
Comments
There are no entries to display.
Before user intervenes
After user intervenes

Figure 1: A step-by-step illustration of how human intervention enables the agent to overcome a failure point during task execution. The figure uses gray edges to represent the agent's autonomous actions and blue edges to indicate human intervention. The process begins with the agent attempting the task independently (Step 1) and navigating to the interface to list available forums (Step 2). At this stage, the agent gets stuck, unable to locate the desired 'space' forum. A human intervenes (Step 3), guiding the agent to the correct forum. The user then resumes the agent's operation (Step 4), allowing it to retrieve the required post and complete the task by navigating to the comments section (Step 5)."

2 CowPilot

In this section, we introduce the CowPilot framework (§2.1) and evaluation metrics for task accuracy and collaboration quality (§2.2).

2.1 The CowPilot System

Given an objective o stated in natural language (NL) (e.g., *book a flight*) for the web environment, we define two agents: one agent instantiated with an LLM policy $\mathcal{L}$, and one human agent $\mathcal{H}$. At each time step t, based on the observation o_t from the environment state s_t, either the LLM agent or human agent generates an action a_t, formalized as $a_t = \mathcal{L}(t, o_t, a_{0:t-1})$. Executing a_t on the environment results in a new state s_{t+1} that gives observation o_{t+1} that drives the next step. The two agents collectively generate a sequence of actions $a_{0:n}$ over n steps, until it reaches a task termination condition, e.g., output `STOP` or a maximum number of steps. By default, the LLM agent starts generating actions $a_t^{\mathcal{L}}$ from $t = 0$, defined in Table 1, unless intervened by the human agent $\mathcal{H}$.

Actions taken by the human agent $\mathcal{H}$ are critical to optimizing CowPilot's decision-making pipeline. When the human agent intervenes, they provide contextual feedback by identifying and correcting prior mistakes made by the LLM agent. This redirection helps the agent recover from a suboptimal path and proceed with a more viable course of action.At the same time, by integrating human actions into its action history, CowPilot ensures the LLM agent is aware of human corrections since its last decision, preventing redundant actions and enabling efficient task progression. To ensure effective integration of these human actions, CowPilot incorporates the following core modules:

Suggest-then-Execute under Human Supervision At any time step, the human agent $\mathcal{H}$ can decide to take over by generating $a_t^{\mathcal{H}}$. More concretely, the LLM agent $\mathcal{L}$ generates an action a_i and presents it as a *suggestion* for the tentative next step to the user (Figure 2), which includes a visual indicator highlighting the target element for the proposed action, accompanied by a textual explanation of the agent's reasoning. This tentative step is presented to the human agent for at most five seconds, and is automatically executed if the human agent does not oppose. Otherwise, the human agent can choose to *reject* or *pause* the action, and then take over to produce action. They can also transfer the action back to the LLM-based agent by hitting the *resume* button. This take-over-then-back process can be conducted unlimited times per task-solving session. This mechanism balances operational efficiency with user oversight, allowing users to intercept potential errors without the burden of manually approving every step.

Pause LLM Agent: Extract Human Actions Whenever the human agent $\mathcal{H}$ rejects the LLM-proposed action, our CowPilot system starts tracking human activity on the websites, particularly what webpages and UI elements they interact with. To capture this micro-level metadata, we utilize HTML event listener[1], that are attached to the interactive elements (e.g., text field, buttons, drop-

[1] https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Action	Raw Human Action	Description
`click(elem)`	`click`	Click on a webpage element using the mouse.
`hover(elem)`	`mouseover`	Hover the mouse over an element without clicking it.
`type(elem, text)`	`input`	Enter text into a text area or text box.
`scroll(dir)`	`wheel`	Scroll the webpage up/down/left/right.
`goto(url)`	`Tabs.onUpdated`	Navigate to a specific URL.
`goto(tab)`	-	Navigate to a specific tab.
`finishwithanswer(text)`	-	For information retrieval task, terminate the task with retrieved textual information.
`finish()`	-	Mark the task as completed.
`failure()`	-	Mark the task as failed and uncompleted.

Table 1: Action space of agents in CowPilot. LLM agent supports all actions in the *Action* column. Human actions are captured by entries in the *Raw Human Action* column and transformed into *Action*s.

down menu) in the current webpage and triggered each time the elements are accessed by the user (Figure 2).

Note that, actions captured by the HTML event listener can include noisy actions irrelevant to the task (Siqueira and Baldochi, 2018; Cheng and Kumar, 2015), such as unintentional mouseover events. Hence, we transform the listener-captured actions to the LLM agent actions space, from *Raw Human Action* to *Action* column in Table 1, which are also associated with proper textual descriptions that can help the agent interpret user inputs effectively. To facilitate this transformation, we use an off-the-shelf LLM (in this case, `GPT-4o-2024-08-06`) and provide the raw human actions as input. The model outputs the transformed and cleaned version of the actions. (The exact prompt for this transformation is shown in §A.1.)

Resume LLM Agent: Predict next Action using Human Input If the human agent chooses to *resume* the LLM agent at any given step, our CowPilot stops tracking human actions and restarts LLM agent generation (Figure 2). Note that the LLM agent has access to all previous actions generated by itself and the human agent.

2.2 Evaluation Metrics

To evaluate the agent performance in CowPilot, we report general agent task success. In addition, to better quantify human-agent collaboration, we introduce five evaluation metrics to measure various aspects throughout task execution.

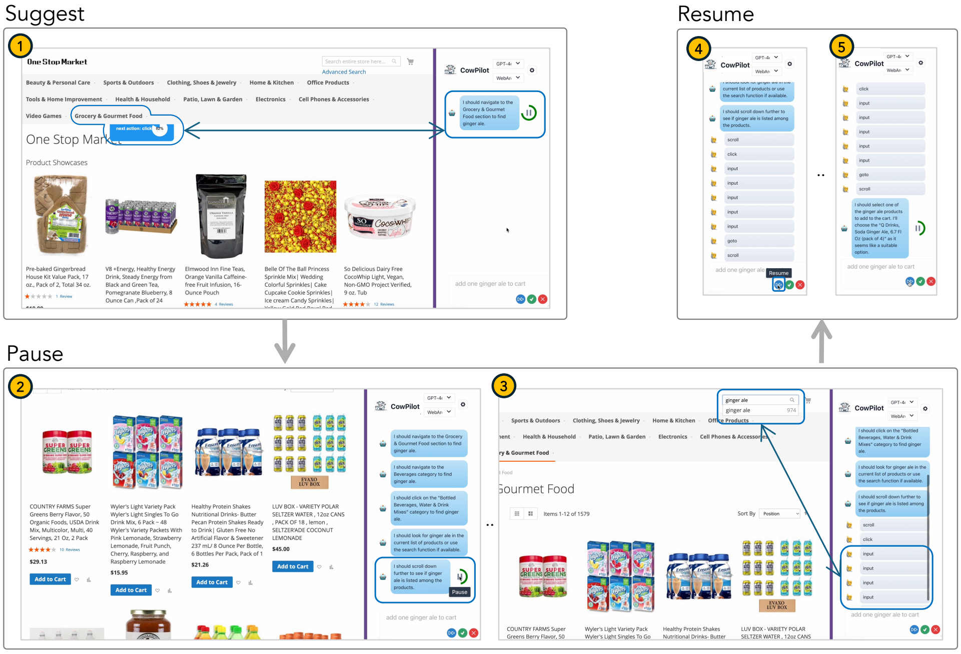
General Task Success To measure generic task success, we measure *end-to-end task accuracy*, which measures if the task objective is achieved after the agent task-solving process. At the end of the task, the agent self-marks its success or failure by generating a `finish/failure` action highlighted in Table 1. Optionally, the user can overwrite the result if they disagree with the agent's self-evaluation.

Human-Agent Collaboration To measure how human and agent interacted with each other throughout the task execution, we first measure the engagement of both parties, by: (1) *Agent step count*: How many steps are taken by the agent per task; (2) *Human step count*: How many steps are taken by the human per task; (3) *Total step count*: the sum of steps taken by human and agent.

Meanwhile, we measure agent capabilities via (4) *Human intervention count*: How many times does the user pause the agent to take actions themselves. Note that a single intervention may involve multiple steps performed by the human, as the intervention continues until the agent resumes. A higher value potentially suggests that the agent made frequent errors and users had to step in to resolve the mistakes; and (5) *Agent-driven Completion Accuracy*: Measures how many tasks were successfully completed by the agent, i.e., the terminating step was taken by the agent. A higher value indicates the agent's ability to recover and complete tasks autonomously after human intervention, whereas a lower value reflects its dependency on human assistance.

3 Usecases of CowPilot

Our CowPilot unveils numerous potential use cases. We particularly highlight three use cases under the scope of this work.



1: I should navigate to the Grocery & Gourmet Food section to find ginger ale.

add one ginger ale to cart

2: I should navigate to the Grocery & Gourmet Food section to find ginger ale.

I should navigate to the Beverages category to find ginger ale.

I should click on the "Bottled Beverages, Water & Drink Mixes" category to find ginger ale.

I should look for ginger ale in the current list of products or use the search function if available.

I should scroll down further to see if ginger ale is listed among the products.

add one ginger ale to cart

3: I should click on the "Bottled Beverages, Water & Drink Mixes" category to find ginger ale.

I should look for ginger ale in the current list of products or use the search function if available.

I should scroll down further to see if ginger ale is listed among the products.

scroll
click
input
input
input
input

add one ginger ale to cart

4: current list of products or use the search function if available.

I should scroll down further to see if ginger ale is listed among the products.

scroll
click
input
input
input
input
input
goto
scroll

5: click
input
input
input
input
input
goto
scroll

I should select one of the ginger ale products to add to the cart. I'll choose the "Q Drinks, Soda Ginger Ale, 6.7 Fl Oz (pack of 4)" as it seems like a suitable option.

add one ginger ale to cart

Figure 2: Example of CowPilot's core interaction modules during task execution. At step 1, the LLM agent generates a ***suggestion***, highlighting the textual description and the UI element where the action will be performed. At step 2, the user identifies an erroneous action, chooses to ***pause*** the LLM agent, and proceeds to perform corrective actions manually (step 3, e.g., typing in the textfield, highlighted in blue). At step 4, the user chooses to ***resume*** the LLM agent, allowing it to continue generating actions. The agent resumes successfully and proceeds to execute subsequent steps autonomously (step 5).

3.1 Web Automation

CowPilot can be a standalone agent framework to automatically conduct web tasks for end users. CowPilot is implemented as a Chrome extension where all computations other than the LLM calls are handled locally with minimal storage requirement (<50MB). Any users can easily install CowPilot with just four clicks and use it with their personal API key. We use LiteLLM[2] proxy server for our backend LLM, enabling CowPilot to support all available models, including GPT4 and Llama. Depending on whether the user wants to participate in task-solving, our agent can operate in two modes: 1) *Fully autonomous mode*: The agent conducts a user-issued task start-to-end; 2) *CoPilot mode*: Human and agent collaboratively solve a task, which is useful for complex tasks where the agent is more prone to make mistakes.

3.2 Data Collection from Websites

CowPilot can also be used as a data annotation tool to collect task trajectories across any website accessible via the Chrome browser. Deployed as a Chrome extension, CowPilot requires no additional setup and supports both simulated and self-hosted websites.

We can track all actions conducted by LLM agents and humans. Beyond human and LLM agent actions, CowPilot also collects human action feedback at both (i) step-level: whether the user judges the current step correctly leads to task success, and (2) task-level: whether the entire trajectory correctly solves the task. These rich data collections can easily facilitate various studies such as user behavior studies and advanced agent learning strategies.

3.3 Evaluation and Comparative Analysis of Agent Performance

CowPilot can be used to evaluate and compare agent performance. We support a wide range of open-weight and closed-source models served via LiteLLM. While this paper focuses on comparing GPT and LLaMA, the framework can easily extend to other open and closed-source models.

To evaluate a particular model, the user can select a model before initiating a task. Once the task is completed, CowPilot presents results evaluated in the metrics from §2.2. To compare different models on the same task, the user can re-do the task

[2] https://docs.litellm.ai/docs/

			Human-Agent Collaboration Metrics				
Mode	LLM Backbone	End-to-End Task Accuracy (↑)	Agent Step Count (↑)	Human Step Count (↓)	Total Step Count (↓)	Human Intervention Count (↓)	Agent-driven Completion Accuracy (↑)
Fully Autonomous	`GPT-4o`	0.48	5.48	0.00	5.48	0.00	0.48
	`Llama 8B`	0.04	7.00	0.00	7.00	0.00	0.04
CoPilot	`GPT-4o`	**0.95**	**6.36**	**1.14**	**7.50**	**0.73**	**0.52**
	`Llama 8B`	0.81	4.77	4.15	8.92	1.15	0.05
Human-only	-	0.89	0.00	9.93	9.93	-	-

Table 2: Evaluation on WebArena subset using COWPILOT.

with different models, allowing for clear, unbiased comparisons under identical conditions.

4 Exemplar Findings via COWPILOT

To demonstrate the usage of COWPILOT, we evaluate on a subset of WebArena (Zhou et al., 2023) benchmark, including 27 tasks categorized into easy, medium, and hard difficulty levels. We categorize the difficulty by the number of examples successfully solved by the top-performing agent (Wang et al., 2024b) on WebArena, and assign them as easy, medium, hard if they have <2, 2–4, and >4 correctly solved examples among the same task template group. We evaluate under two settings: fully autonomous and copilot mode, using `GPT-4o-2024-08-06` and `Llama-3.1-8B-Instruct` as backbones for the LLM agent. For this study, three authors served as human agents, independently performing the tasks for both settings. The results reported represent the average performance across these evaluations.

Additionally, we included a baseline where tasks were executed solely by humans without any agent participation. Table 2 reports results on all metrics introduced in §2.2.

4.1 Copilot Mode Achieves the Best Accuracy

CoPilot mode with `GPT-4o` achieves 95% task accuracy, significantly outperforming the 48% accuracy under autonomous mode (relatively by 97.9%), and even surpassing human task-solving accuracy by 6.7%. This suggests potential productivity increases when solving tasks together with strong LLM-based agents.

On the other hand, copilot mode with the smaller `Llama 8B` model does not bring similar accuracy increases, but slightly degrades the task accuracy by 8%, indicating the limited utility of LLM-based agents backboned by weaker LLMs.

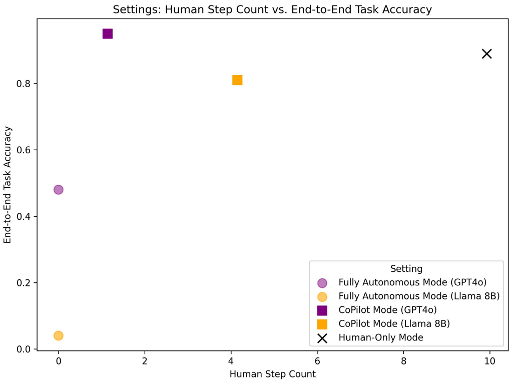


Figure 3: Correlation between Human Step Count and End-to-End Task Accuracy.

4.2 Copilot Mode Requires Minimal Human Intervention

Despite the high task success rates, the GPT-based agent easily achieves the highest accuracy with an average of 1.1 human steps, taking only 15.2% of the entire trajectory. Instead, the LLM agent performs the majority, more precisely 84.8% of task steps. Similarly, when shifting to the weaker LLaMa model, the human-llm collaboration process requires two times more human involvement, resulting in humans and LLM agents spending roughly similar amounts of effort, taking 4.47 and 4.15 respectively. Figure 3 shows the correlation between human step count and end-to-end task accuracy.

Qualitatively, humans often choose to intervene when they observe that the LLM has gotten stuck (e.g., producing the same invalid actions multiple times) or performs an obviously wrong action (e.g., clicking 'Customers' instead of 'Orders' tab when searching for a particular order), especially when the webpage layout is less common or has a confusingly large number of elements.

	Live Website	Dynamic Website	End-to-End Human Annotation	Human-Agent Interaction	Human-Agent Co-task Execution	Agent Evaluation
WebArena	✓	✗	✗	✗	✗	✗
SeeAcT	✗	✗	✓	✗	✗	✗
BrowserGym	✓	✗	✗	✗	✗	✗
WebLinX	✗	✗	✓	✓	✗	✗
WebCanvas	✓	✓	✓	✗	✗	✓
WebOlympus	✓	✓	✗	✓	✗	✓
CowPilot	✓	✓	✓	✓	✓	✓

Table 3: Comparison of CowPilot with existing agent web navigation frameworks.

4.3 Agents Drive Up to Half of the Success

In CoPilot mode, we notice that agent-drive completion accuracy was up to 52% of the time with GPT-4o model. Note that, given the task accuracy was 0.95, the copilot-mode agent successfully initiated half of the successes. These findings highlight that agents can follow the task objective and understand user actions to drive the task up to succeed.

5 Related Works

5.1 Web Agent Plugin

The rise of LLM agents has led to the development of open-source toolkits for web automation, available as APIs, simulated environments, and Chrome extensions. Tools like MultiOn (MultiOn, 2024) and Anthropic (Anthropic, 2024) provide APIs for agent use but require setting up Docker images, posing barriers for non-technical users. BrowserGym (Drouin et al., 2024), AgentLab (Chezelles et al., 2024), WebArena (Zhou et al., 2023) utilize a dedicated Chromium browser instance to perform tasks on specified websites. However, this approach isolates browsing sessions, restricting multi-tab navigation and diverging from standard workflows, which limits practical usability.

Chrome extensions, as adopted by tools like WebCanvas (Pan et al., 2024b), WebOlympus (Zheng et al., 2024b), OpenWebAgent (Iong et al., 2024), and Taxy (TaxyAI, 2024), present a more user-friendly alternative. They are easy to install, lightweight, and integrate seamlessly into standard browsing environments, making them accessible to end-users. While similar to CowPilot, the aforementioned extensions lack features for fostering richer human-agent collaboration. Table 3 further compares how CowPilot with the existing frameworks by illustrating its novel features.

5.2 LLM Agents for Web automation

Web automation has evolved through advancements in LLM-based agents and benchmarks. Early systems relied on HTML structures and accessibility trees (Deng et al., 2024; Gur et al., 2023, 2022; Kim et al., 2023). Visual-based systems such as SeeACT (Zheng et al., 2024a), VisualWebArena (Koh et al., 2024), WebGUM (Furuta et al., 2023) integrate spatial and visual understanding, enhancing agent performance in multimodal tasks. Benchmarks such as MiniWoB (Shi et al., 2017) laid the foundation for evaluating these interactions, while systems like WebShop (Yao et al., 2022), WebArena (Zhou et al., 2023), WebLINX (Lù et al., 2024) expanded to complex multi-step tasks in e-commerce and real-world websites.

Despite these advances, existing systems focus largely on full autonomy, with limited support for human-in-the-loop collaboration. In contrast, CowPilot bridges this gap by enabling dynamic, real-time human-agent interaction. Features like suggest-then-execute, pause, and resume facilitate adaptive task execution, make CowPilot a robust platform for developing and evaluating agents in practical, real-world settings.

6 Limitation and Future Work

Currently CowPilot requires a human to act as an observer to oversee the task execution. This setup is intentional so that we can simulate task execution in *live* setting. We would like to extend our work so that it does not require constant human observation. Rather, we would detect the critical steps that require human observation only. In the future, we would extend CowPilot for a multi-LLM agent setup where we can simulate a user by a second LLM agent. Such setup would help us to approximate human decisions automatically using LLM autorater (Pan et al., 2024a) and incorporate an active learning framework (Bai et al., 2024). We acknowledge a potential ordering bias in the comparative evaluation of autonomous and CoPilot modes. We are currently conducting a large-scale study across a diverse demographic to assess and mitigate the impact of such biases.

Societal Impact

Web agents have significant potential in promoting web accessibility and enhancing user satisfaction. However, their deployment raises important privacy and security concerns. For instance, tracking user actions may expose sensitive information, which could be exploited for malicious purposes (e.g.: data theft). Additionally, in rare cases, agents may inadvertently perform harmful or irreversible actions (e.g.:confirming financial transactions without explicit user consent). We firmly discourage any malicious use of COWPILOT. To balance accessibility with safety, we will not open-source our codebase. Instead, we will release the extension publicly through the Chrome Web Store to ensure controlled access. We will ensure that users can pick their own API key so that they can use their preferred third-party LLM provider or their own local LLM instances so that their information is not shared with us. Future work must focus on addressing these safety risks, including developing robust safeguards to prevent unintended actions and enhancing privacy protection mechanisms.

Acknowledgments

We would like to thank Daniel Fried, Tianqi Chen, and Yonatan Bisk for their valuable feedback on the design of COWPILOT, which contributed significantly to its development. We also acknowledge the creators of Taxy (TaxyAI, 2024) and BrowserGym (Drouin et al., 2024) for open-sourcing their codebases on GitHub, which provided invaluable resources for the development of COWPILOT.

References

Anthropic. 2024. Computer use (beta).

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*.

Hsin-Jung Cheng and Akhil Kumar. 2015. Process mining on noisy logs - can log sanitization help to improve performance? *Decis. Support Syst.*, 79:138–149.

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. 2024. The browsergym ecosystem for web agent research. *Preprint*, arXiv:2412.05467.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*.

Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models. *ArXiv*, abs/2305.11854.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *ArXiv*, abs/2307.12856.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2022. Understanding html with large language models. *ArXiv*, abs/2210.03945.

Iat Long Iong, Xiao Liu, Yuxuan Chen, Hanyu Lai, Shuntian Yao, Pengbo Shen, Hao Yu, Yuxiao Dong, and Jie Tang. 2024. OpenWebAgent: An open toolkit to enable web agents on large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 72–81, Bangkok, Thailand. Association for Computational Linguistics.

Geunwoo Kim, Pierre Baldi, and Stephen Marcus McAleer. 2023. Language models can solve computer tasks. *ArXiv*, abs/2303.17491.

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

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

MultiOn. 2024. Agent api.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024a. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. 2024b. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR.

Wesley G. Siqueira and Laércio Augusto Baldochi. 2018. Leveraging analysis of user behavior from web usage extraction over dom-tree structure. In *International Conference on Web Engineering*.

TaxyAI. 2024. Taxy ai.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024a. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Boyuan Zheng, Boyu Gou, Scott Salisbury, Zheng Du, Huan Sun, and Yu Su. 2024b. WebOlympus: An open platform for web agents on live websites. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 187–197, Miami, Florida, USA. Association for Computational Linguistics.

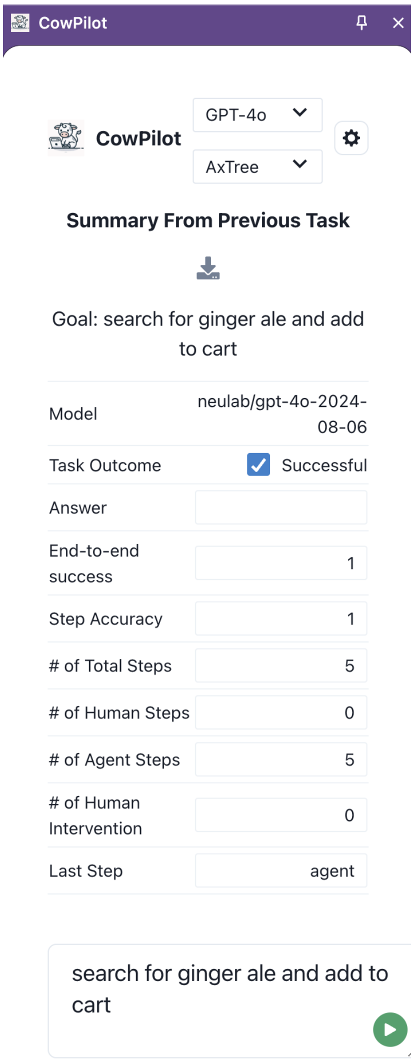
Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Appendix

Figure 4 shows a screenshot of evaluation results by CowPilot. After the task is completed, the summary will be shown containing the metric values covered by subsection 2.2. The scores are auto-calculated based on user intervention count and pause/resume statistics. The user can also modify each entry and save a local copy of the trajectory data by clicking on the download icon.

A.1 Prompt for Action Transformation

Figure 5 shows the prompt we used for `GPT-4o-2024-08-06` to transform the raw user action into filtered actions. The prompt is provided with the event log data structure as well as the agent action space structure. These structures help the LLM to be aware of the structural representation of input raw action. The LLM replies with a list of actions together with their natural language description. The output will be used for the LLM agent action prediction when it is resumed.



GPT-4o

AxTree

Summary From Previous Task

Goal: search for ginger ale and add to cart

Model	neulab/gpt-4o-2024-08-06
Task Outcome	☑ Successful
Answer	
End-to-end success	1
Step Accuracy	1
# of Total Steps	5
# of Human Steps	0
# of Agent Steps	5
# of Human Intervention	0
Last Step	agent

search for ginger ale and add to cart

Figure 4: Screenshot of CowPilot evaluation result page. After each task is completed, the evaluation metric values are shown as summary.

```
You will be shown a list to HTML eventlistener logdata of the following format:
‘export interface EventLogStructure{
action_type: string; // event type (click/scroll/keyup/input/KeyboardEvent/mouseover/contextmenu)
nodeID?: string; // if set, unique ID of the element acted on
elementName?: string;
DOM?: string;
elementouterHTML?: string;
AXTree?: string; // accessibility tree of the HTML page
Screenshot?: string;
coordinateX?: number;
coordinateY?: number;
clickType?: string;
position?: string;
URL?: string; // URL of the current page whre the events are taking place
scrollData?: {
deltaX: number;
deltaY: number;
deltaMode: number;
isLine: boolean;
isPage: boolean;
isPixel: boolean;};
keyData?: {
key: string;
code: string;
isCtrlPressed: boolean;
isShiftPressed: boolean;
isAltPressed: boolean;
isMetaPressed: boolean;
fulltextentry: string;};
urldata?: { // when new tab is opened, the information of the new url and tab id
url_name: string;
tab_id: number;};}‘
Your task is to clean up the raw event data and make a clean list of user actions in the following format: Agent Action Space
Rules:
1. Try to merge consecutive UserLogStructure whenever possible. For example, you can merge multiple keyup actions in the same input field as a setvalue event. For consecutive input in a textbox, always pick the final one. For example, 1) setValue(20, ’Hello’) ... 10) setValue (20, ’Hello world’) can be merged into a single action setValue (20, ’Hello world’)
2. If there are repetitive user actions of the same type in the same place, feel free to discard duplicates. This might specially be true for scroll and mouseover event. For example: two consecutive scrolls in the same direction can be merged. Or, a random, disjoint scroll can be considered as a noise to be ignored.
3. Only reply with availableActions.name(args) format. Do not write any code.
4. Mouseover user log can often be noisy, only add this to the final list if it is meaningful with the rest of the action trajectory in prior and after the mouseover event. For example, a mouseover while tying into a textfield is not useful and can be discarded.
5. Your response must follow json format: [{”thought”: short summary of the action, ”action”: your generated action}].‘
Input: Raw User Actions
```

Figure 5: Prompt for Action Transformation from Raw Event to Agent Action Space.