MolmoWeb: Open Visual Web Agent and Open Data for the Open Web

Tanmay Gupta♥†1* Piper Wolters♥1* Zixian Ma♥1,2* Peter Sushko♥1*

Rock Yuren Pang♥1,2 Diego Llanes♥1 Yue Yang♥1 Taira Anderson[1] Boyuan Zheng[1] Zhongzheng Ren[1,2,3] Harsh Trivedi[1] Taylor Blanton[1] Caleb Ouellette[1] Winson Han[1]

Ali Farhadi[1,2] Ranjay Krishna♥1,2

[1]Allen Institute for AI, [2]University of Washington, [3]UNC-Chapel Hill

* denotes equal technical contribution, † indicates project lead, ♥ marks core contributors.

Abstract

Web agents—autonomous systems that navigate and execute tasks on the web on behalf of users—have the potential to transform how people interact with the digital world. However, the most capable web agents today rely on proprietary models with undisclosed training data and recipes, limiting scientific understanding, reproducibility, and community-driven progress. We believe agents for the open web should be built in the open. To this end, we introduce (1) MolmoWebMix, a large and diverse mixture of browser task demonstrations and web-GUI perception data and (2) MolmoWeb a family of fully open multimodal web agents. Specifically, MolmoWebMix combines over 100K synthetic task trajectories from multiple complementary generation pipelines with 30K+ human demonstrations, atomic web-skill trajectories, and GUI perception data, including referring expression grounding and screenshot question answering. MolmoWeb agents operate as instruction-conditioned visual-language action policies: given a task instruction and a webpage screenshot, they predict the next browser action, requiring no access to HTML, accessibility trees, or specialized APIs. Available in 4B and 8B size, on browser-use benchmarks like WebVoyager, Online-Mind2Web, and DeepShop, MolmoWeb agents achieve state-of-the-art results outperforming similar scale open-weight-only models such as Fara-7B, UI-Tars-1.5-7B, and Holo1-7B. MolmoWeb-8B also surpasses set-of-marks (SoM) agents built on much larger closed frontier models like GPT-4o. We further demonstrate consistent gains through test-time scaling via parallel rollouts with best-of-N selection, achieving 94.7% and 60.5% pass@4 (compared to 78.2% and 35.3% pass@1)on WebVoyager and Online-Mind2Web respectively. We will release model checkpoints, training data, code, and a unified evaluation harness to enable reproducibility and accelerate open research on web agents.

1 Introduction

The World Wide Web is an indispensable part of modern human life, with billions of people relying on it daily for information, communication, commerce, and entertainment [1]. From booking flights and managing finances to navigating government services, comparison shopping, and coordinating logistics, countless everyday tasks require users to interact with complex, multi-step web interfaces that demand sustained attention, domain knowledge, and patience [1, 2]. Web agents—autonomous systems that can navigate and execute such tasks on the web at the behest of users [3]—have the potential to fundamentally reshape how humans interact with this vast digital ecosystem [4, 5]. Effective web agents could broaden digital access for users who face

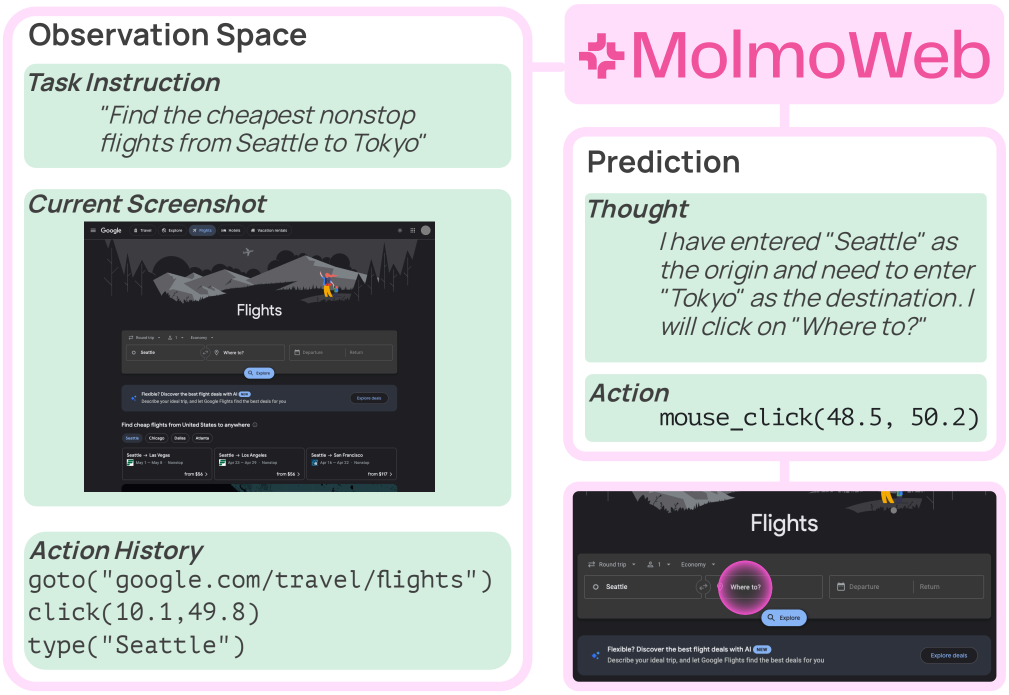

Observation Space
Task Instruction
"Find the cheapest nonstop flights from Seattle to Tokyo"
Current Screenshot
Flights
Action History
goto("google.com/travel/flights")
click(10.1,49.8)
type("Seattle")
MolmoWeb
Prediction
Thought
I have entered "Seattle" as the origin and need to enter "Tokyo" as the destination. I will click on "Where to?"
Action
mouse_click(48.5, 50.2)
Flights
Seattle
Where to?
Explore

Figure 1 Overview of MolmoWeb. At each step, the agent receives the task instruction, current screenshot, and action history as input. It then predicts a natural language thought followed by a browser action.

barriers due to disability or inaccessible design [4, 5], as well as for users with limited digital skills or digital literacy [2].

Recent progress in large language and vision-language models has brought web agents closer to practical viability, supported by research benchmarks demonstrating multi-step web interaction (e.g., clicking, form filling, and navigation) under natural-language instructions [6–9]. In parallel, frontier labs now expose computer- or browser-use capabilities where models perceive GUI state via screenshots and output low-level actions such as click/type/scroll to complete workflows [10–12]. However, the most capable end-to-end systems are typically offered as hosted, proprietary services with limited disclosure of training data and full recipes [10, 12]. This exacerbates long-standing concerns that insufficient reporting and artifacts hinder reproducibility and scientific understanding of what actually drives performance [13, 14]. The resulting opacity is especially concerning for autonomous agents operating on the open web, where trustworthy deployment depends on transparency, accountability, and auditable behavior [15, 16].

We believe agents for the open web should be built in the open. To this end, we provide the research community with a complete, transparent, and reproducible foundation including: **MolmoWebMix**, a mixture of diverse task demonstrations on the browser and web-GUI perception data (Sec. 2); and **MolmoWeb**, a family of fully open and efficient multimodal web agents (Sec. 3). MolmoWeb agents are vision-language models (VLMs) trained to serve as instruction-conditioned multimodal action policies for the web. Given a task instruction, action history, and a screenshot of the webpage, MolmoWeb agent predicts the next browser action. The action is executed in the browser to yield a new observation and this loop repeats until the task is complete. Crucially, MolmoWeb agents operate on any website using only the visual interface that human users see, without requiring specialized APIs or access to the underlying HTML or accessibility tree (AxTree). This vision-centric design is a deliberate choice motivated by several considerations. First, it mirrors how people actually use the web, grounding the agent in the same perceptual modality and making its behavior more interpretable. Second, it avoids the brittleness of Document Object Model (DOM) based representations, which vary significantly across websites, frameworks, and even minor page updates, and which can be incomplete or misleading for dynamically rendered content. Third, it sidesteps the substantial token consumption and compute overhead that structured page representations incur: AxTree inputs can easily consume tens of thousands of tokens per page, whereas a single screenshot provides a compact, information-rich representation of the same content.

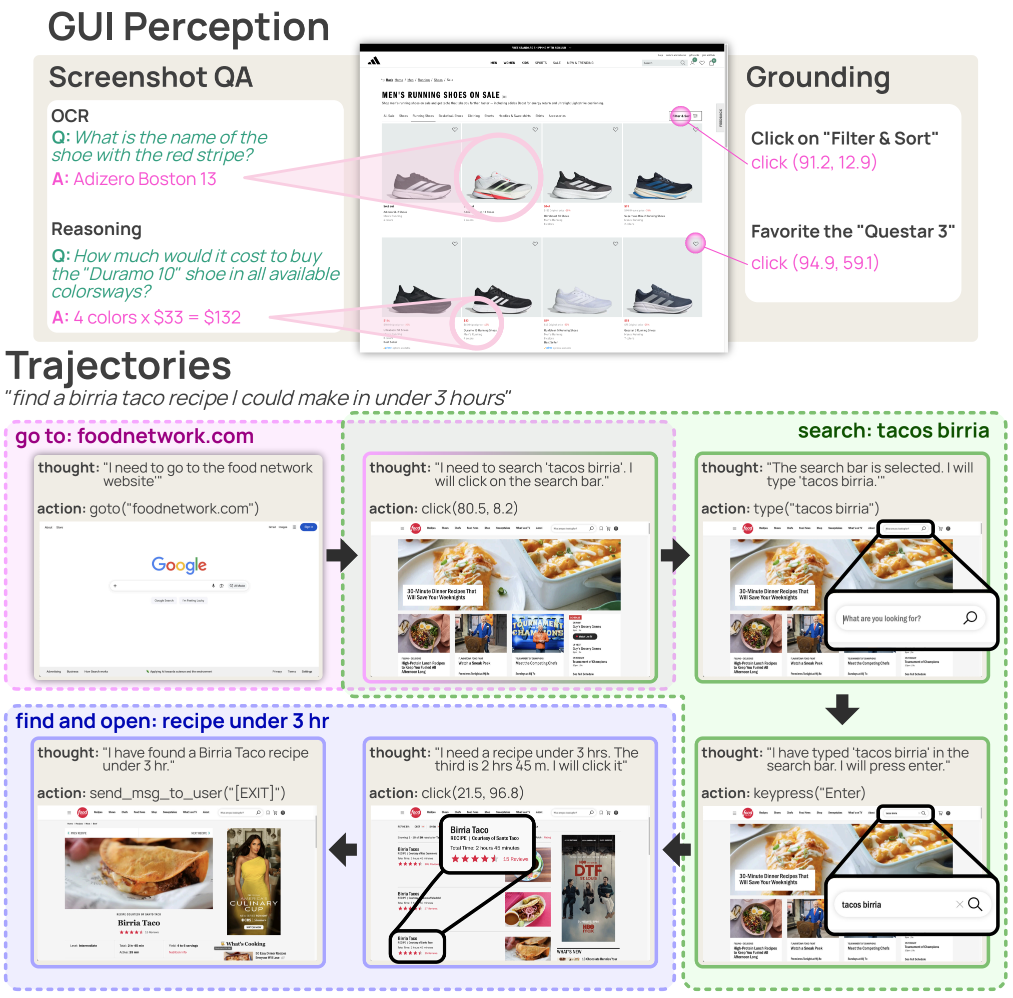
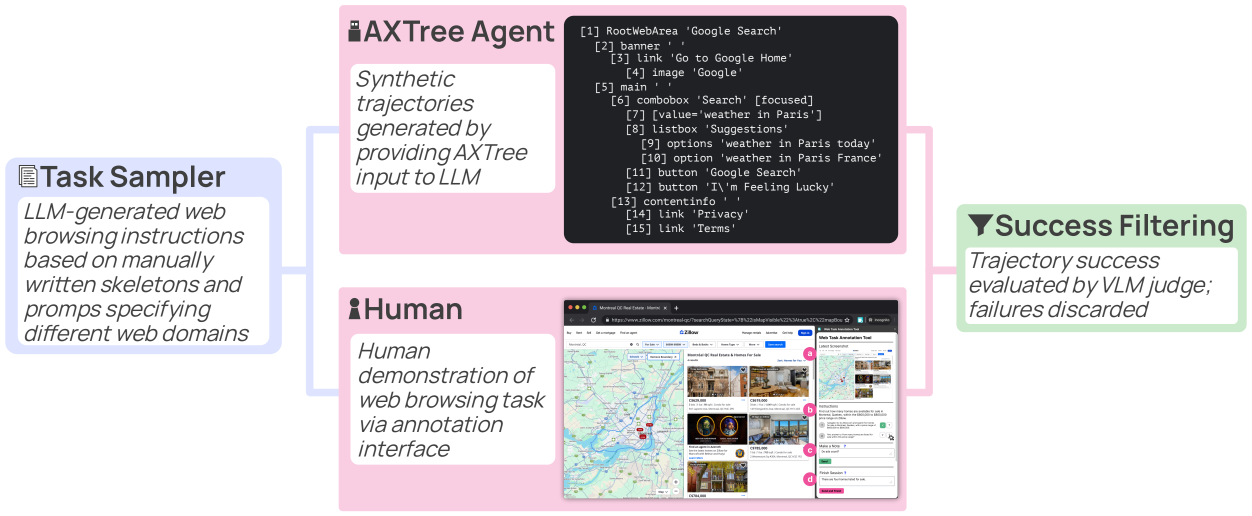

GUI Perception
Screenshot QA
OCR
Q: What is the name of the shoe with the red stripe?
A: Adizero Boston 13
Reasoning
Q: How much would it cost to buy the "Duramo 10" shoe in all available colorsways?
A: 4 colors x $33 = $132
MEN'S RUNNING SHOES ON SALE
Grounding
Click on "Filter & Sort"
click (91.2, 12.9)
Favorite the "Questar 3"
click (94.9, 59.1)
Trajectories
"find a birria taco recipe I could make in under 3 hours"
go to: foodnetwork.com
thought: "I need to go to the food network website"
action: goto("foodnetwork.com")
Google
thought: "I need to search 'tacos birria'. I will click on the search bar."
action: click(80.5, 8.2)
search: tacos birria
thought: "The search bar is selected. I will type 'tacos birria.'"
action: type("tacos birria")
What are you looking for?
thought: "I have typed 'tacos birria' in the search bar. I will press enter."
action: keypress("Enter")
tacos birria
thought: "I need a recipe under 3 hrs. The third is 2 hrs 45 m. I will click it"
action: click(21.5, 96.8)
Birria Taco
RECIPE | Courtesy of Santo Taco
Total Time: 2 hours 45 minutes
15 Reviews
find and open: recipe under 3 hr
thought: "I have found a Birria Taco recipe under 3 hr."
action: send_msg_to_user("[EXIT]")
Birria Taco

Figure 2 Overview of MolmoWebMix dataset. To capture the diverse capabilities required for web agents, MolmoWebMix combines GUI perception data involving screenshot QA **(top left)** and referring expression grounding **(top right)**, with synthetic and human task trajectories demonstrating task completion on the web **(bottom)**. The human trajectories are further segmented into atomic skills (Table 1 shows our skill taxonomy).

The key to training capable MolmoWeb agents is curating a high-quality and diverse training data mixture that simultaneously teaches the model to understand web screenshots as well as task execution behaviors, such as filling forms, multi-hop navigation, and finding relevant information from a webpage. MolmoWebMix contains data from four complementary data sources: (1) synthetic task trajectories generated at scale from multiple complementary pipelines, (2) human demonstrations collected by crowdworkers on real websites, (3) atomic web skill trajectories providing targeted supervision for compositional browser-use skills, and (4) GUI perception data consisting of referring expression grounding and screenshot question-answering examples.

Despite their relatively compact size at 4B or 8B scale, MolmoWeb agents are highly performant. **Without distilling from other visual web agents**, MolmoWeb agents are competitive with or outperform open-weights agents of comparable size, including Fara-7B [17] and Holo1-7B [18]. More notably, they also outperform set-of-marks (SoM) prompting based web agents [19] built on much larger closed frontier models like GPT-4o that take both AxTree and SoM-annotated screenshots as inputs. This result is particularly striking because these closed-model baselines enjoy substantially richer input representations *and* orders-of-magnitude more parameters, yet a well-trained, vision-only MolmoWeb agent achieves stronger task completion rates—suggesting that data quality and targeted training can compensate for raw model scale and additional inputs. Finally, we demonstrate further gains by leveraging increased compute at inference-time via multiple rollouts with best outcome selection using a VLM judge.


Task Sampler
LLM-generated web browsing instructions based on manually written skeletons and promps specifying different web domains
AXTree Agent
Synthetic trajectories generated by providing AXTree input to LLM
Human
Human demonstration of web browsing task via annotation interface
Success Filtering
Trajectory success evaluated by VLM judge; failures discarded

```
[1] RootWebArea 'Google Search'
  [2] banner ' '
    [3] link 'Go to Google Home'
      [4] image 'Google'
  [5] main ' '
    [6] combobox 'Search' [focused]
      [7] [value='weather in Paris']
      [8] listbox 'Suggestions'
        [9] options 'weather in Paris today'
        [10] option 'weather in Paris France'
      [11] button 'Google Search'
      [12] button 'I\'m Feeling Lucky'
    [13] contentinfo ' '
      [14] link 'Privacy'
      [15] link 'Terms'
```

Figure 3 Data generation pipeline. Our pipeline consists of a task sampler, trajectory generation executed by either humans or AxTree agents, and success filtering.

2 MolmoWebMix

Our training data consists of 3 broad categories: task trajectories, atomic skill trajectories, and GUI perception data. The trajectory generation pipeline encompasses web browsing task sampling, trajectory execution, and filtering out failed trajectories, as shown in Figure 3.

2.1 Task trajectories

Trajectories from AxTree agents. A bulk of our synthetic trajectories are generated using an LLM agent that operates on the AxTree representation of the webpage. At each step, the agent receives the serialized AxTree (visualized in Figure 3) as its observation along with the instruction and past actions, and predicts the next action by referencing element browser IDs (`bid`) assigned to each interactive node. We used Gemini-3-Flash-Preview as the agent backbone.

Tasks were sourced from a combination of manually authored and LLM-generated instructions, covering popular websites and benchmark evaluation sites (see Section C in the Appendix for details). Although the agent observes only the AxTree, we capture a screenshot at each step so that all trajectories share a unified observation format. Actions predicted in bid-space are post-processed into pixel-space coordinates (clicks, typing, scrolling). See Table 3 for the complete list of actions.

Synthetic trajectories were filtered for task success using the WebVoyager LLM judge [20], retaining only trajectories deemed successfully completed.

Trajectories from multiagent harness. To generate trajectories with potentially a higher quality of thoughts and trajectories, we design a multi-agent system (Figure 4) where the the system orchestrates web navigation through three specialized roles working in concert: a `Planner` generates the immediate next subgoal based on the high-level task goal as well as task progress determined from verification feedback; an `Operator` executes one low-level browser action (like clicking, scrolling etc.) at each step to accomplish the current subgoal; and a `Verifier` checks whether the current subgoal has been completed by analyzing the most recent 5 screenshots. The trajectory generation process operates iteratively. At each step, the system first runs the `Verifier` to check if the current subgoal is complete. If the subgoal is verified as complete or fails to complete within 5 steps, the `Planner` is called to generate the next subgoal. The system then injects the current subgoal, completed steps, and planner/verifier reasoning into the text prompt and passes this augmented prompt along with the current screenshot to the `Operator`, which returns a concrete browser action. Finally, all historical information is tracked and fed back into subsequent planning and execution cycles. We use Gemini-2.5-Flash for the `Planner`, GPT4-o as the `Verifier`, and Gemini AxTree agent as the Operator. We empirically find

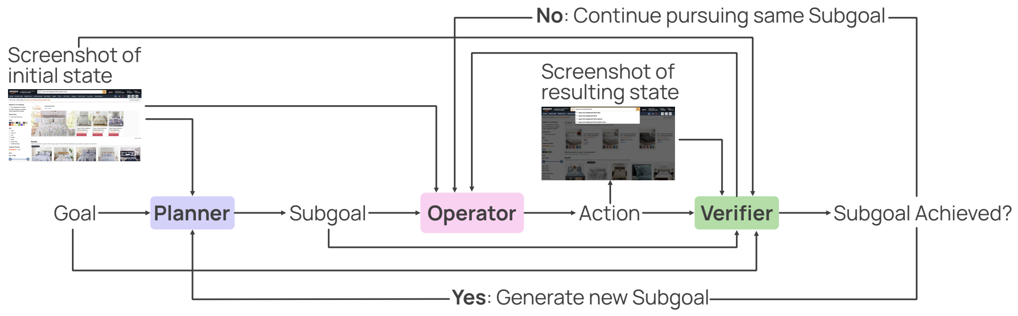

No: Continue pursuing same Subgoal
Screenshot of initial state
Screenshot of resulting state
Goal
Planner
Subgoal
Operator
Action
Verifier
Subgoal Achieved?
Yes: Generate new Subgoal

Figure 4 Multi-agent trajectory generation pipeline. A Planner decomposes the task into sequential subgoals. For each subgoal, an Operator generates an action conditioned on the current state and subgoal. After execution, the resulting state is evaluated by a Verifier using screenshots to determine whether the subgoal is achieved. If not, the system continues attempting the same subgoal; otherwise, the Planner generates the next one. The loop repeats until the overall goal is completed.

that this multi-agent setup achieves higher task completion success rates than using the Gemini AxTree agent alone, scoring 78.5 vs. 74.4 on WebVoyager [20].

Trajectories from humans. In addition to the synthetic trajectories, we collected human demonstrations via a custom Chrome extension that captures browser interaction events (e.g., clicks, scrolls, keystrokes) and corresponding screenshots, and post-processes them into standardized trajectories. Crowdworkers were provided with tasks sourced from a mix of manually authored and LLM-generated instructions (see Section B in the Appendix for details). In general, tasks were chosen to span a wide range of popular websites and to require a variety of browser task-completion skills including search, form-filling, and multi-hop navigation.

To support fine-grained annotation, each task instruction was decomposed into an ordered sequence of subtasks. Workers checked off each subtask upon completion within the annotation tool and submitted a final text response (e.g., an answer to a question or a completion acknowledgment). If a subtask could not be completed due to missing content or an unexpected webpage state, workers recorded their best attempt along with a descriptive note explaining the failure.

Each trajectory is reviewed by a human to verify task completion and ensure that screenshots and actions were correctly captured. Trajectories failing review were revised or re-collected.

Trajectories from node traversal. To complement LLM and human generated task-completion trajectories with verifiable, deterministic supervision, we construct navigation trajectories from precomputed website graphs. We first build a directed graph over 500 popular websites via breadth-first exploration: starting from the homepage, we extract the accessibility tree at each page and prompt an LLM to select a diverse set of navigational links (e.g., category pages, search features, content sections), continuing to a depth of four. Root-to-leaf paths through each graph yield URL sequences that serve as ground-truth navigation plans, as shown in Figure 5.

To convert a chosen sequence of URLs in the node-graph to a trajectory, we use a deterministic process that does not rely on an LLM to replay each path using browser actions like clicking and scrolling. Starting at the root URL, we locates the link to the next target in the accessibility tree, scroll if necessary to bring it into view, and click. Because the agent uses no language model at execution time, trajectories are cheap to produce at scale, and the intended route is known *a priori*, enabling straightforward success verification. When the agent cannot reach a target URL, the path is truncated to the last successfully visited page. At the terminal page of each trajectory, we use an LLM to generate a plausible instruction. Together the instruction-trajectory pair generated via a walk on the node-graph now resembles a goal-directed browser demonstration.

2.2 Atomic skill trajectories

While task trajectories require the agent to compose multiple web skills to complete an end-to-end goal (like searching for a product, applying filters, and adding to cart; each requiring a sequence of low-level browser actions), atomic skill trajectories isolate individual skills. We identify a core taxonomy of web skills that

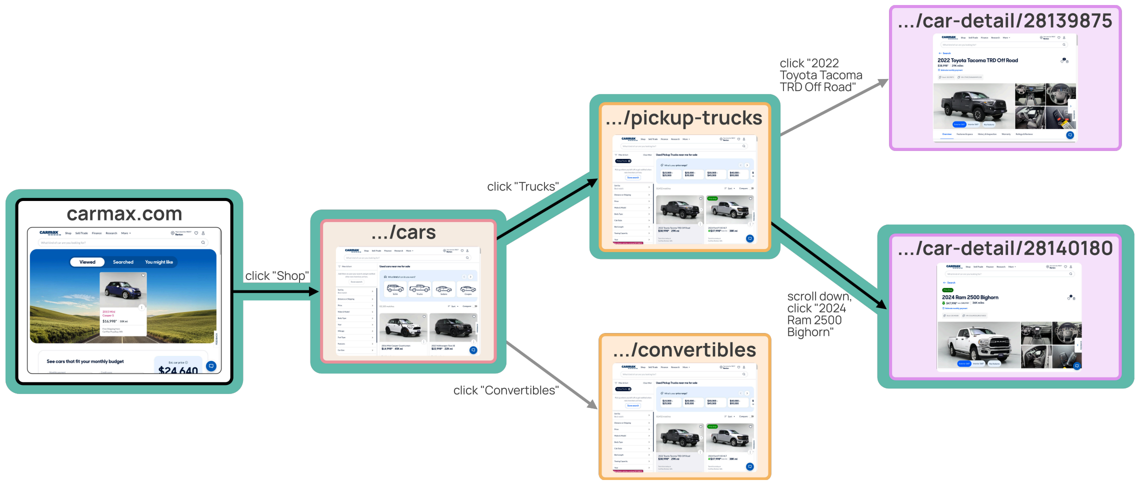


Figure 5 Node traversal trajectory generation pipeline. Starting from a website's homepage, we extract all reachable URLs and use an LLM to select the most informative ones for breadth-first expansion, pruning cycles to avoid loops. A deterministic, LLM-free procedure then generates trajectories by traversing the resulting paths via scrolling and clicking, with success verified through URL matching. Each validated trajectory is paired with an LLM-generated goal conditioned on the full trajectory, producing a task consistent with the produced path.

underlie most browsing tasks, including search, form filling, finding an element or information on page, and filtering (see Table 1). Providing targeted supervision for each skill ensures the model develops reliable competence in these building blocks.

Extracted from human trajectories. Because human task trajectories were annotated with ordered subtask decompositions, each subtask segment naturally constitutes an atomic skill trajectory. Specifically, each segment begins from the browser state in which the previous subtask ended, and carries the subtask instruction as its goal. We automatically extract these segments from the full human trajectories, yielding short, focused demonstrations. Since skill segments within a trajectory begin from the browser state where the last skill segment ends, these demonstrations resemble response to a follow-up query to the user's previous query. Therefore, this data might also enable the agent to learn multi-turn interaction with the user.

Generated by an AxTree agent. To supplement the extracted segments, we additionally prompted the AxTree agent to execute targeted skill instructions for two important skills: `fill_form` and `find_and_open`. For each, instructions were of the form "`go to:[URL]\n fill form:[form details]`" or "`go to:[URL]\n find and open:[target]`." This directly yields skill-level trajectories without requiring segmentation of longer task trajectories.

2.3 GUI perception

Effective web navigation requires the agent to perceive and understand webpage screenshots before taking any action. GUI perception data teaches the model to identify interactive elements on the page (e.g., buttons, links, input fields, and other controls), understand their function and affordances, and extract information from the page in response to natural language queries. This combines skills akin to referring expression grounding, OCR, and reading comprehension, grounded in the visual structure of web interfaces.

Grounding. Grounding data consists of (screenshot, element description) → click point pairs, where the model must predict the pixel coordinate to click for interacting with a specified element. We extract these pairs automatically from AxTree agent trajectories. For each screenshot, we enumerate all clickable elements in the AxTree and for each generate a natural language description using its accessible name and role either with a template or GPT5. The ground-truth click coordinate is sampled randomly within the element's bounding box using a clipped Gaussian prior centered at the element's center, encouraging the model to learn spatially robust clicking rather than always targeting exact centers. In total, we generate more than 7M grounding QA pairs, comprising 3.4M templated queries and 3.8M GPT-5-generated queries. In addition to new grounding

Table 1 Taxonomy of atomic web skills used in human annotation.

Skill	Description
`go_to`	Navigate directly to a specified URL
`search`	Enter a query into a search box and submit
`find`	Locate an element or information on the page
`find_and_open`	Locate an element or subpage and open it
`find_and_click`	Locate an element and click it
`fill_form`	Fill form fields with specified values
`fill_form_and_submit`	Fill form fields and submit the form
`apply_filters`	Set filter or sort controls to specified values
`apply_filters_and_search`	Set filters and trigger a search
`add_to_cart`	Add the current item to a shopping cart
`navigate`	Free-form navigation when stepwise decomposition is unknown

data, we also repurpose the PixmoPoints data from Molmo [22] by formatting single-point QA pairs into click actions and include 1.1M examples in our grounding mixture.

Screenshot QA. Screenshot QA data consists of (screenshot, question) → answer pairs that enhance the model's ability to read and reason about webpage content. For each screenshot in a subset of the AxTree agent trajectories, we provide the corresponding AxTree to an LLM and prompt it to generate question–answer pairs. Questions cover three categories: OCR queries about text and values present on the page (e.g., prices, counts, text content), affordance queries about actions available on the page (e.g., "Where would I find financial news on this page?"), and summarization queries about the overall content or purpose of a page element. To ensure that questions and answers rely solely on visual content, we remove samples that contain references to AxTree-specific information, such as element IDs (e.g., "Click on Bid 32"). In total, the dataset covers 395 websites and 2,237,252 QA pairs, split between 54% OCR, 26% affordance, and 20% summarization questions.

3 MolmoWeb

In this section, we describe the MolmoWeb architecture, its observation and action space, and the training procedure. MolmoWeb is designed to be simple and practical: we build on an existing vision-language model backbone, define a minimal but expressive action space over browser operations, and train end-to-end via supervised fine-tuning on the MolmoWebMix mixture.

3.1 Architecture

MolmoWeb is built on the Molmo2 architecture [22], a multimodal language model that processes interleaved sequences of images and text. As shown in Figure 1, the model takes as input the current webpage screenshot together with the task instruction and the history of past actions, and produces a structured output comprising a natural language thought followed by the next action to execute.

3.2 Observation and action space

Observations. At each step t, the model receives a screenshot of the current browser viewport along with the task instruction. To provide temporal context, the history of actions taken in 10 prior steps is appended as context along with the URL and title of the current page.

Actions. The model predicts a JSON object encoding both a *thought* and an *action* specifying the operation to perform. The thought is a brief natural language rationale for the chosen action. The full action space is

Table 2 Statistics of MolmoWebMix. We also report its constituent sources compared to other datasets.

Dataset / Source	Trajectories	Steps	Domains	Avg steps	Mixture Ratio	Open-source
Mind2Web [6]	2.4K	17K	137	7.3	–	Yes
AutoWebWorld [21]	11.6K	255K	29	21.9	–	Yes
Fara [17]	145K	1.01M	–	6.9	–	No
MolmoWebMix-Traj	278.5K	2.2M	2.6K	13.2	0.80	Yes
Synthetic						
AxTree Single-Agent	70K	793K	1.3K	11.4	0.35	Yes
Axtree Multi-Agent	35K	438K	1.1K	12.5	0.18	Yes
Axtree Atomic Skills	5.5K	68.7K	540	12.4	0.02	Yes
Node-Traversal	16K	151K	833	9.5	0.02	Yes
Human						
Human	36K	623K	1.1K	20.8	0.18	Yes
Human Skills	116K	781K	1.1K	6.8	0.05	Yes
MolmoWebMix-Perception	–	10.5M	–	–	0.20	Yes
PixMoPoints + SyntheticGround	–	8.3M	–	–	0.15	Yes
ScreenshotQA	–	2.2M	–	–	0.05	Yes

described in table 3. Mouse actions are parameterized by spatial coordinates normalized to $[0, 100]$, with 2 decimal points, which are denormalized to viewport pixel coordinates at execution time.

3.3 Training

We train MolmoWeb end-to-end via supervised fine-tuning (SFT) on all training data described in section 2. All data types, including task trajectories, atomic skill trajectories, and GUI perception data, are mixed in a single training stage. The mixing ratios across data sources are treated as hyperparameters; we ablate different mixtures and select the combination that best balances performance across evaluation benchmarks. We report the datasets' final ratios in the mixture in Table 2.

We finetune Molmo2 [22] (with Qwen3 language model and SigLIP2 vision encoder) following its best practice and tune the language model, the vision encoder, and the adapter starting from the single-image checkpoint (pretrained on image captioning and finetuned on single-image QA). We train all models with 64 H100 GPUs with a global batch size of 128 for up to 50K steps (approximately 3.2 epochs on average, with slightly different numbers of epochs across datasets based on their final ratio in the mixture).

4 Experiments

Our experiments consist of comparison to prior art, a study on scaling test-time compute through more inference steps per rollout as well as parallel rollouts with best-of-N selection, data ablation studies to understand the impact of scale and role of human vs synthetic data, comparison of token sampling strategies, and grounding evaluation. We first describe our evaluation setup and then present our experimental results.

4.1 Evaluation setup

We evaluate MolmoWeb on four popular web browsing benchmarks using live websites: WebVoyager [20], Online-Mind2Web [23], DeepShop [24], and WebTailBench [17]. Because some tasks are time-sensitive, we change dates in outdated requests (e.g., find a flight on August 5, 2025) to be meaningful for the task across all benchmarks. For each benchmark and model, we run 3 evaluations with up to 30 steps per task and report

Table 3 Action space of MolmoWeb. Spatial coordinates are normalized to $[0, 100]$ during training and denormalized to viewport pixels for browser execution. Figure 8 in the Appendix shows the distribution of actions in various subsets of MolmoWebMix.

Action	Description
`goto(url)`	Navigate the browser to a URL
`mouse_click(x, y, ...)`	Click at a viewport coordinate
`mouse_drag_and_drop(...)`	Drag from one coordinate to another
`scroll(delta_x, delta_y)`	Scroll the page by an offset
`scroll_at(x, y, dx, dy)`	Scroll at a specific coordinate
`hover_at(x, y)`	Hover at a specific coordinate
`keyboard_type(text)`	Type a string of text
`keyboard_press(key)`	Press a key or key combination
`go_back()`	Navigate to the previous page
`new_tab()`	Open a new browser tab
`tab_focus(index)`	Switch focus to a browser tab
`noop(wait_ms)`	Wait (e.g., for page load or captcha)
`send_msg_to_user(msg)`	Display a message to the user

the average score across runs. If a model does not complete the task by the maximum number of steps, it is considered a failure. As environment errors occasionally occur, we allow up to 10 retries per trajectory; tasks that do not complete within this budget are also marked as failures. We use the same prompt and LLM-as-a-judge setup corresponding to each published benchmark. Specifically, we use GPT-4o with the official prompt from WebVoyager and DeepShop, and o4-mini with respective judge for Online-Mind2Web. Since, it is unclear what judge was used for WebTailBench, we used the WebVoyager judge. For WebVoyager, we use the task set provided by [17] which removed infeasible tasks. All evaluations are run in a Browserbase environment, allowing many live browsers to run in parallel with some captcha-solving capability.

4.2 Comparison to prior work

In Tab. 4, we compare MolmoWeb to web-agents based on proprietary APIs as well as open-weight models.

MolmoWeb establishes a new state-of-the-art among open-weight models. MolmoWeb-8B shows healthy gains over leading open-weight models like Fara across all benchmarks. Even MolmoWeb-4B outperforms all open-weight models on WebVoyager and DeepShop while being competitive on Online-Mind2Web and WebTailBench. These gains largely demonstrate efficacy of MolmoWebMix in teaching a strong foundation vision-language model like Molmo2 to follow instructions for browser-task completion.

Comparison to closed visual agents. Compared to SoM agents that use set-of-marks annotated visual prompts, MolmoWeb-8B easily outperforms GPT-4o, matches the performance of o3 on WebVoyager, and is only 6 pts behind GPT-5 and o3 on DeepShop. MolmoWeb-8B also outperforms OpenAI computer-use-preview on WebVoyager and DeepShop as per the performance reported in [17].

Comparison to non-visual teacher agent. Our synthetic data generation pipeline is powered by Gemini-3-flash axtree agent. It is natural to ask how does the visual student, MolmoWeb, fair against the non-visual axtree-guided teacher? MolmoWeb-8B agent is 5+ pts behind the teacher on WebVoyager and Online-Mind2Web while being 10+ pts behind on DeepShop and WebTailBench. Besides the observation space, other factors contributing to this gap include: (i) differences in model sizes with Gemini presumably being a much larger model compared to our humble 4B and 8B models); (ii) click action requires pixel-space pointing capability from MolmoWeb but the Axtree agent uses unique numeric identifiers (`bid` [27]) of the target

Table 4 Comparison to prior work on browser-use benchmarks. For models marked with a *, the numbers reported are from Fara [17]. Since its unclear what judge was used by [17] for WebTailBench we used the WebVoyager judge to compute numbers marked with †. We highlight the **best** and <u>second-best</u> performing numbers for each benchmark in the sub-8B, open-weight model category.

Model	# Steps	WebVoyager [20]	Online-Mind2Web [23]	DeepShop [24]	WebTailBench [17]
API only					
w/o vision					
Axtree Agent (GPT-5)	30	70.6	41.9	40.7	29.2†
Axtree Agent (Gemini-3-flash)	30	74.4	34.8	45.1	62.1†
Axtree Agent (Gemini-3-flash)	100	85.6	44.8	55.3	63.5†
w/ vision					
SoM Agent (GPT-4o)*	100	65.1	34.6	16.0	30.8
SoM Agent (o3)*	100	79.3	55.4	49.7	52.7
SoM Agent (GPT-5)*	100	90.6	57.7	49.1	60.4
OpenAI computer-use-preview*	100	70.9	42.9	24.7	25.7
Gemini computer-use-preview	100	88.6	53.7	62.0	63.0†
Open weight					
Holo1-7B [18]	30	55.4	–	–	–
UI-TARS-1.5-7B* [25]	100	66.4	31.3	11.6	19.5
GLM-4.1V-9B-Thinking* [26]	100	66.8	33.9	32.0	22.4
Fara-7B* [17]	100	73.5	<u>34.1</u>	26.2	38.4
Ours: Open weight, data, training, and evaluation					
MolmoWeb-4B	100	<u>75.2</u>	31.3	<u>35.6</u>	<u>43.8</u>†
MolmoWeb-8B	100	**78.2**	**35.3**	**42.3**	**49.5**†

element which is then programmatically mapped to a click location; and (iii) answering questions based on text rendered on the webpage requires MolmoWeb to implicitly perform OCR to parse text from screenshot in addition to reading comprehension required by the Axtree agent.

4.3 Test-time scaling

We explore two ways of utilizing more compute at inference time: (i) performing parallel rollouts with a best-of-N selection with an LLM-judge; and (ii) increasing the number of inference steps.

Parallel rollouts. We study how much performance can be gained by running k independent agents on the same task and selecting the best outcome. To get an unbiased, low variance estimate of pass@k, we collect $m > k$ rollouts per task and compute the estimate:

$$\widehat{\text{pass@}k} = 1 - \frac{\binom{m-c}{k}}{\binom{m}{k}}, \tag{1}$$

where c is the number of successful rollouts among the m attempts. We use an LLM judge to evaluate success for each rollout, and we set $m = 5$ to balance accuracy with computational cost. We plot MolmoWeb-4B and MolmoWeb-8B's pass@k performance on the WebVoyager benchmark with max steps of 30 and $k = 1, ..., 4$ in Fig. 6 using the respective judge for best rollout selection. We see that for both 4B and 8B models, parallel rollouts improve success rates substantially with pass@4 leading to 20+ pt gains over single rollout performance. Parallel rollouts help mitigate the compounding error problem where a single incorrect action could throw the agent off-course. These massive gains from parallel rollouts suggest (or, perhaps explain to the unsurprised why) self-distillation from best-of-N rollouts and RL might be effective strategies for further improving single rollout performance.

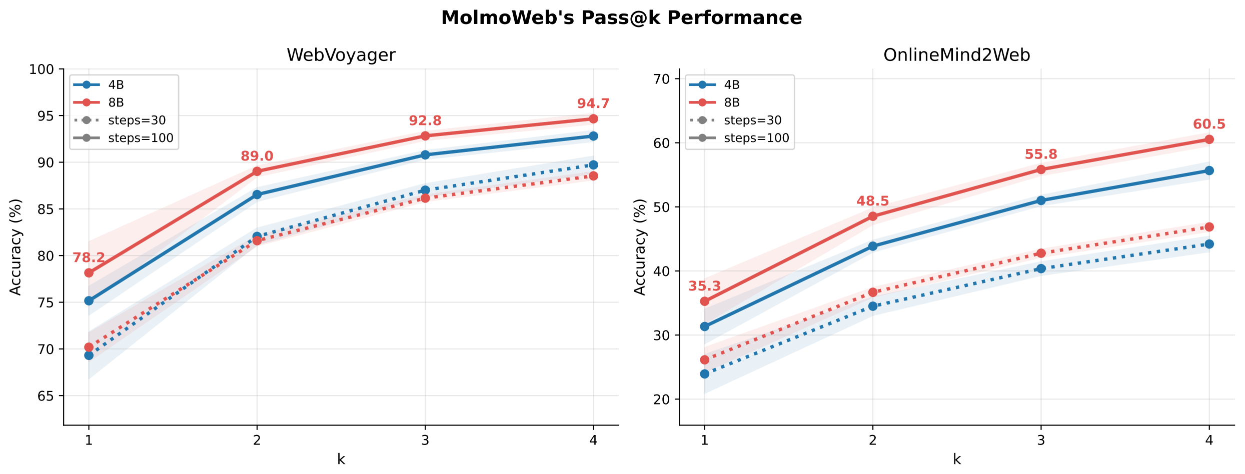


Figure 6 Pass@k performance on WebVoyager and Online-Mind2Web Accuracy improves as k increases, with MolmoWeb-8B generally outperforming MolmoWeb-4B, and 100 steps outperforming 30 steps by a large margin. The MolmoWeb-8B model with max 100 steps reaches 94.7% and 60.5% at Pass@4.

Increasing inference steps. Figure 6 shows consistent gain from increasing the maximum number of inference steps from 30 to 100 across all benchmarks. However, we note that the gains from scaling via parallel runs (each with 30 steps) and picking the best result using an LLM judge far outperform increasing the number of inference steps (e.g., 8B model achieves 86.2% via 3 parallel runs with max 30 steps resulting in a total of 90 steps compared to 78.2% via increasing the steps to 100 in one run).

4.4 Training data ablations

In Table 5, we evaluate the effect of data scale and contributions from synthetic and human data to final performance. The ablations used a slightly earlier version of MolmoWebMix which had fewer human trajectories as well as fewer synthetic trajectories but largely had a similar composition to the final version of MolmoWebMix.

Performance improves with training data scale. Table 5a shows that, unsurprisingly, performance improves with data scale. Interestingly, about 85 to 90% of the performance can be achieved with just 10% of the dataset highlighting the effectiveness of the training mixture.

Limited gains from human data on benchmarks. Table 5b shows that gains from human data are somewhat limited and not consistent across benchmarks. This is likely due to lower volume of human data, differences in task distribution compared to benchmarks, difficulty in learning new actions present only in human data (`scroll_at` and `mouse_drag_and_drop`), difference in style and quality of post-hoc generated thoughts, and human annotation noise (e.g. variable scroll amounts in human annotation as opposed to always scrolling by 100% of viewport height in synthetic data). We hypothesize that synthetic and human data represent distinctly different web task completion policies and the model struggles to learn generalizable behavior across both. For instance, while we observe the model trained only on human data produces both `scroll` and `scroll_at` actions, when trained along with synthetic trajectories the model almost always produces `scroll` which tries to scroll the page instead of the element.

Synthetic trajectories are more effective for learning than human trajectories for the same tasks. To further investigate the difference in learning from trajectories collected by AxTree agents and humans, we collect 2700 web browsing trajectories with each method, using the same task instructions. We then train the model on each of these datasets, mixed with MolmoWebMix-Perception data. Table 6 shows that the model trained on AxTree-sourced trajectories consistently outperforms the one trained on human trajectories, suggesting that the synthetic data provides a more reliable learning signal. We hypothesize two contributing factors for

Table 5 Data ablations. These ablations use an earlier version of MolmoWebMix with both fewer synthetic and human trajectories compared to the final experiments and use 30 inference steps.

(a) Effect of training data scale

Training data	WebV	OM2W
1%	44.5	11.7
10%	63.2	20.4
100%	**68.5**	**21.9**

(b) Humans vs. synthetic data

Training data	#trajs	WebV	OM2W
human only	28K	27.8	13.2
synthetic only	106K	67.8	**22.0**
synthetic + human	134K	**68.5**	21.4

higher signal-to-noise ratio in synthetic trajectories. First, humans tend to exhibit more exploratory behavior, particularly on unfamiliar websites, resulting in longer and noisier trajectories with detours that may hinder imitation learning. Second, the LLM agent operates on the accessibility tree, which encodes rich structural and semantic information about page elements that may not be immediately apparent from visual cues alone. This additional context likely enables the LLM to produce more direct and consistent trajectories, making them more effective as training data despite being synthetically generated.

Table 6 Learning from LLM-generated trajectories is more effective than learning from human demonstrations.

Source	#trajs	DeepShop	WebV	OM2W
human	2.7K	19.8	35.4	9.0
synthetic	2.7K	**24.4**	**53.0**	**16.8**

4.5 Sampling Strategies

Table 7 compares greedy, top-k, and top-p (nucleus) sampling on WebV. We find that sampling strategy significantly affects MolmoWeb's performance, achieving over 5% gains with top-k and top-p sampling compared to greedy decoding. Qualitatively, we find the greedy-decoding could get stuck at specific states (e.g. trying to click at the same location or continuing to scroll even when past attempts at doing so have failed), while the other randomized strategies overcome this behavior. Between top-k and top-p sampling, we see that nucleus sampling with p=0.8 and temperature=0.7 performs the best on Webvoyager and hence is the default sampling strategy used for all experiments. We choose the sampling parameters based on Qwen3's (the base LLM used in Molmo2) recommended parameters on HuggingFace [28].

Table 7 Effect of sampling strategy. Greedy sampling performs significantly worse than random sampling strategies.

Strategy	WebV
greedy (temperature=0.0)	61.4
top-k (temperature=0.7, k=20)	67.4
top-p (temperature=0.7, p=0.8)	**68.5**

4.6 Grounding Experiments

In Table 8, we evaluate MolmoWeb-Ground-8B—an 8B grounding specialist trained only on grounding data as well as the MolmoWeb-4B agent on two grounding benchmarks: ScreenSpot [32] and ScreenSpot v2 [33]. We see that the grounding specialist outperforms open-weight models as well as much larger proprietary models like Claude 3.7 and OpenAI CUA. Our humble MolmoWeb-4B agent is only a few points behind the specialist grounding model while gaining competence on web task completion in addition to grounding.

Table 8 We present MolmoWeb-4B's and MolmoWeb-Ground's (grounding specialist trained on grounding data only) performance on two prominent benchmarks: ScreenSpot and ScreenSpotV2 against open and closed baselines. **Best** and <u>second-best</u> numbers are highlighted.

Model	ScreenSpot	ScreenSpot v2
Claude 3.7 [29]	—	87.6
OpenAI CUA [10]	—	87.9
Gemini-3-Pro [30]	—	**93.7**
UGround-7B [31]	86.7	76.3
Qwen2.5VL-7B [28]	85.5	89.3
Holo1-7B [18]	<u>87.4</u>	89.9
Fara-7B [17]	86.7	89.3
MolmoWeb-4B	87.2	89.5
MolmoWeb-Ground-8B	**88.7**	<u>91.8</u>

5 Related work

We now review related work on GUI understanding, different kinds of web agents, and evaluation benchmarks for web agents.

LLM-driven web agents. Given the generalizable reasoning skills demonstrated by modern LLMs, many works have explored using an LLM as the core reasoning engine for a web agent, often employing prompting frameworks like ReAct [34] that interleave reasoning and action steps. This bifurcates into two broad approaches: (1) using an LLM to directly operate on a language representation of the web page derived from the DOM (such as the accessibility tree representation) to predict browser interactions [6, 7, 27, 35], or (2) exposing web capabilities via API-based tools (e.g. using search engine APIs or other website specific APIs) to a tool-use or coding agent [36, 37]. While this approach works well for specific use cases where the APIs are available and known a priori, it lacks the generality of directly operating the browser as a human would, which unlocks the full range of economically useful tasks on the web. Other work has explored fine-tuning LLMs on web interaction data [38], multi-agent orchestration frameworks that decompose web tasks across specialized agents [39], and tree search methods for planning [40].

Multimodal web agents. In contrast, there are increasing efforts to build agents that process screenshots to produce actions. Earlier work in this direction involved creating modular systems involving planning, grounding, and verification modules [41, 42] or training dedicated visual language models for GUI interaction [43, 44]. Recent work focuses more on a unified approach using a VLM to output actions given screenshot, instruction, and action history. This includes proprietary closed models like Gemini and OpenAI computer-use models [45, 46], and a handful of open-weights-only models like Fara [17], UI-Tars family of models [25, 47, 48], Holo-1 [18], and OpenCUA [49]. Others have explored reinforcement learning with search for web agent training [50]. Unfortunately, without fully open training data and transparent training and evaluation pipelines, it's impossible to advance the science of these multimodal agentic systems. Our work aims to fill this gap with our fully open training dataset MolmoWebMix, training and evaluation pipelines, and weights for our 4B and 8B MolmoWeb models. It is also worth noting that, unlike prior work such as Fara, we avoid distillation from proprietary vision-based web agents. Our data pipeline largely relies on human trajectories and synthetic trajectories generated from AxTree agents that do not see screenshots.

GUI understanding. Separate from applications to web agents, many works have looked at the GUI parsing tasks in isolation, including GUI referring expression grounding [31, 51–54], answering questions about screenshots, and parsing screenshots into structured representations [55–57]. While we use GUI grounding and QA data as sources of auxiliary supervision, our primary goal is to learn to follow instructions to solve tasks on the web.

Evaluation of web agents. Evaluating web agents is challenging. Early evaluation work focused on sandboxed web environments [7, 58–61], desktop environments [62], and multi-turn dialogue navigation datasets [63] where the answer is known or verifiable using oracle knowledge of environment state. Recently, several benchmarks have proposed evaluating on live websites. While some use automatic verifiers [64, 65] or simple text answers that are unlikely to change over time [66], other use a VLM-as-a-judge to verify task completion correctness [20, 23, 24, 67]. A VLM-judge (typically a frontier model such as GPT-4o [68]) takes the instruction, screenshots, and the final answer produced by the agent, along with a prompt specifying the success criteria, and outputs a success or failure decision, along with a rationale for that decision. Online-Mind2Web [23] proposes a more thorough judge that first prompts an LLM to identify key requirements for the successful completion of the task, then identifies key screenshots, and then focuses only on these key screenshots to make the final judgment.

6 Capabilities and Limitations

We now discuss the capabilities and limitations of MolmoWeb along various dimensions:

- **Instruction following:** MolmoWeb was trained with a wide range of instructions across different levels of specificity. The agent performs best for more specific instructions but performance may degrade as the instructions become more ambiguous or require more exploration. MolmoWeb may work best when the name or URL of the target website is mentioned in the instruction. MolmoWeb may also struggle to follow instructions with many constraints or search filters.
- **OCR and reading comprehension:** To answer questions about the current screenshot (e.g., a question pertaining to a paragraph visible in the screenshot), the agent needs to not only localize the relevant part of the webpage, but also implicitly perform OCR and reading comprehension. MolmoWeb is surprisingly capable at this challenging task, but we do see instances of failures due to OCR for smaller texts or for answering complex questions requiring understanding of large passages of text.
- **Latency:** MolmoWeb is trained on trajectories that aim to be efficient and minimize the number of actions required to accomplish the task. However, the model is not optimized for latency (i.e., model feedforward inference time per step). Feedforward efficiency may further be optimized through techniques like quantization or pruning. Another latency bottleneck is the time it takes for the action to be executed in the browser. Browser action execution is faster for local browsers than hosted browsers like Browserbase (especially with proxy configurations), but the latter allows effortless scaling to many concurrent sessions.
- **Thoughts:** An interesting capability by virtue of producing thoughts is that MolmoWeb can sometimes use thoughts as memory for storing information during the trajectory and reference it for producing the final answer (since past thoughts and actions are provided as context at every step). For instance, when asked to list top news headlines, the agent would keep track of all headlines via thoughts as it scrolls through the news headlines list. However, does not demonstrate this behavior robustly. We also see failure modes where thoughts sometimes do not correlate well with actions. For instance, the thought might say the agent needs to type some text and press Enter but the action is directly to press Enter.
- **Action space:** MolmoWeb uses the action space that humans use when interacting with GUIs. Combining some of these actions could yield shorter trajectories. For example, for searching a query with a search box, MolmoWeb currently clicks to select the input box, uses a `keyboard_type` action, and then either uses `keyboard_press('Enter')` or clicks on the search button to execute the search. This sequence could be combined into a single `type_at(text,x,y,press_enter=True)` action. Another potentially useful action not currently included is `web_search(query)` that directly passes query arguments to a search engine via URL parameters (e.g. [17] may benefit from this action). MolmoWeb struggles with predicting more infrequent actions like `scroll_at`, `mouse_drag_and_drop`, or `hover`.
- **Error correction and restarts:** While MolmoWeb shows some error correction behavior (e.g., using the `go_back` action or scrolling to the top of the page if the agent did not find the information on scrolling down), the agent may sometimes get stuck in states where it incorrectly keeps predicting the same action (eg. repeatedly scrolling or clicking at the same location) without being able to recover or course-correct.

7 Conclusion

We introduced MolmoWebMix and MolmoWeb, a fully open data and model suite for multimodal web agents. Operating purely from screenshots, MolmoWeb agents outperform both comparable open-weight models and set-of-marks agents built on much larger proprietary models, while benefiting from test-time scaling via parallel rollouts. We'll release all checkpoints, data, code, and evaluation tools to support reproducible research and accelerate progress on open web agents.

Author Contributions

Tanmay Gupta, Piper Wolters, Zixian Ma and Peter Sushko collectively contributed to dataset construction, model training, and conducted numerous exploratory experiments for this project.

Tanmay Gupta conceptualized and led the project. He worked with all contributors to provide directional guidance, ideate, plan, and debug. Tanmay contributed code to all aspects of MolmoWeb–data collection, training, and evaluation. He also wrote the initial versions of synthetic data generation and evaluation harnesses, created the task generation pipeline for human data annotation, and coordinated with Snorkel AI throughout the human trajectory annotation effort.
Piper Wolters led the development and integration of the project's core infrastructure, spanning human data collection, synthetic data generation, environment setup, model training, evaluation, and the demo.
Zixian Ma led model training, grounding, and multi-agent synthetic trajectories. She generated and experimented with the grounding datasets and multi-agent synthetic trajectories. She also performed extensive experiments on training and inference setups. Together with Tanmay and Piper, she shaped the training and evaluation codebases in the beginning and contributed throughout the project. Together with the team, she trained and evaluated the final models.
Peter Sushko led the synthetic data effort and model evaluation. He optimised the axtree data generation pipeline, produced synthetic tasks, collected the single-agent axtree and node traversal datasets, and developed data filtering strategies. Together with Piper Wolters he wrote much of the model evaluation code, focusing on web environments and large-scale distributed evaluation. He ran modeling ablations on dataset mixtures and ran evaluations for many baseline models.
Rock Yuren Pang led development of the demo and scaled it for broad public deployment, implementing safety guardrails to ensure responsible use and designing the human-AI interaction. He also played a key role in improving the human annotation tool, refining its initial implementation to enhance accuracy and streamline the human-in-the-loop workflow.
Diego Llanes created the Screenshot QA dataset and contributed to the task generation and trajectory filtering pipelines.
Yue Yang explored synthetic data generation using code generation models and was involved in discussions related to data generation, training, and evaluation.
Taira Anderson managed the project, resources, timelines and helped with cross-team communication.
Boyuan Zheng, Zhongzhen Ren, Harsh Trivedi were involved in discussions related to training and evaluation.
Taylor Blanton created the initial versions of the human annotation tool.
Caleb Ouellete worked with Rock in setting up the required infrastructure for the demo.
Winson Han created beautiful figures for this technical report.
Ali Farhadi advised the project.
Ranjay Krishna was the PI for the project.

References

[1] International Telecommunication Union (ITU). Facts and figures 2024: Internet use. Web page, 2024. URL https://www.itu.int/itu-d/reports/statistics/2024/11/10/ff24-internet-use/. Accessed: 2026-03-05.

[2] OECD. Briefing note: Digital skills and digital inclusion. PDF, 2023. URL https://www.oecd.org/content/dam/oecd/en/about/projects/cfe/oecd-city-network-on-jobs-and-skills/Briefing-note-Digital-skills-and-digital-inclusion.pdf. Accessed: 2026-03-05.

[3] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[4] World Health Organization (WHO). Disability and health. Web page, 2023. URL https://www.who.int/news-room/fact-sheets/detail/disability-and-health. Accessed: 2026-03-05.

[5] W3C Web Accessibility Initiative (WAI). Diverse abilities and barriers. Web page, 2024. URL https://www.w3.org/WAI/people-use-web/abilities-barriers/. Accessed: 2026-03-05.

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

[7] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *ArXiv*, abs/2307.13854, 2023. URL https://api.semanticscholar.org/CorpusID:260164780.

[8] Tian Tian Shi, Andrej Karpathy, Linxi Fan, Julio Hernandez, Percy Liang, et al. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017. URL https://proceedings.mlr.press/v70/shi17a/shi17a.pdf.

[9] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tian Tian Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://arxiv.org/pdf/1802.08802.

[10] OpenAI. Computer use | openai api. Documentation. URL https://developers.openai.com/api/docs/guides/tools-computer-use/. Accessed: 2026-03-05.

[11] OpenAI. Openai for developers in 2025. Blog post, 2025. URL https://developers.openai.com/blog/openai-for-developers-2025/. Accessed: 2026-03-05.

[12] Google. Computer use | gemini api | google ai for developers. Documentation. URL https://ai.google.dev/gemini-api/docs/computer-use. Accessed: 2026-03-05.

[13] Odd Erik Gundersen and Sigbjørn Kjensmo. State of the art: Reproducibility in artificial intelligence. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. URL https://ojs.aaai.org/index.php/AAAI/article/view/11503.

[14] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivère, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *Journal of Machine Learning Research*, 22(164), 2021. URL https://jmlr.org/papers/v22/20-303.html.

[15] National Institute of Standards and Technology (NIST). Artificial intelligence risk management framework (ai rmf 1.0). NIST Special Publication, 2023. URL https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf. Accessed: 2026-03-05.

[16] Laura Weidinger, Jonathan Uesato, Maribeth Rauh, Conor Griffin, John Mellor, et al. Taxonomy of risks posed by language models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, 2022. URL https://facctconference.org/static/pdfs_2022/facct22-3533088.pdf.

[17] Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-7b: An efficient agentic model for computer use. *arXiv:2511.19663*, 2025.

[18] Mathieu Andreux, Breno Baldas Skuk, Hamza Benchekroun, Emilien Bir'e, Antoine Bonnet, Riaz Bordie, Nathan Bout, Matthias Brunel, Pierre-Louis Cedoz, Antoine Chassang, Mickaël Chen, Alexandra D. Constantinou, Antoine d'Andign'e, Hubert de la Jonquiere, Aurélien Delfosse, Ludovic Denoyer, Alexis Deprez, Augustin

Derupti, Michael Eickenberg, Marcello Federico, Charles Kantor, Xavier Koegler, Yann Labb'e, Matt Lee, Erwan Le Jumeau de Kergaradec, Amir Mahla, Avshalom Manevich, Adrien Maret, Charles Masson, Rafael Maurin, Arturo Mena, Philippe Modard, Axel Moyal, Axel Nguyen Kerbel, Julien Revelle, Mats L. Richter, María Santos, Laurent Sifre, Maxime Theillard, Marc Thibault, Louis Thiry, Léo Tronchon, Nicolas Usunier, and Tony Wu. Surfer-h meets holo1: Cost-efficient web agent powered by open weights. *ArXiv*, abs/2506.02865, 2025. URL https://api.semanticscholar.org/CorpusID:279119644.

[19] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023.

[20] 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 *ACL*, 2024.

[21] Yifan Wu, Yiran Peng, Yiyu Chen, Jianhao Ruan, Zijie Zhuang, Cheng Yang, Jiayi Zhang, Man Chen, Yenchi Tseng, Zhaoyang Yu, Liang Chen, Yuyao Zhai, Bang Liu, Chenglin Wu, and Yuyu Luo. Autowebworld: Synthesizing infinite verifiable web environments via finite state machines, 2026. URL https://arxiv.org/abs/2602.14296.

[22] Christopher Clark, Jieyu Zhang, Zixian Ma, Jae Sung Park, Mohammadreza Salehi, Rohun Tripathi, Sangho Lee, Zhongzheng Ren, Chris Dongjoo Kim, Yinuo Yang, Vincent Shao, Yue Yang, Weikai Huang, Ziqi Gao, Taira Anderson, Jianrui Zhang, Jitesh Jain, George Stoica, Winson Han, Ali Farhadi, and Ranjay Krishna. Molmo2: Open weights and data for vision-language models with video understanding and grounding, 2026. URL https://arxiv.org/abs/2601.10611.

[23] Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Xiaodong Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *ArXiv*, abs/2504.01382, 2025. URL https://api.semanticscholar.org/CorpusID:277502135.

[24] Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, Maarten de Rijke, Zhaochun Ren, and Xiuyi Chen. Deepshop: A benchmark for deep research shopping agents. *ArXiv*, abs/2506.02839, 2025. URL https://api.semanticscholar.org/CorpusID:279118560.

[25] ByteDance Seed. Ui-tars-1.5. https://seed-tars.com/1.5, 2025.

[26] Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.5 v and glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*, 2025.

[27] Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Alexandre Drouin, Massimo Caccia, L'eo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. *ArXiv*, abs/2412.05467, 2024. URL https://api.semanticscholar.org/CorpusID:274598279.

[28] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

[29] Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

[30] Google. Gemini 3 Pro model card, 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf.

[31] Rui Qian, Xin Yin, Chuanhang Deng, Zhiyuan Peng, Jian Xiong, Wei Zhai, and Dejing Dou. Uground: Towards unified visual grounding with unrolled transformers. *ArXiv*, abs/2510.03853, 2025. URL https://api.semanticscholar.org/CorpusID:281843677.

[32] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:267069082.

[33] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

[34] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[35] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world WebAgent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*, 2023.

[36] Yueqi Song, Frank F. Xu, Shuyan Zhou, and Graham Neubig. Beyond browsing: Api-based web agents. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:273507298.

[37] H. Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Raj Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *ArXiv*, abs/2407.18901, 2024. URL https://api.semanticscholar.org/CorpusID:271516633.

[38] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. AutoWebGLM: Bootstrap and reinforce a large language model-based web navigating agent, 2024. URL https://arxiv.org/abs/2404.03648.

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

[40] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

[41] Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. Agent-e: From autonomous web navigation to foundational design principles in agentic systems. *ArXiv*, abs/2407.13032, 2024. URL https://api.semanticscholar.org/CorpusID:271270241.

[42] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, 2024. URL https://api.semanticscholar.org/CorpusID:266741821.

[43] 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.

[44] Zhuosheng Zhan and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*, 2023.

[45] Google. Computer use | gemini API documentation, 2026. URL https://ai.google.dev/gemini-api/docs/computer-use. Accessed: 2026-03-04.

[46] OpenAI. Computer use | openai api documentation, 2025. URL https://developers.openai.com/api/docs/guides/tools-computer-use/. Accessed: 2026-03-04.

[47] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haolin Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents. *ArXiv*, abs/2501.12326, 2025. URL https://api.semanticscholar.org/CorpusID:275788034.

[48] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

[49] Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Bo Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Hua Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations

for computer-use agents. *ArXiv*, abs/2508.09123, 2025. URL https://api.semanticscholar.org/CorpusID:280635573.

[50] 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.

[51] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*, 2024. URL https://api.semanticscholar.org/CorpusID:269005503.

[52] Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Prasad Sathya Moorthy, Jeff Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui 2: Mastering universal user interface understanding across platforms. *ArXiv*, abs/2410.18967, 2024. URL https://api.semanticscholar.org/CorpusID:273549934.

[53] Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. *Proceedings of the 33rd ACM International Conference on Multimedia*, 2025. URL https://api.semanticscholar.org/CorpusID:277740982.

[54] Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, Si Qin, Lars Lidén, Qingwei Lin, Huan Zhang, Tongxing Zhang, Jianbing Zhang, Dongmei Zhang, and Jianfeng Gao. GUI-Actor: Coordinate-free visual grounding for GUI agents. *ArXiv*, abs/2506.03143, 2025. URL https://api.semanticscholar.org/CorpusID:279118510.

[55] Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Carbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. ScreenAI: A vision-language model for UI and infographics understanding. *ArXiv*, abs/2402.04615, 2024. URL https://api.semanticscholar.org/CorpusID:267523393.

[56] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. *ArXiv*, abs/2408.00203, 2024. URL https://api.semanticscholar.org/CorpusID:271601072.

[57] Wenwen Yu, Zhibo Yang, Jianqiang Wan, Sibo Song, Jun Tang, Wenqing Cheng, Yuliang Liu, and Xiang Bai. Omniparser v2: Structured-points-of-thought for unified visual text parsing and its generality to multimodal large language models. *ArXiv*, abs/2502.16161, 2025. URL https://api.semanticscholar.org/CorpusID:276575751.

[58] Tianlin Shi, Andrej Karpathy, Linxi (Jim) Fan, Josefa Z. Hernández, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, 2017. URL https://api.semanticscholar.org/CorpusID:34953552.

[59] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *ArXiv*, abs/1802.08802, 2018. URL https://api.semanticscholar.org/CorpusID:3530344.

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

[61] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *ArXiv*, abs/2401.13649, 2024. URL https://api.semanticscholar.org/CorpusID:267199749.

[62] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

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

[64] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam Hadj Laradji, Manuel Del Verme, Tom Marty, L'eo Boisvert, Megh Thakkar, Quentin Cappart, David Vázquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks? *ArXiv*, abs/2403.07718, 2024. URL https://api.semanticscholar.org/CorpusID:268363855.

[65] L'eo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier de Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional

planning and reasoning-based common knowledge work tasks. *ArXiv*, abs/2407.05291, 2024. URL https://api.semanticscholar.org/CorpusID:271051028.

[66] Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://api.semanticscholar.org/CorpusID:271328691.

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

[68] OpenAI. GPT-4o system card. *arXiv:2410.21276*, 2024.

[69] Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas. *ArXiv*, abs/2406.20094, 2024. URL https://api.semanticscholar.org/CorpusID:270845490.

A Overview

In this appendix, we provide the following:

- Details for human (section B)and synthetic (section C) trajectory generation pipelines, including details and prompts for how the tasks are sampled.
- Additional dataset statistics like distribution of web categories, websites, and actions covered in MolmoWebMix. (section D)

B Human Trajectory Annotation

B.1 Annotation Tool

Collecting high-accuracy human trajectories from the annotation tool (Figure 7) is challenging due to various quirks of the Chrome extension, particularly around capturing screenshots along with DOM events. These timestamped events and screenshots were post-processed into a clean sequence of screenshots and user actions taken on those screenshots. Annotation workers were given detailed guidelines to ensure all important actions are captured. Particularly, the workers were asked to wait long enough after each action for the webpage to complete loading and for the annotation tool to trigger a screenshot automatically, or to manually take a screenshot in case an automatic screenshot is not triggered. Workers were also discouraged from acting too quickly to avoid missing events and screenshots via interventions like displaying a warning message and through annotation training.

B.2 Task Sampling

To ensure a diversity of tasks for workers to annotate, we generate tasks using multiple strategies as described below:

B.2.1 Manually written task templates.

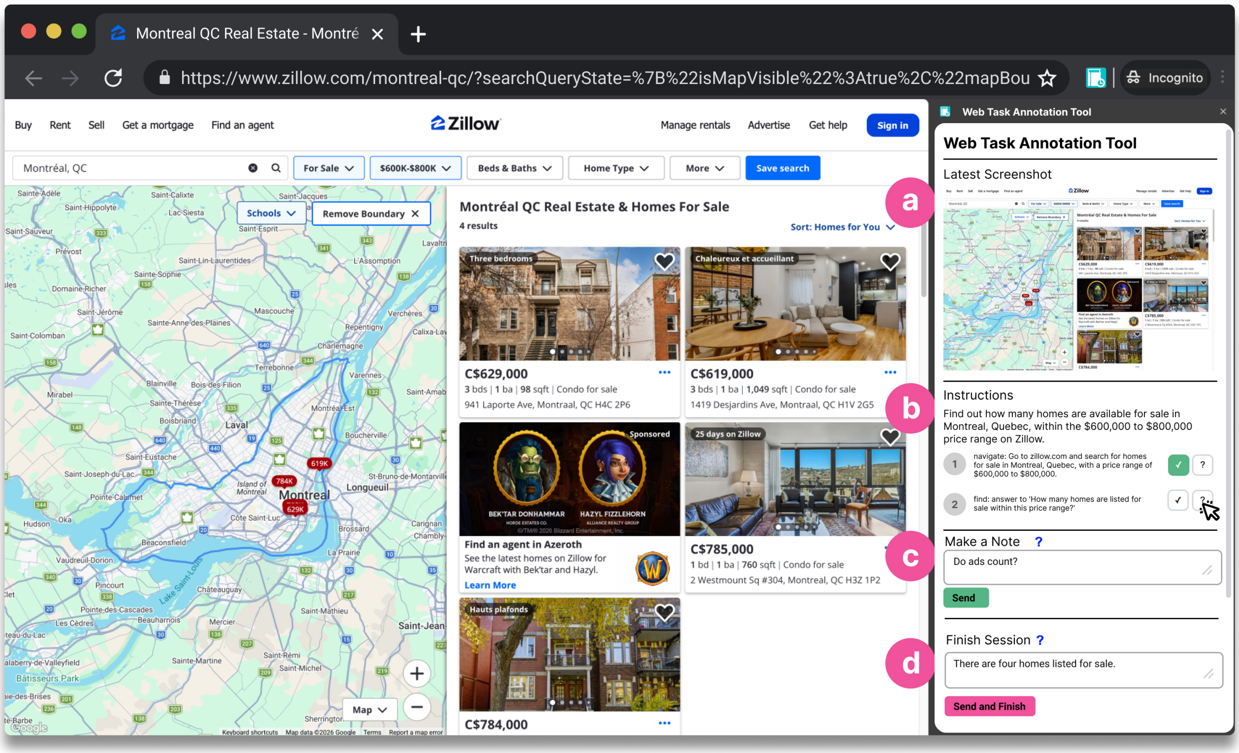
To generate a core set of common use cases, authors wrote task templates. These include tasks related to shopping, news, real estate, travel (including flights, car rentals, hotels, etc.), maps, food & recipes, job & salary search, health & wellness, and cars. Each task template consists of a sequence of atomic skills (using the taxonomy defined in Tab.1 in the main paper) with placeholders that are populated with samples drawn from predefined allowed constants (e.g., destinations for flight search) or sometimes generated on the fly (e.g., filters to apply while shopping for a product). Here's an example for a shopping search and filter task:

```
go to: walgreens.com
search: coffee
apply filters: brand=Lavazza, availability=Pickup
find and open: most relevant product
```

The annotation tool requires the workers to check each of these steps as they are completed. If a step can not be completed, the tool allows the worker to mark the step as incomplete along with a reason why the step could not be completed (e.g., a filter does not exist).

For training purposes, in addition to these step-by-step instructions, we use an LLM to rewrite these instructions with different levels of specificity. The most specific low-level instruction is just a paraphrase of atomic steps into colloquial English, mid-level instruction is less verbose and could leave out details that may be obvious from context, and the highest-level instruction just states the intent without necessarily prescribing a way to achieve the intent. Here is an example of the different levels of specificity:

- **Low-level**: Go to walgreens.com, search for coffee, apply filters for Lavazza brand and available to pickup, and open the most relevant product.
- **Mid-level**: On walgreens, search for Lavazza coffee available for pickup, and open the closest matching product.


Montreal QC Real Estate - Montré
https://www.zillow.com/montreal-qc/?searchQueryState=%7B%22isMapVisible%22%3Atrue%2C%22mapBou
Incognito
Buy
Rent
Sell
Get a mortgage
Find an agent
Zillow
Manage rentals
Advertise
Get help
Sign in
Montréal, QC
For Sale
$600K-$800K
Beds & Baths
Home Type
More
Save search
Schools
Remove Boundary
Montréal QC Real Estate & Homes For Sale
4 results
Sort: Homes for You
Three bedrooms
C$629,000
3 bds | 1 ba | 98 sqft | Condo for sale
941 Laporte Ave, Montraal, QC H4C 2P6
Chaleureux et accueillant
C$619,000
3 bds | 1 ba | 1,049 sqft | Condo for sale
1419 Desjardins Ave, Montraal, QC H1V 2G5
Sponsored
Find an agent in Azeroth
See the latest homes on Zillow for Warcraft with Bek'tar and Hazyl.
Learn More
25 days on Zillow
C$785,000
1 bd | 1 ba | 760 sqft | Condo for sale
2 Westmount Sq #304, Montreal, QC H3Z 1P2
Hauts plafonds
C$784,000
Map
a
b
c
d
Web Task Annotation Tool
Web Task Annotation Tool
Latest Screenshot
Instructions
Find out how many homes are available for sale in Montreal, Quebec, within the $600,000 to $800,000 price range on Zillow.

1 navigate: Go to zillow.com and search for homes for sale in Montreal, Quebec, with a price range of $600,000 to $800,000. ✓ ?

2 find: answer to 'How many homes are listed for sale within this price range?' ✓ ?

Make a Note ?

Do ads count?

Send

Finish Session ?

There are four homes listed for sale.

Send and Finish

Figure 7 Screenshot of the web browsing trajectory collection tool. The left half of the screen shows the webpage, the panel on the right is our chrome extension with annotations showing: **(a)** last captured screenshot; **(b)** instruction given to the annotator with a step-by-step breakdown; **(c)** input box for leaving a note when marking a step could not be completed; **(d)** final answer.

- **High-level**: Find Lavazza coffee for pickup at walgreens.

During training, we randomly sample one of the 4 available instructions (steps-by-step and 3 levels) for each trajectory using a slightly higher sampling probability for the high-level instruction.

B.2.2 LLM sampled tasks with steps.

To increase diversity, we also sample instructions using an LLM. We use a persona sampled from PersonaHub [69] to amplify task diversity along with the following prompt:

```
You are {persona}. Your goal is to generate interesting web tasks that can be performed on
the given website. The task must be specified first as sequence of allowed steps and then as
 a natural language instruction to a web agent.

# Allowed steps
Here are the allowed steps with descriptions -
- go_to: navigate to the website url using url bar. Takes the website url as argument
- search: searching using a simple search box like on google.com or amazon.com. Takes the
search string as argument. Only use this if you know there is a search box on the page.
- find: scroll to locate relevant information in the current page. Takes a description of
what to find as argument
- find_and_open: find and open the relevant page. Takes a description of what to find and
open as an argument. Could be specific like "shopping cart" or general like "most relevant
search result"
- find_and_click: find a relevant element on the page and click on it. Takes a description
of what to find and click as an argument.
- fill_form: fill a form like on airbnb.com or southwest.com. Takes form details as argument
. Useful when you need to take some other action like applying filters before submitting the
 form
- fill_form_and_submit: fill the form and submit it too. Takes form details as argument
- apply_filters: apply filters like product related filters on amazon but don't execute the
search yet. Takes the filter keys and values are arguments
- apply_filters_and_search: apply filters and run the search. Takes filter keys and and
values as arguments
- add_to_cart: when on the product page, add it to cart (for products) or to booking/
reservation (for hotels, flights etc). Takes the amount to add to cart as argument

# What kinds of task to generate?
1. Generate tasks that require navigating to a target page and optionally answering a
question based on the target page. When requiring question answering, the last step should
be find: answer to "{{question}}"
2. Typically the tasks should have 3-10 steps
3. Task should neither be too easy nor too complex. Generally doable in 5-10min
4. Some websites may provide advanced search functionality - make use of these in your tasks
 whenever possible
5. When providing dates for booking or reservation tasks provide dates in 2026. For tasks
that require searching existing data provide dates before 2025 or provide relative dates
like "last 30 days", "next 2 days", "a week from now" etc
6. Do not generate tasks that require uploading files, or opening pdfs
7. Do not generate tasks requiring logins, or user's personal information like name, address
, credit card details etc
8. Generate website relevant tasks, specially search keywords, filters, and form details

WEBSITE: {url}
```

Similar to the manually written steps, we generate instructions at 3 levels of specificity for training from these LLM-sampled step-by-step tasks.

B.2.3 LLM sampled tasks with navigation and QA.

While well structured and easy to follow, the tasks generated by the above prompt can be too rigid and may sometimes be infeasible (e.g., strict search filter constraints result in no matches on the website). To generate more flexible tasks, for a portion of the tasks, we relax the constraint of generating sequence of atomic steps, and ask the LLM to generate a flexible request to navigate to a target webpage, followed by an optional question to be answered once on the target page. Here's an example of such a task:

```
navigate: Find Lavazza coffee for pickup on Walgreens
question: How much does it cost?
```

The prompt used for sampling these navigation and QA tasks was as follows:

```
You are {persona}. Your goal is to generate interesting web tasks that can be performed on
the given website. The task must be specified first as sequence of allowed steps and then as
 a natural language instruction to a web agent.

# Allowed steps
Here are the allowed steps with descriptions -
- navigate: a self contained instruction to navigate to a target webpage. This could involve
 using a search engine, navigation bar, directly navigating to a url, as well as performing
simple or advanced search on websites (like Github, Arxiv), and/or opening a particular
search result, shopping cart, etc. When targeting a specific website, include the name of
the website (if popular) or the URL of the website.
- question: a question about the target page

# What kinds of task to generate?
1. Generate tasks that require navigating to a target page and optionally answering a
question based on the target page.
2. There are only two kinds of tasks to generate: navigation only tasks that only contain a
single navigation step; and 2 step task that contain a navigation step followed by a
question.
3. Task should neither be too easy nor too complex. Generally doable in 5-10min
4. Some websites like Github, Arxiv, Google Scholar etc may provide advanced search
functionality - make use of these in your tasks whenever possible
5. When providing dates for booking or reservation tasks provide dates after March 2026. For
 tasks that require searching existing data provide dates before 2025 or provide relative
dates like "last 30 days", "next 2 days", "a week from now" etc
6. Do not generate tasks that require uploading files, or opening pdfs
```

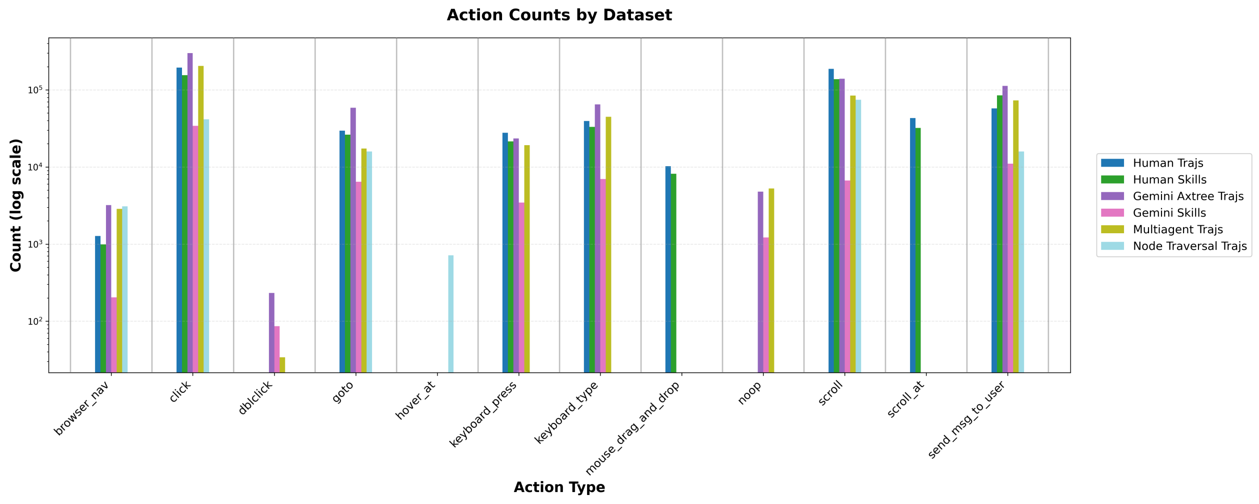


Figure 8 Action distribution breakdown across the dataset.

```
7. Do not generate tasks requiring logins, or user's personal information like name, address
, credit card details etc
8. Generate website relevant tasks, specially search keywords, filters, and form details

WEBSITE: {url}
```

B.2.4 LLM sampled benchmark-like tasks.

All the above generation methods are agnostic to the target evaluation benchmarks and generally aim to cover diverse websites and tasks. However, the above approaches contain biases of authors (either directly in manually written templates or indirectly through prompts) that are likely to be different from the biases of the downstream benchmarks. Therefore, to close the gap in the distribution of tasks in the benchmarks, we generate a set of tasks using tasks from two of the four target benchmarks as in-context examples. Specifically, given a website and all tasks for that website from WebVoyager and Online-Mind2Web, we ask an LLM to generate tasks that are similar to the in-context examples, but not simply paraphrases.

We use GPT-4o for the majority of the LLM sampled tasks, but for some tasks we cycle between GPT-4o, GPT-4.1, GPT-5-mini, and GPT-5 to avoid biases of any single LLM and to increase task diversity.

C Synthetic Trajectories Generation

C.1 Task Sampling

Sampling tasks for synthetic trajectories use the following strategies:

C.1.1 Manually written task templates.

Similar to manually written task templates for human annotation, a portion of tasks for generating trajectories are manually authored with detailed task templates. The main difference from the manual templates for human annotation is that these are natural language templates without a very well-defined atomic skill taxonomy. These manually written templates focused mainly on the websites in the WebVoyager benchmark.

C.1.2 LLM sampled benchmark-like tasks.

In a manner similar to the LLM sampled benchmark-like tasks for human annotations, we generated tasks that match the distribution of WebVoyager. For Online-Mind2Web, for a given task, instead of just generating tasks for the original website, with some probability, we sample a website from the same category (e.g., if

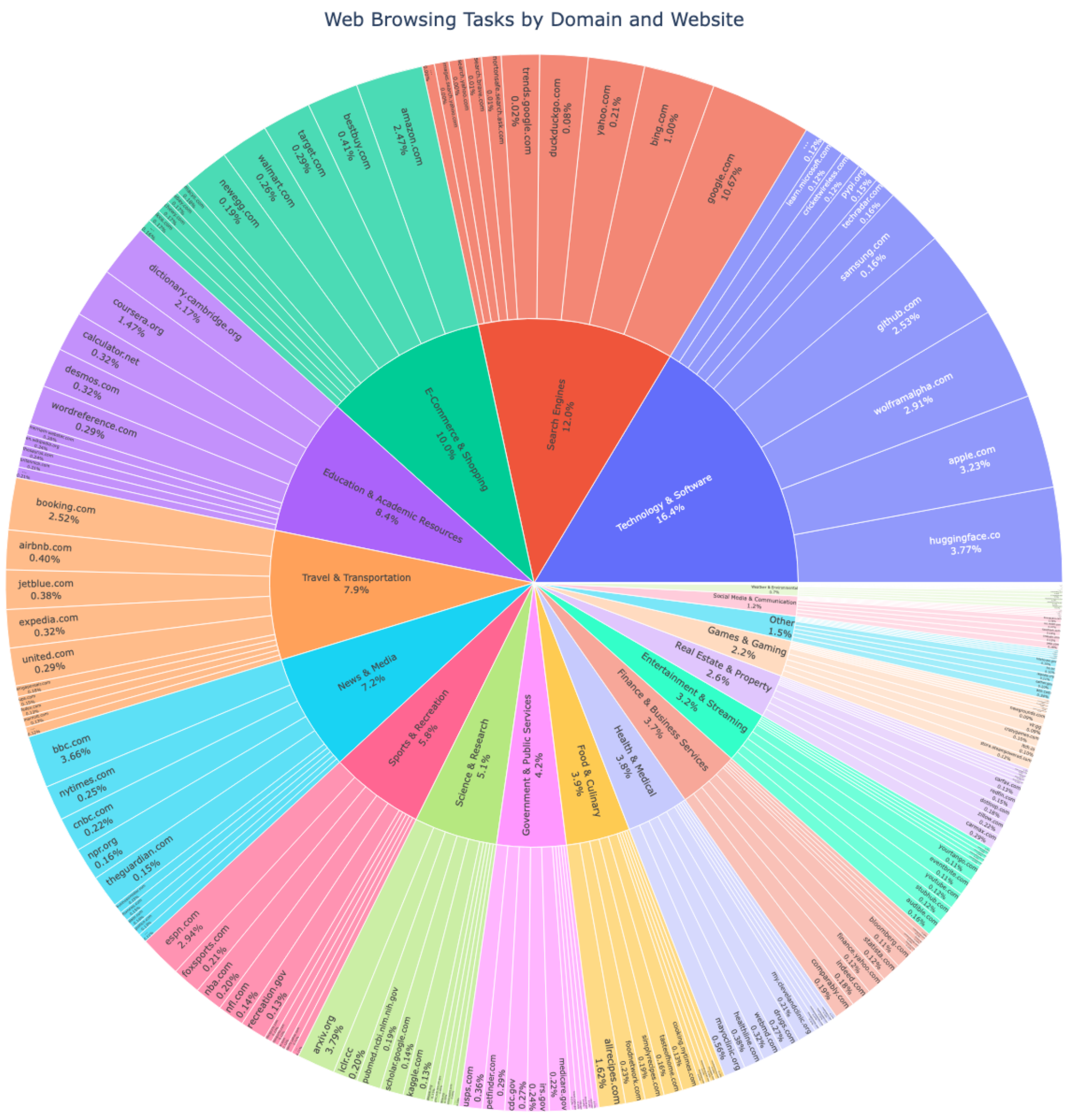


Figure 9 Sunburst chart of domains and websites in the dataset.

the original website is a shopping website, we sample another shopping website from a pre-categorized list of websites), and ask an LLM to generate a similar task for that website.

C.1.3 Taxonomy-based tasks generation

To enable generating diverse tasks and supplementing existing tasks, we define a comprehensive task taxonomy across four axes: (1) intent – what the user wants to do: info-seeking (looking up information), transactional (purchasing/booking), tool-use (using web tools to compute or create), messaging (sending messages or posting content), and navigation (reaching a specific page); (2) domain – the website category, spanning 13 areas: travel, ecommerce, productivity-apps, communication-apps, devtools, social, finance, education, media, reference, local, gov, and health; (3) difficulty – task complexity by constraint count: D0 (0–1 constraints), D1 (2–3), D2 (4–5), D3 (6+). Higher levels require satisfying more simultaneous requirements; (4) ambiguity – goal clarity and solution space: A0 (single correct answer), A1 (optimize under constraints, multiple valid solutions), A2 (open-ended comparison or recommendation), A3 (vague/underspecified goals needing clarification).

We then develop a pipeline to generate synthetic web tasks by systematically iterating over this taxonomy. For each cell in the Cartesian product of these axes, it prompts an LLM (i.e., GPT-4o) to produce a specified number of diverse, realistic web browsing tasks. Each generated task includes a target website, a task type label, a sequence of concrete browser action steps, and three verbosity levels of natural language instructions (high-level, mid-level, low-level). The prompt constrains the LLM with detailed descriptions of each taxonomy axis value and optionally restricts website choices using a list of curated websites. Tasks are tagged with their taxonomy metadata during the generation process.

D Additional Dataset Statistics

Across our synthetic and human datasets, a wide variety of web browsing domains and websites are covered. Figure 9 presents a sunburst chart illustrating the hierarchical breakdown of broad web domains and websites represented in our training data, with just the top 9 websites for each category listed for readability. Our datasets are constructed to closely approximate the action space and input distribution characteristic of human web browsing behavior. In Figure 8 we show the distribution of actions across our various training datasets.