# Explorer: Scaling Exploration-driven Web Trajectory Synthesis for Multimodal Web Agents

**Vardaan Pahuja**[1*†], **Yadong Lu**[2*¶], **Corby Rosset**[2], **Boyu Gou**[1], **Arindam Mitra**[2], **Spencer Whitehead**[2], **Yu Su**[1], **Ahmed Awadallah**[2]
[1]The Ohio State University [2]Microsoft Research, Redmond
pahuja.9@osu.edu, yadonglu@microsoft.com

## Abstract

Recent success in large multimodal models (LMMs) has sparked promising applications of agents capable of autonomously completing complex web tasks. While open-source LMM agents have made significant advances in offline evaluation benchmarks, their performance still falls substantially short of human-level capabilities in more realistic online settings. A key bottleneck is the lack of diverse and large-scale trajectory-level datasets across various domains, which are expensive to collect. In this paper, we address this challenge by developing a scalable recipe to synthesize the largest and most diverse trajectory-level dataset to date, containing over 94K successful multimodal web trajectories, spanning 49K unique URLs, 720K screenshots, and 33M web elements. In particular, we leverage extensive web exploration and refinement to obtain diverse task intents. The average cost is 28 cents per successful trajectory, making it affordable to a wide range of users in the community. Leveraging this dataset, we train **Explorer**, a multimodal web agent, and demonstrate strong performance on both offline and online web agent benchmarks such as Mind2Web-Live, Multimodal-Mind2Web, and MiniWob++. Additionally, our experiments highlight data scaling as a key driver for improving web agent capabilities. We hope this study makes state-of-the-art LMM-based agent research at a larger scale more accessible.[1]

## 1 Introduction

Graphical User Interfaces (GUIs) serve as the primary medium for user interaction across digital environments. Within the GUI environment, LLM-based agents (Su et al., 2024) have shown great potential in automating complex workflows for human users. These agents are designed to operate across diverse interfaces, including the web (Deng et al., 2023; Zhou et al., 2024; Zheng et al., 2024, 2025), desktop (Xie et al., 2024; Wu et al., 2024), and mobile platforms (Rawles et al., 2023; Yan et al., 2023). Navigating modern GUI interfaces, which integrate textual, graphical, and interactive components, typically requires agents to possess visual grounding, long-term planning, and memory management capabilities.

Recent work (Cheng et al., 2024; Gou et al., 2025) has demonstrated the effectiveness of synthetic data for enhancing visual grounding (Gou et al., 2025; Chen et al., 2024a; Kapoor et al., 2024; Chen et al., 2024b) and planning (Xu et al., 2025b; Zhang et al., 2024). Developing end-to-end GUI agents with long-term planning and grounding capabilities requires training on multi-step trajectory data (Xu et al., 2025a,b; Qin et al., 2025). However, existing trajectory datasets are primarily human-annotated (Deng et al., 2023; Li et al., 2024; Lu et al., 2024) or leverage synthetic data just for task proposal curation (Lai et al., 2024; Chen et al., 2024a). And human annotation is expensive to scale for collecting large and diverse training datasets. Therefore, synthetic data has emerged as a promising alternative to human-annotated data (Hartvigsen et al., 2022; Sahu et al., 2022; Ye et al., 2022; Tang et al., 2023; Mukherjee et al., 2023; Mitra et al., 2024). Collecting trajectory-level datasets presents unique challenges: 1) curating a diverse set of task intents at scale, 2) deploying an agent capable of interacting with a real-world environment to complete these tasks through a series of actions, and 3) verifying whether the task is accomplished by the executed action sequence.

Data diversity is essential for equipping generalist web agents with a broad range of skills. Existing work on synthetic web trajectory generation employs self-instruct for task proposal generation (He et al., 2024b). It formulates task proposals from homepages or parametric LLM knowledge,

[*]Equal Contribution. † Work partly done during internship at Microsoft Research. ¶ Project Lead.
[1]Project website: https://osu-nlp-group.github.io/Explorer/



*Findings of the Association for Computational Linguistics: ACL 2025*, pages 6300–6323
July 27 - August 1, 2025 

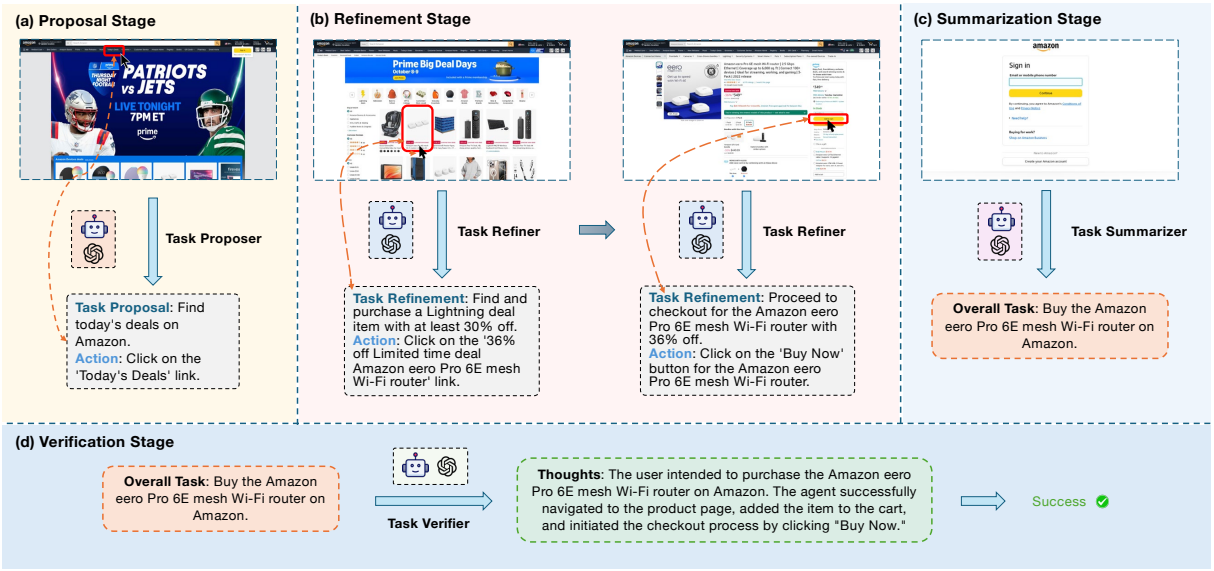



Figure 1: Data Generation Pipeline. The task proposer agent generates an abstract task proposal and the first action based on the website homepage. The task is then iteratively refined in subsequent steps by the refiner agent. Finally, the task summarizer agent constructs an overall task description from the action sequence, followed by task verification to assess correctness.

overlooking the richer content available in deeper web pages, which is essential for achieving broader task diversity. Another line of work leverages web tutorials as a form of supervision for generating web trajectories (Ou et al., 2024; Xu et al., 2025a). While web tutorials effectively cover common daily user tasks, the resulting trajectory data exhibits limited domain diversity in terms of website and domain coverage (Table 1). Additionally, information-seeking tasks remain underrepresented. Due to these limitations, web agents trained on existing synthetic trajectory datasets have not seen much success in more realistic online evaluation settings. To enhance web agents' performance in real-world settings, it is essential to incorporate greater diversity in their training trajectories.

In this work, we develop a *scalable* and *diverse* web trajectory data synthesis recipe for training GUI agent models. Inspired by how humans learn to use the internet, *we leverage exploration as a key mechanism for achieving diversity in task intents*. We introduce **Explorer**, a framework for systematic web exploration to generate diverse, high-quality trajectory datasets. Unlike prior work that relies on static task proposals, Explorer dynamically explores web environments to curate diverse, real-world tasks. This exploration-based approach ensures broader task coverage and better generalization to real-world scenarios. We instantiate this framework using popular URLs from several sources, such as Tranco (Pochat et al., 2019) and `similarweb.com` as seeds. Our dataset comprises 94K diverse web trajectories spanning 49K unique URLs, making it the largest web trajectory dataset to date. Each trajectory is richly annotated with artifacts such as screenshots, raw and set-of-mark (Yang et al., 2023) annotated versions, HTML, and the accessibility tree, enabling comprehensive web agent training. To construct this dataset, we develop a multi-agent pipeline that starts with an abstract task proposal and iteratively refines it into a more specific task through web exploration (Figure 1). Unlike previous approaches, our pipeline generates tasks better grounded in real-world websites, improving task relevance and diversity. To demonstrate the effectiveness of our dataset, we train small language models using just the synthetic data and outperform existing web agent baselines by a significant margin. The main contributions of this work are as follows:

- We develop a scalable and easily customizable multi-agent pipeline for web agent trajectory synthesis. This pipeline leverages exploration as a core mechanism to generate diverse trajectory data, ensuring broad domain coverage and skill diversity in the resulting dataset.

- We leverage this pipeline to generate a diverse and high-quality GUI trajectory dataset consisting of **94K trajectories**, spanning **49K**

| | # Trajectories | # Websites | Modality |
|---|---|---|---|
| RUSS (Xu et al., 2021) | 80 | 22 | HTML |
| Mind2Web (Deng et al., 2023) | 2350 | 137 | HTML + Screenshot |
| WebLINX (Lu et al., 2024) | 2337 | 155 | HTML + Screenshot |
| GUIAct (Chen et al., 2024a) | 5696 | 121 | Screenshot |
| OpenWebVoyager (He et al., 2024b) | 1165 | 48 | A11y tree + Screenshot |
| NNetnav (Murty et al., 2024) | 6K | 4 | A11y tree + Screenshot |
| AgentTrek (Xu et al., 2025a) | 10.4K | 127 | A11y tree + HTML + Screenshot |
| **Explorer** | **94K** | **49K** | **A11y tree + Screenshot (raw + SoM) + HTML** |

Table 1: Comparison to existing web agent benchmarks.

**unique URLs** with 720K screenshots and 33M web elements, making it the largest web trajectory dataset of this scale.

- We demonstrate the effectiveness of our dataset by training small language models, which achieve strong performance on both online and offline benchmarks, significantly surpassing existing web agent baselines, including those with larger parameter counts.

## 2 Related Work

Recent advances in multimodal language models have facilitated the development of web agents — autonomous systems designed to interact with real-world websites to perform everyday tasks (Deng et al., 2023; Hong et al., 2024; Cheng et al., 2024; Zheng et al., 2024, 2025; Xue et al., 2025). Early efforts to acquire trajectory data for training web agents primarily relied on crowd-sourcing (Deng et al., 2023; Lu et al., 2024). However, due to the high cost of human annotation, recent work has adopted synthetic data generation for large-scale collection. AutoWebGLM (Lai et al., 2024) and GUIAct (Chen et al., 2024a) utilize LLMs to generate task proposals, which human experts subsequently annotate. OpenWebVoyager (He et al., 2024b) employs a web agent to execute auto-generated task descriptions. However, since these task descriptions are generated using LLMs without exploring a website, they fail to capture the full diversity of possible tasks on that website. Another line of work, including Synatra (Ou et al., 2024) and AgentTrek (Xu et al., 2025a), leverages web tutorials to guide web trajectory generation. Meanwhile, concurrent effort (Murty et al., 2024) employs an exploration-based trajectory generation in WebArena's sandbox, while our work focuses on more realistic web agent evaluation on live websites. To address diversity limitations in prior trajectory synthesis work, we design a bottom-up web trajectory synthesis pipeline that explores websites dynamically while maintaining a coherent high-level task intent. We refer readers to Appendix F for further discussion.

## 3 Data Recipe

We design an automatic web trajectory synthesis pipeline that explores websites to generate diverse web trajectories. It utilizes Playwright[2] to execute actions and collect metadata from real-world websites, starting from an initial URL.[3] The metadata includes screenshots, HTML, A11y tree, and actions in grounded and natural language forms.

### 3.1 Website Selection

We use a combination of URL sources to generate the synthetic web trajectories. We obtain the top 100 URLs from `similarweb.com` corresponding to the high-traffic portion of the web with transactional tasks like booking flights, restaurant reservations, government services, sports, entertainment, etc. The Tranco (Pochat et al., 2019) URLs include 49K URLs representing the head portion of the web, which is less trafficked but popular nonetheless. We filter out harmful websites containing violent or explicit content to ensure safety compliance. Overall, we generate 94K trajectories across both sources. The complete data generation takes 50 hours, utilizing 60 parallel processes. The viewport resolution is up to $1980 \times 1080$.

### 3.2 Data Generation Pipeline

We aim to develop a generalized pipeline for web exploration to collect diverse web trajectory data. To enhance diversity, we adopt a bottom-up approach, starting with low-level actions and progressively shaping them into high-level task descrip-

[2]https://playwright.dev/

[3]For a 4K subset of trajectories, we instruct GPT-4o to navigate to the target website by formulating a Google search query based on the task description.

| **Information** |
|---|
| View the detailed 7-day weather forecast for Toronto, ON on The Weather Network website. |
| Convert 100 US Dollars to Euros using the XE currency converter. |
| Find directions from Seattle, WA to Bellevue, WA using Bing Maps. |
| **Service** |
| Research the French Bulldog breed on the American Kennel Club website, including its popularity and family life traits. |
| Find the nearest Penske truck rental location in Anaheim, California, and start the reservation process for a truck. |
| Explore and purchase a subscription for the UpToDate Pro Suite on the Wolters Kluwer website. |
| **Entertainment** |
| Find the Basscon presents: Darren Styles EDM event on Eventbrite, save it, and share it on Twitter. |
| View the details of the Photography Competition Winners - Season X and share the article on Twitter. |
| **Shopping** |
| Browse through the fall home decor section on the Target website to explore a variety of fall-themed home decor items. |
| Purchase a three-seat fabric sofa, specifically the UPPLAND Sofa, from IKEA's website. |
| **Travel** |
| Search for flights from Seattle to New York, select travel dates, and explore various flight options. |
| Find the weight of baggage allowance for economy class on qatarairways. |

Table 2: Example task descriptions from Explorer.

| Metric | Value |
|---|---|
| # Total trajectories | 175K |
| # Success trajectories | 94K |
| # Unique URLs | 49K |
| Average steps per trajectory | 7.7 |
| Average elements per image | 46.3 |
| # Tokens | 830M |
| # Elements | 33.3M |
| # Images | 720K |
| Cost per trajectory | $0.15 |
| Cost per successful trajectory | $0.28 |

Table 3: Dataset statistics for Explorer. The number of unique URLs, average steps per trajectory, average elements per image, and number of tokens, elements, and images correspond to the successful trajectories.

tions while maintaining a coherent task intent. In the first step, the proposer agent generates an abstract task, which is refined to a more specific task through a refinement process (Figure 1). Since the agents execute actions alongside the refinement process, the generated tasks respect real-world constraints, such as product availability, available color options, and other specifications, ensuring practical applicability. Our pipeline consists of the following LLM-powered agents[4]:

**Task Proposer.** Given a website homepage, including its screenshot and accessibility tree, the task proposer agent generates diverse initial tasks that could be performed on that website. The task descriptions at this stage are instructed to be high-level and abstract versions of the real-world tasks, which will be refined into more specific tasks in later stages. Along with generating the task proposal, the agent proposes and executes the first action toward completing that task. Furthermore, the agent is instructed to halt upon encountering robot detection such as CAPTCHA verification, login prompts, or payment requests.

**Task Refiner.** The task refiner agent receives the initial task proposal or the refined task description from the previous step, along with the corresponding action history as input. It then predicts the next action consistent with the input task description and the updated, refined task description while incorporating the complete action history. By iteratively refining the task description after each action, the agent ensures that the updated task remains aligned with the action history.

**Task Summarizer.** This module processes the entire action and screenshot history to predict an overall task description that aligns with the trajectory. The task summary is expected to be high level, *i.e.*, it should describe what the task entails while omitting how it is accomplished.

**Task Verifier.** Inspired by Pan et al. (2024a), the task verifier agent receives the task description and action history, serving as a critic to evaluate whether the trajectory successfully completes the specified task. In addition to the screenshots of the trajectory, it also receives a markdown representation of the last page. This ensures the verifier has the full context of the website's final state, even when the viewport cannot capture all the content. To ensure data quality, trajectories that are incoherent or misaligned with the high-level intent are discarded during this stage. Such automatic evalu-

[4]We use GPT-4o as the agent backbone throughout the data generation process.

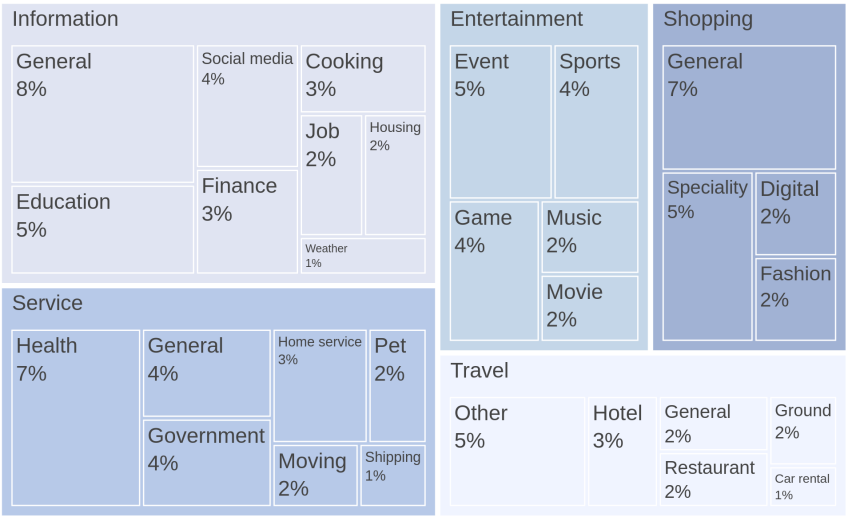



Figure 2: Data composition for Explorer. Its extensive diversity showcases its potential to train end-to-end generalist web agents.

ation of web trajectories has been widely adopted in prior work (Xu et al., 2025a; He et al., 2024a; Koh et al., 2024). Figure 1 illustrates the above pipeline. The prompts for the above agents are given in Appendix D.

### 3.3 Dataset Analysis

Explorer comprises web trajectories spanning diverse domains, including services, entertainment, shopping, travel, and information, ensuring broad task diversity. Sample tasks from Explorer are presented in Table 2. Figure 2 visualizes the domain and subdomain distribution, highlighting the dataset's rich diversity. To the best of our knowledge, Explorer with 94K trajectories is the largest web trajectory dataset of this scale. Table 1 shows a comparison with existing web agent datasets from the literature. The detailed statistics are given in Table 3. Beyond diversity, Explorer is also highly scalable and cost-efficient. Our approach achieves a cost of $0.28 per successful trajectory, making it approximately 2× more cost-effective than AgentTrek (Xu et al., 2025a) (which incurs $0.55 per trajectory) and significantly cheaper than human annotation (Table 4). Unlike human annotation, which requires training crowd workers and continuous quality monitoring, Explorer's automated generation pipeline eliminates these bottlenecks, ensuring scalability with minimal overhead. By combining diversity, scalability, and cost efficiency, Explorer sets a new benchmark for generating large-scale web trajectory datasets, making it an invaluable resource for training generalist GUI agents.

| Model | Cost per trajectory |
|---|---|
| Mind2Web (Deng et al., 2023) | $0.85 |
| AgentTrek (Xu et al., 2025a) | $0.55 |
| **Explorer** | $0.28 |

Table 4: Cost comparison with other approaches.

## 4 Experiments

We use the synthetic trajectories generated by our pipeline to train small multimodal language models (SLMs) for web agent tasks. To ensure computational efficiency, we select 40K trajectories from the full set for training. We further refine this subset by filtering out trajectories that contain more than two scroll actions to mitigate potential model bias toward excessive scrolling behavior. Finally, we use ~30K trajectories obtained after filtering to fine-tune multimodal language models like Phi-3.5V (Abdin et al., 2024) and Qwen2-VL-7B (Wang et al., 2024a). For brevity, we denote the models trained on Phi-3.5V and Qwen2-VL-7B as Explorer-4B and Explorer-7B, respectively. To test the effectiveness of our data for web-based agentic tasks, we evaluate Explorer-4B and Explorer-7B on Mind2Web-Live (Pan et al., 2024b), Multimodal-Mind2Web (Deng et al., 2023; Zheng et al., 2024), and MiniWob++ (Liu et al., 2018).

**Multimodal-Mind2Web.** Multimodal-Mind2Web is an offline web agent benchmark comprising 2K open-ended tasks spanning 137 websites across 31 domains. Each task comprises a sequence of actions with screenshots, action type,

| Model | Avg. Step SR (%) | Completion Rate (%) | Task SR (1) (%) | Full Task SR (%) |
|---|---|---|---|---|
| **API-based Models** | | | | |
| GPT-4o | 58.5 | 52.8 | 44.6 | 25.3 |
| GPT-3.5 | – | 36.5 | – | 15.4 |
| **Open-source Instructed Models** | | | | |
| Mistral-7B-Instr. (Jiang et al., 2023) | 32.8 | 29.5 | 24.1 | 9.6 |
| Qwen2-72B-Instr. (Bai et al., 2023) | – | **40.9** | – | 15.4 |
| Qwen2-VL-7B (Wang et al., 2024a) | 40.2 | 35.4 | **34.9** | 14.5 |
| Phi-3.5V (Abdin et al., 2024) | 28.5 | 23.5 | 20.5 | 2.4 |
| **Supervised Fine-Tuning** | | | | |
| **Explorer-4B** | 44.0 | 39.4 | 31.3 | 18.1 |
| **Explorer-7B** | **45.3** | 40.2 | **34.9** | **19.3** |

Table 5: Results on Mind2Web-Live benchmark. Missing values are denoted by –. The results for GPT-4 and Mistral-7B-Instruct have been reproduced on our Linux servers. The results for GPT-3.5 and Qwen2-72B-Instruct have been taken from Pan et al. (2024b). The full task success rate represents the successful completion of all key nodes for a given task. The average step success rate represents the proportion of completed key nodes, macro-averaged across tasks. The completion rate represents the proportion of completed key nodes, micro-averaged across tasks. Task SR (1) represents task SR with a tolerance of up to one error/key node.

and HTML. We follow the setting in Zheng et al. (2024) and report element accuracy, operation F1, and step success rate (SR) as evaluation metrics.

**Mind2Web-Live.** Mind2Web-Live is a benchmark modified from Mind2Web to test web agents on live websites rather than static trajectories. The benchmark evaluates performance using a key-node-based evaluation approach rather than using a golden action sequence, requiring valid trajectories to reach annotated "key nodes" across 104 test tasks in Mind2Web. Since Mind2Web-Live relies on real-world dynamic websites, it encounters robot detection such as reCAPTCHA, which hinders testing (Xu et al., 2025b). To address this, we select a subset of 83 test set tasks that remain consistently accessible throughout our tests. Following Pan et al. (2024b), we report the average step success rate, completion rate, and full task success rate on the test set.

**MiniWob++.** This benchmark consists of low-level tasks on a single webpage. Typical examples include clicking a sequence of buttons, selecting items from a drop-down list, and filling out a form. We use the subset of 46 tasks used for evaluation in prior work (Zeng et al., 2024; Ou et al., 2024). The final score is obtained by averaging the results of four runs per task. We use the zero-shot evaluation setting, which does not use any environment-specific trajectories for training.

## 5 Results

### 5.1 In-domain Evaluation

As an intrinsic evaluation of the trajectory collection pipeline, we generate 100 test tasks using Explorer, disjoint from the train set. The SLM agents are tasked with executing the given tasks on live websites while an LLM-as-a-judge verifier (§ 3.2) evaluates the correctness of their actions at the trajectory level. Table 7 shows the results. We observe that the fine-tuned agents significantly outperform their pre-trained counterparts. Thus, using in-domain web trajectory data training helps, which is a valuable sanity check.

### 5.2 Mind2Web-Live Results

We evaluate Explorer-4B and Explorer-7B trained on the synthetic trajectory dataset (Table 5). We make the following observations from the results:

**Improvement over base pre-trained models.** We observe that Explorer-7B yields improvements of 5.1% and 4.8% in average step success rate (SR) and key node completion rate, respectively, compared to the pre-trained Qwen2-VL-7B model. Similarly, Explorer-4B obtains gains of 15.5% and 15.9% in average step SR and key node completion rate, respectively, over its pre-trained counterpart. In terms of full task success rate, Phi-3.5V improves significantly from 2.4% to 18.1%, while Qwen2-VL-7B improves from 14.5% to 19.3%. To the best of our knowledge,

| Model | Train Data | Cross-Task | | | Cross-Website | | | Cross-Domain | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Ele. Acc | Op. F1 | Step SR | Ele. Acc | Op. F1 | Step SR | Ele. Acc | Op. F1 | Step SR | |
| **In-Context Learning** | | | | | | | | | | | |
| GPT-3.5 | | 19.4 | 59.2 | 16.8 | 14.9 | 56.5 | 14.1 | 25.2 | 57.9 | 24.1 | 18.3 |
| GPT-4 | | 40.8 | 63.1 | 32.3 | 30.2 | 61.0 | 27.0 | 35.4 | 61.9 | 29.7 | 29.7 |
| SeeAct (Zheng et al., 2024) | | 46.4 | 73.4 | 40.2 | 38 | 67.8 | 32.4 | 42.4 | 69.3 | 36.8 | 36.5 |
| **Supervised Fine-Tuning** | | | | | | | | | | | |
| SeeClick-9.6B (Cheng et al., 2024) | Syn. + M2W | 26.3 | 86.2 | 23.7 | 21.9 | 82.9 | 18.8 | 22.1 | 84.1 | 20.2 | 20.9 |
| EDGE-9.6B (Chen et al., 2024b) | Syn. + M2W | – | – | 30.0 | – | – | 21.1 | – | – | 22.4 | 24.5 |
| MiniCPM-3.1B (Chen et al., 2024a) | Syn. + M2W | 23.8 | 86.8 | 20.8 | 20.3 | 81.7 | 17.3 | 17.9 | 74.5 | 14.6 | 17.6 |
| ScribeAgent-32B (Shen et al., 2024) | Syn. traj. | 38.0 | 52.9 | 35.6 | 34.1 | 52.7 | 32.5 | 39.4 | 54.7 | 37.3 | 35.1 |
| AgentTrek-7B (Xu et al., 2025a) | Syn. + M2W | **60.8** | 88.9 | **55.7** | 57.6 | 88.1 | 51.4 | **56.0** | 87.5 | 52.6 | 53.2 |
| **Explorer-4B** | Syn. traj. | 36.5 | 82.9 | 33.2 | 44.1 | 87.7 | 39.3 | 42.5 | 86.3 | 39.8 | 37.4 |
| **Explorer-4B** | M2W | 48.1 | 88.0 | 44.8 | 49.1 | 87.2 | 45.0 | 46.9 | 87.7 | 44.6 | 44.8 |
| **Explorer-4B** | Syn. + M2W | 53.4 | 88.1 | 50.7 | 55.6 | 89.5 | 51.4 | 49.8 | 88.8 | 47.2 | 49.8 |
| **Explorer-7B** | Syn. traj. | 43.6 | 86.6 | 39.6 | 48.7 | 87.7 | 44.5 | 47.6 | 87.2 | 44.7 | 43.0 |
| **Explorer-7B** | M2W | 51.8 | 88.0 | 48.3 | 56.3 | 89.7 | 52.0 | 50.9 | 88.9 | 48.1 | 49.5 |
| **Explorer-7B** | Syn. + M2W | 56.5 | **90.3** | 53.2 | **60.5** | **90.7** | **56.7** | 55.7 | **90.4** | **53.0** | **54.3** |

Table 6: Multimodal-Mind2Web evaluation results. The baseline numbers have been taken from Zheng et al. (2024); Cheng et al. (2024); Chen et al. (2024b,a); Shen et al. (2024). The last column denotes the average step success rates over the three test splits. Explorer significantly outperforms existing GUI agent baselines.

this represents the state-of-the-art performance on Mind2Web-Live for models of this size trained exclusively on synthetic data.

| Model | Full Task SR (%) |
|---|---|
| GPT-4o | 16.0 |
| Phi-3.5V | 1.0 |
| **Explorer-4B** | 17.0 |
| Qwen2-VL-7B | 6.0 |
| **Explorer-7B** | **18.0** |

Table 7: In-domain evaluation results. The fine-tuned Explorer models achieve significant improvements over their pre-trained counterparts and surpass closed-source LLMs, including GPT-4o.

**Improvement over higher capacity pre-trained models.** Despite having much fewer parameters, we observe that Explorer-4B outperforms strong baselines such as Mistral-7B-Instruct-0.3 and Qwen2-72B-Instruct in full task SR by margins of 8.5% and 2.7%, respectively. The Phi-3.5V model obtains an 18.1% full task success rate, which is better than GPT-3.5 (15.4%), despite using orders of magnitude fewer parameters. The corresponding results for the entire set of 104 tasks, including unreachable websites, are given in Appendix A.1. We provide the ablation studies in Appendix A.3 and the error analysis in Appendix A.4.

### 5.3 Multimodal-Mind2Web Results

Following Deng et al. (2023), we obtain the top-50 elements from a pre-trained DeBERTa (He et al., 2021) candidate generation model, which are then used to construct the accessibility tree and SoM image inputs. The results are shown in Table 6.

Among baselines, we include API-based models for in-context learning – GPT-3.5, GPT-4, and SeeAct (Zheng et al., 2024). SeeAct is a web agent that performs web tasks using a two-step procedure of action generation and grounding using GPT-4V. Additionally, we include baselines that fine-tune small language models using synthetic data, followed by further fine-tuning on the Mind2Web training set. SeeClick (Cheng et al., 2024) introduces a visual grounding model (Qwen-VL) trained on synthetically-generated grounding data. EDGE (Chen et al., 2024b) synthesizes QA data on webpages to improve the grounded GUI understanding capabilities of MLLMs. ScribeAgent-Large (Shen et al., 2024) and MiniCPM-GUI (Chen et al., 2024a) use human-annotated trajectory data to train web agents. AgentTrek (Xu et al., 2025a) is a GUI agent baseline that also utilizes synthetic trajectory data to fine-tune SLMs for Mind2Web, similar to our setting. It synthesizes web trajectory data by guided replay from web tutorials. We observe that Explorer-7B fine-tuned on synthetic data from Explorer plus Mind2Web outperforms all baselines in average step success rate. Notably, it surpasses AgentTrek, which uses the same Qwen2-VL-7B MLLM backbone, highlighting the superior quality of our dataset. The broad domain coverage and task diversity in Explorer contribute to its superior generalization across environments.

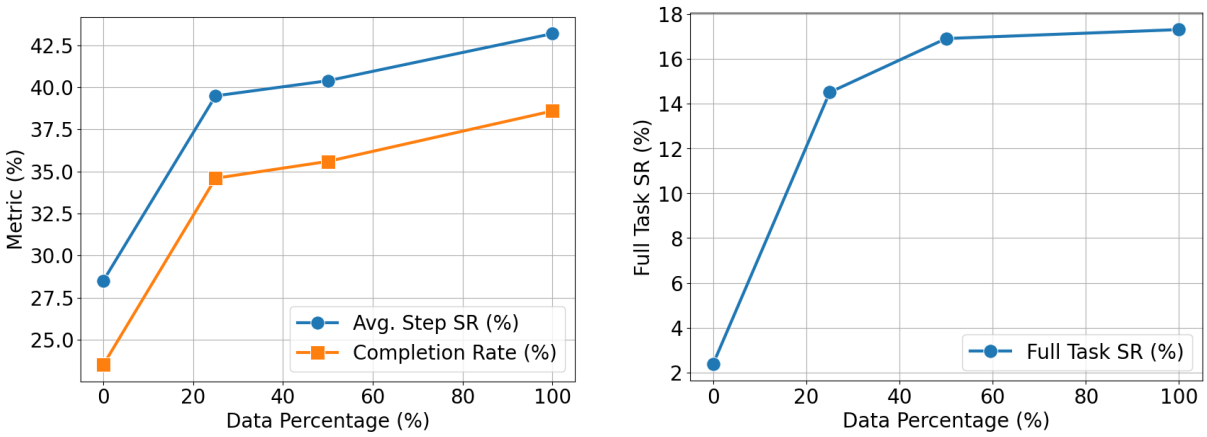



Figure 3: Experiments with data scaling using Explorer-4B on Mind2Web-Live. We experiment with using 100%, 50%, and 25% of the trajectory data. All results are averaged over three runs. All metrics exhibit improvement with an increase in data scale.

| Model | Accuracy (%) |
|---|---|
| *API-based Models* | |
| GPT-3.5 | 39.57 |
| GPT-4 | 53.04 |
| *Open-source Instructed Models* | |
| Phi-3.5V | 35.87 |
| Qwen2-VL-7B | 36.96 |
| Llama3-chat-8B | 31.74 |
| Llama3-chat-70B | 48.70 |
| *Open-source Interactive Data Finetuned Models* | |
| AgentLM-7B (Zeng et al., 2024) | 15.65 |
| CodeActAgent-7B (Wang et al., 2024b) | 9.78 |
| AgentFlan-7B (Chen et al., 2024c) | 20.87 |
| Lemur-chat-70B (Xu et al., 2024) | 21.30 |
| AgentLM-70B (Zeng et al., 2024) | 36.52 |
| Synatra-CodeLlama-7B (Ou et al., 2024) | 38.20 |
| AgentTrek-7B (Xu et al., 2025a) | 45.28 |
| **Explorer-4B** | 46.74 |
| **Explorer-7B** | **53.26** |

Table 8: Results on MiniWob++ benchmark (Liu et al., 2018) in zero-shot evaluation setting. The baseline numbers correspond to Ou et al. (2024). Explorer outperforms much larger models by a significant margin.

| Statistic | Count |
|---|---|
| # Unique task proposals | 53K |
| # Unique final task descriptions | 81K |
| # Final task descriptions | 94K |

Table 9: Task diversity statistics at different stages of the data synthesis pipeline. Abstract task proposals evolve into diverse, fine-grained task descriptions as the agent explores the environment.

### 5.4 MiniWob++ Results

Table 8 shows the results on the MiniWob++ benchmark in the zero-shot evaluation setting. Among baselines, we have API-based models, in-context learning using open-source LMs, and agentic models like AgentLM (Zeng et al., 2024), CodeActAgent (Wang et al., 2024b), Lemur-Chat (Xu et al., 2024) and AgentFlan (Chen et al., 2024c) which include web-based demonstrations in their instruction tuning dataset. Synatra-CodeLlama-7B (Ou et al., 2024) and AgentTrek (Xu et al., 2025a) also synthesize web-agent trajectories automatically. We observe that Explorer outperforms GPT-4 and general-purpose agent baselines. Explorer-4B surpasses Synatra-CodeLlama-7B and AgentTrek-7B despite using a much smaller model with 4.2B params, highlighting our synthetic data's superior quality and strong potential for generalization to new web environments.

### 5.5 Data Scaling Experiments

We conduct experiments with different data scales for Explorer-4B to analyze the impact of training data size. Specifically, we subsample the original trajectory dataset to utilize 50% and 25% of its original size. Figure 3 presents the resulting performance curves. Our results show that, even with just 25% of the training data, the model exhibits rapid performance gains over the base pre-trained model. Increasing the dataset size further leads to gradual improvements across all reported metrics. However, the increase in the overall task success rate is more gradual compared to the stepwise metrics, as it is a more coarse-grained metric.

## 6 Analyses

**Diversity Analysis.** We analyze the diversity of task descriptions across different stages of the data generation pipeline. At the initial proposal stage, the task proposer generates approximately 53K unique high-level goals. As the agent explores the web environment, these proposals evolve into a total of 94K final task descriptions, of which 81K are unique. This progression shows that abstract task proposals can evolve into unique, fine-grained tasks with different constraints, shaped by the agent's specific exploration path. Moreover, multiple trajectories corresponding to the same task description expose the agent to alternative solution paths, enhancing generalization.

| | Pred = Success | Pred = Failure |
|---|---|---|
| **GT = Success** | **0.39** | 0.05 |
| **GT = Failure** | 0.14 | **0.42** |

Table 10: Confusion matrix of the task verifier evaluated on a subset of 100 Explorer-generated trajectories. We observe 81% agreement with human judgment.

**Analysis of Verifier Accuracy.** We did a human evaluation of the task verifier on a random set of 100 trajectories generated using Explorer. We obtain 81% agreement with human judgement for the task verifier, which is comparable to prior work (Pan et al., 2024b; Xu et al., 2025a).

**Failure Modes of Trajectory Generation.** To better understand failure cases in our pipeline, we perform a qualitative analysis of trajectories that were marked as unsuccessful by human annotators (Figure 4). The observed failure modes can be categorized as follows:

- **Grounding error during refinement**: This error happens when the grounded action does not align with the natural language form of the agent's action during the task refinement phase. This misalignment propagates to the summarizer, resulting in an inaccurate final task description.
- **Unresponsive website**: The grounded action is correct and aligns with the task objective, but the website interaction fails due to non-responsive web elements during execution or dynamic content changes.
- **Summarization hallucination**: The summarizer introduces extraneous constraints or goals to the task description that are not present in the underlying trajectory, causing misalignment.
- **Technical issues**: The agent fails to complete the task due to external technical issues such as login requirements, media playback issues, or automated bot detection mechanisms.
- **Task incomplete**: The agent reaches the step limit before completing all required actions, resulting in an incomplete trajectory.

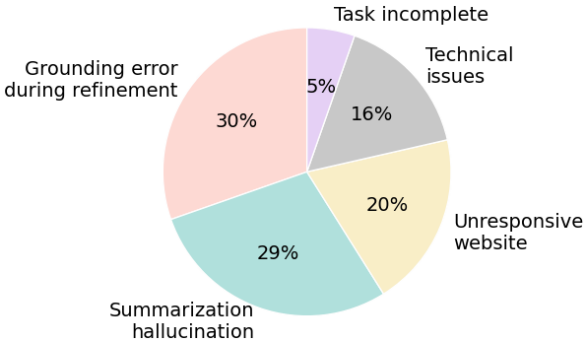



Figure 4: Error type distribution in synthetic trajectory synthesis. Grounding errors in the refinement phase and summarization hallucinations are the most dominant error types.

## 7 Conclusion

In this work, we introduce Explorer, a scalable framework for synthesizing web trajectories on a large scale. By leveraging thorough web exploration, Explorer ensures diversity in both domains and the skills acquired by web agents. Unlike previous approaches, our framework generates contextually grounded trajectories that adapt to real-world constraints, improving both task relevance and generalization. We instantiate this framework using URLs collected from diverse sources. Explorer outperforms existing web agent baselines by a significant margin on both online and offline web agent benchmarks. Furthermore, our results highlight the critical role of data scale in enhancing web agents' performance. Future work will focus on extending this framework to encompass a broader range of GUI environments, such as operating systems with diverse applications. GUI agents require specialized skills for different tasks, including information-seeking, operational, and navigation skills. Efficient exploration of the environment to acquire these skills presents another promising avenue for future research.

## Limitations

Explorer explores the web environment autonomously, which may occasionally result in incoherent tasks. Synthetic data collection using closed-source LLMs can be costly due to associated API expenses. While this work serves as a proof of concept, future research will focus on developing tailor-made open-source LLMs for this task. Additionally, some website content remains inaccessible due to login requirements, leading to insufficient data for those websites.

## Ethical Considerations

The synthetic data collection pipeline proposed in this paper is intended solely for academic research on GUI agents, with strict ethical safeguards to prevent unauthorized website interactions. To ensure ethical compliance and mitigate risks, we prompt our agents to automatically terminate upon encountering CAPTCHA verifications, login prompts, or payment requests, ensuring that no actual transactions or bookings occur. Additionally, we filter out websites containing violent or explicit content and strictly adhere to privacy regulations, ensuring that no personal information is used during action execution. To enforce responsible data collection, we monitor a subset of automatically generated trajectories to ensure compliance with website access policies. Moreover, we distribute the workload across websites to prevent excessive requests and minimize the impact on any single domain.

## Acknowledgements

We would like to thank colleagues from the OSU NLP group and Microsoft Research for their valuable feedback. This research was supported in part by ARL W911NF2220144 and by computational resources provided by the Ohio Supercomputer Center (Center, 1987).

## References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Ohio Supercomputer Center. 1987. Ohio supercomputer center.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, et al. 2024a. Guicourse: From general vision language models to versatile gui agents. *arXiv preprint arXiv:2406.11317*.

Xuetian Chen, Hangcheng Li, Jiaqing Liang, Sihang Jiang, and Deqing Yang. 2024b. Edge: Enhanced grounded gui understanding with enriched multi-granularity synthetic data. *arXiv preprint arXiv:2410.19461*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024c. Agent-flan: Designing data and methods of effective agent tuning for large language models. In *Findings of ACL*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of ACL*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Proceedings of NeurIPS*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *Proceedings of ICLR*.

Izzeddin Gur, Hiroki Furuta, Austin V. Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *Proceedings of ICLR*.

Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. 2022. Toxigen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection. In *Proceedings of ACL*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024a. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of ACL*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2024b. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. *arXiv preprint arXiv:2410.19609*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-enhanced bert with disentangled attention. In *Proceedings of ICLR*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for GUI agents. In *Proceedings of CVPR*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *Proceedings of ECCV*.

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

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: A large language model-based web navigating agent. In *Proceedings of KDD*.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on UI control agents. In *Proceedings of NeurIPS Datasets and Benchmarks Track*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *Proceedings of ICLR*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. In *Proceedings of NeurIPS*.

Xing Han Lu, Zdenek Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. In *Proceedings of ICML*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for general AI assistants. In *Proceedings of ICLR*.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, et al. 2024. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D Manning. 2024. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator. *arXiv preprint arXiv:2410.02907*.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. 2024. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In *Proceedings of NeurIPS*.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024a. Autonomous evaluation and refinement of digital agents. In *Proceedings of Conference on Language Modeling*.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. 2024b. Webcanvas: Benchmarking web agents in online environments. In *Agentic Markets Workshop at ICML 2024*.

Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of Network and Distributed System Security Symposium*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. In *Proceedings of NeurIPS Datasets and Benchmarks Track*.

Gaurav Sahu, Pau Rodríguez, Issam H. Laradji, Parmida Atighehchian, David Vázquez, and Dzmitry Bahdanau. 2022. Data augmentation for intent classification with off-the-shelf large language models. In *Proceedings of the 4th Workshop on NLP for Conversational AI, ConvAI@ACL*.

Junhong Shen, Atishay Jain, Zedian Xiao, Ishan Amlekar, Mouad Hadji, Aaron Podolny, and Ameet Talwalkar. 2024. Scribeagent: Towards specialized web agents using production-scale workflow data. *arXiv preprint arXiv:2411.15004*.

Yu Su, Diyi Yang, Shunyu Yao, and Tao Yu. 2024. Language agents: Foundations, prospects, and risks. In *Proceedings of EMNLP: Tutorial Abstracts*.

Ruixiang Tang, Xiaotian Han, Xiaoqian Jiang, and Xia Hu. 2023. Does synthetic data generation of llms help clinical text mining? *arXiv preprint arXiv:2303.04360*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better LLM agents. In *Proceedings of ICML*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. OS-copilot: Towards generalist computer agents with self-improvement. In *Proceedings of ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

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. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Proceedings of NeurIPS Datasets and Benchmarks Track*.

Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna, Larry Heck, James A. Landay, and Monica Lam. 2021. Grounding open-domain instructions to automate web support tasks. In *Proceedings of NAACL*.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. 2025a. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *Proceedings of ICLR*.

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, Zhoujun Cheng, Siheng Zhao, Lingpeng Kong, Bailin Wang, Caiming Xiong, and Tao Yu. 2024. Lemur: Harmonizing natural language and code for language agents. In *Proceedings of ICLR*.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025b. Aguvis: Unified pure vision agents for autonomous gui interaction. In *Proceedings of ICML*.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*.

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Proceedings of NeurIPS*.

Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. 2022. Zerogen: Efficient zero-shot learning via dataset generation. In *Proceedings of EMNLP*.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. 2024. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of EMNLP*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of ACL*.

Jiwen Zhang, Jihao Wu, Teng Yihua, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024. Android in the zoo: Chain-of-action-thought for GUI agents. In *Findings of EMNLP*.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. In *Proceedings of ICML*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *Proceedings of ICLR*.

## Appendices

In this supplementary material, we provide further details as follows:

- Appendix A: Mind2Web Training and Evaluation Details
- Appendix B: Cost Analysis
- Appendix C: Task Complexity Analysis
- Appendix D: System Prompts
- Appendix E: Trajectory Examples
- Appendix F: More Related Work

## A Mind2Web Training and Evaluation Details

Table A.2 shows the hyperparameters and training time for experiments on Mind2Web-Live and Multimodal-Mind2Web. All experiments use Nvidia H100 GPUs.

### A.1 Mind2Web-Live

We exclude the following websites - `https://www.kbb.com`, `https://www.sixflags.com`, `https://www.viator.com`, `https://www.menards.com`, `https://www.amctheatres.com`, `https://www.cargurus.com`, `https://www.gamestop.com`, `https://www.cabelas.com`, `https://www.rei.com` due to denial of access faced during our tests. Table 5 shows the results on Mind2Web-Live for 83 out of 104 tasks across the remaining 37 websites. The results on the whole Mind2Web-Live evaluation set are given in Table A.1. The results in Table 5 are reported as the maximum over three runs, accounting for intermittent website access issues that may affect evaluation consistency. For Mind2Web-Live, the dataloader first samples training instances at the trajectory level and then randomly samples a step from the trajectory to construct the final training instance. Thus, the number of epochs is calculated at the trajectory level. We use a viewport resolution of $1280 \times 720$ during inference. The Mind2Web-Live dataset is released under the MIT license, which permits its use in academic research.

### A.2 Multimodal-Mind2Web

Following Deng et al. (2023), we obtain the top-50 elements from a pre-trained DeBERTa (He et al., 2021) candidate generation model, which are then used to construct the accessibility tree and SoM image inputs. Following Ou et al. (2024), we always include the ground truth element in the input. We use a viewport resolution of $1280 \times 720$ which includes the GT element during inference. We follow the setting in Zheng et al. (2024) and report element accuracy, operation F1, and step SR as evaluation metrics. All experiments on Multimodal-Mind2Web use a single training and evaluation run. The dataloader uniformly samples training instances from the set of action steps across all trajectories. The Multimodal-Mind2Web dataset is released under the Responsible AI license, which permits its use in academic research.

### A.3 Ablation Studies

We conduct ablation studies to assess the impact of various design choices on overall performance (Table A.3). To evaluate the importance of visual modality, we experiment with using just the textual modality for the Phi-3.5V model, replacing it with the text-only Phi-3-mini (Abdin et al., 2024). In addition to Qwen2-VL-7B and Phi-3.5V, we also evaluate LLaVA-Mistral-7B (Liu et al., 2023), a strong MLLM baseline. Our results show that omitting the visual modality leads to a sharp 4.8% drop in performance for Phi-3.5V, underscoring its importance for effective GUI grounding. Furthermore, LLaVA-Mistral-7B significantly underperforms compared to both Qwen2-VL-7B and Phi-3.5V, highlighting the necessity of a stronger MLLM backbone for better GUI agent performance.

### A.4 Case Studies for Mind2Web-Live

We randomly sample 20 error cases for Explorer on Mind2Web-Live to gain insights for future improvement. These errors fall into the following categories:

- **Task deviation**: The agent executes actions unrelated to the given task, thus failing to complete it.
- **Missing key steps**: The agent retrieves results that partially satisfy the required constraints, *e.g.*, the agent finds women's clothes of the correct size but incorrect type or color.

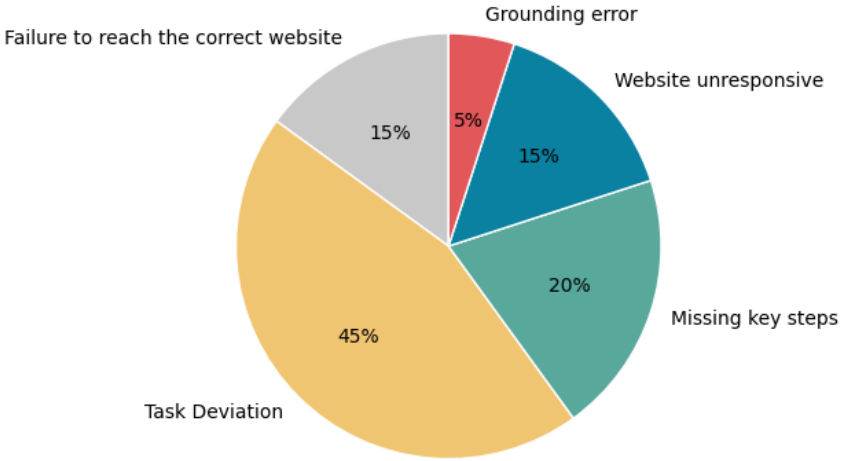



Figure A.1: Statistics for different error cases in Mind2Web-Live evaluation. Task deviation is the most prevalent error type.

| Model | Avg. Step SR (%) | Completion Rate (%) | Task SR (1) (%) | Full Task SR (%) |
|---|---|---|---|---|
| **API-based Models** | | | | |
| GPT-4o | 56.4 | 50.4 | 44.2 | 22.1 |
| GPT-3.5 | – | 36.5 | – | 15.4 |
| **Open-source Instructed Models** | | | | |
| Mistral-7B-Instr. (Jiang et al., 2023) | 33.0 | 28.6 | 25.0 | 11.5 |
| Qwen2-72B-Instr. (Bai et al., 2023) | – | **40.9** | – | 15.4 |
| Qwen2-VL-7B (Wang et al., 2024a) | 37.9 | 33.3 | 31.7 | 12.5 |
| Phi-3.5V (Abdin et al., 2024) | 27.0 | 22.3 | 21.2 | 1.9 |
| **Supervised Fine-Tuning** | | | | |
| **Explorer-4B** | 41.6 | 36.7 | 30.8 | 16.4 |
| **Explorer-7B** | **42.0** | 36.9 | **32.7** | **16.4** |

Table A.1: Results on Mind2Web-Live benchmark. The results for GPT-4, GPT-3.5, and Mistral-7B have been reproduced on our Linux servers. The full task success rate (SR) represents the successful completion of all key nodes for a given task. The average step success rate represents the proportion of completed key nodes, macro-averaged across tasks. The completion rate represents the proportion of completed key nodes, micro-averaged across tasks. Task SR (1) represents task SR with a tolerance of up to one error/key node. Our Phi-3.5V model, finetuned on synthetic trajectory data from Explorer, outperforms much larger models, including Mistral-7B and Qwen2-72B-Instruct, by a significant margin and is comparable to GPT-3.5.

| Dataset | Model | Train Data | Hyperparamerters | Train time (hours) |
|---|---|---|---|---|
| M2W-Live | Qwen2-VL-7B | Syn. | `batch_size:64, epoch:2, learning_rate:`$1 \times 10^{-5}$ | 15 |
| | Qwen2-VL-7B | M2W | `batch_size:64, epoch:2, learning_rate:`$1 \times 10^{-5}$ | 1.5 |
| | Qwen2-VL-7B | Syn. + M2W | `batch_size:64, epoch:2, learning_rate:`$1 \times 10^{-5}$ | 15.5 |
| M2W-Live | Phi-3.5V | Syn. | `batch_size:64, epoch:2, learning_rate:`$4 \times 10^{-5}$ | 12.5 |
| | Phi-3.5V | M2W | `batch_size:64, epoch:2, learning_rate:`$1 \times 10^{-5}$ | 1 |
| | Phi-3.5V | Syn. + M2W | `batch_size:64, epoch:2, learning_rate:`$4 \times 10^{-5}$ | 12.5 |
| Multi.-M2W | Qwen2-VL-7B | Syn. | `batch_size:64, epoch:10, learning_rate:`$4 \times 10^{-5}$ | 17 |
| | Phi-3.5V | Syn. | `batch_size:64, epoch:10, learning_rate:`$4 \times 10^{-5}$ | 12 |

Table A.2: Hyperparameters used in our experiments.

| Model | Avg. Step SR (%) | Completion Rate (%) | Full Task SR (%) |
|---|---|---|---|
| LLaVA-Mistral-7B | 32.0 | 30.3 | 4.8 |
| Phi-3-mini (text-only) | 36.6 | 34.0 | 13.3 |
| Phi-3.5V | 44.0 | 39.4 | 18.1 |
| Qwen2-VL-7B | **45.3** | **40.2** | **19.3** |

Table A.3: Ablation studies on language models used for fine-tuning (Mind2Web-Live).

- **Grounding error**: The agent fails to interact with a valid element on the page.
- **Website unresponsive**: The agent executes the correct action, but the website does not respond.
- **Failure to reach the correct website**: This happens when the agent fails to output the correct website URL or use the search engine to arrive at the correct website.

Figure A.1 presents the statistics for these error types.

## B Cost Analysis

We use GPT-4o-turbo, which costs $2.5 per 1M tokens for our trajectory synthesis. Each proposal or refinement stage uses 3.6K textual tokens on average. Each input image costs $0.0028. The calculation assumes an average of 7.7 steps per trajectory, including the proposal stage. Table B.4 shows the breakdown for the different stages of trajectory generation.

$$\text{Total cost} = \$0.0128 * 7.7 + \$0.02581 + \$0.02381 = \$0.148$$

The average cost per raw trajectory is $0.15. The success rate is estimated as 53.1%. Thus, the average cost per successful trajectory is estimated to be $0.28.

| Phase | Cost per step | Total cost |
|---|---|---|
| Proposal | $0.0128 | $0.0128 |
| Refinement | $0.0128 | $0.0856 |
| Verification | $0.02381 | $0.02381 |
| Summarization | $0.02581 | $0.02581 |

Table B.4: Cost breakdown for different modules in the pipeline.

| Complexity level | Count |
|---|---|
| Easy | 8.2K |
| Medium | 44.3K |
| Hard | 41.2K |

Table B.5: Task complexity statistics. Most tasks fall within the medium to high complexity range.

## C Task Complexity Analysis

Explorer contains web trajectories spanning multiple steps with an average of 7.7 steps per trajectory (Table 3). Following prior work (Zheng et al., 2024), we use the number of action steps in a trajectory as a proxy for task complexity, categorizing tasks as *easy* (2–4 steps), *medium* (5–7 steps), and *hard* (8–12 steps) in Table B.5. We observe that the majority of tasks fall within the medium to high difficulty range, indicating a high level of complexity in the dataset.

## D System Prompts

The prompts for the task proposer agent, task refiner agent, task summarizer agent, task verifier agent, and captcha detection agent are given in Table D.7, Table D.9, Table D.10, Table D.11, and Table D.12, respectively. The training prompt for Explorer is given in Table D.13.

**System Role**

What does this webpage show? Imagine you are a real user on this webpage. Given the webpage screenshot and parsed HTML/accessibility tree, please provide a single task that a user might perform on this page and the corresponding first action towards completing that task.
Do the following step by step:
1. Generate a single task that a user might perform on this webpage. Be creative and come up with diverse tasks.
2. Given the webpage screenshot and parsed HTML/accessibility tree, generate the first action towards completing that task (in natural language form).
3. Given the webpage screenshot, parsed HTML/accessibility tree, and the natural language action, generate the grounded version of that action.

---

**ACTION SPACE**: Your action space is: [‘click [element ID]’, ‘type [element ID] [content]’, ‘select [element ID] [content of option to select]’, ‘scroll [up]’, ‘scroll [down]’, and ‘stop’].
Action output should follow the syntax as given below:
‘click [element ID]’: This action clicks on an element with a specific ID on the webpage.
‘type [element ID] [content]’: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing. Both the content and the ID should be within square braces as per the syntax.
‘select [element ID] [content of option to select]’: Select an option from a dropdown menu. The content of the option to select should be within square braces. When you get (select an option) tags from the accessibility tree, you need to select the serial number (element_id) corresponding to the select tag, not the option, and select the most likely content corresponding to the option as input.
‘scroll [down]’: Scroll the page down.
‘scroll [up]’: Scroll the page up.

---

**IMPORTANT**: To be successful, it is important to STRICTLY follow the below rules:

**Action generation rules**:
1. You should generate a single atomic action at each step.
2. The action should be an atomic action from the given vocabulary - click, type, select, scroll (up or down), or stop.
3. The arguments to each action should be within square braces. For example, "click [127]", "type [43] [content to type]", "scroll [up]", "scroll [down]".
4. The natural language form of action (corresponding to the field "action_in_natural_language") should be consistent with the grounded version of the action (corresponding to the field "grounded _action"). Do NOT add any additional information in the grounded action. For example, if a particular element ID is specified in the grounded action, a description of that element must be present in the natural language action.
5. If the type action is selected, the natural language form of action ("action_in_natural_language") should always specify the actual text to be typed.
6. You should issue a “stop” action if the current webpage asks to log in or for credit card information.
7. To input text, there is NO need to click the textbox first, directly type content. After typing, the system automatically hits the ‘ENTER’ key.
8. STRICTLY Avoid repeating the same action (click/type) if the webpage remains unchanged. You may have selected the wrong web element.
9. Do NOT use quotation marks in the action generation.

---

**Task proposal rules**:
1. You should propose tasks that are relevant to the website and can be completed using the website.
2. You should only propose tasks that do not require login to execute the task.
3. You should propose tasks that are clear and specific.
4. For each task, provide concrete information or constraints, and use mock-up information (identifier, number, personal information, name, attributes, etc.) to make the task more specific and realistic.
5. The task description should provide all the necessary information to complete the task.
6. The task should be feasible to complete by a real user and should not require any additional information that is not available on the website.

The output should be in below format:

*Continued on next page*

*Continued from previous page*

| | |
|---|---|
| | **OUTPUT FORMAT**: Please give a short analysis of the screenshot, parsed HTML/accessibility tree, then put your answer within ``` ```, for example, "In summary, the proposed task and the corresponding action is: ```{"task": <TASK>:str, "action_in_natural_language":<ACTION_IN_NATURAL_LANGUAGE>:str, "grounded_action": <ACTION>:str}"``` |
| **User Role** | Website URL: {INIT_URL}<br>Parsed HTML/Accessibility Tree: {A11Y_TREE}<br>{SCREENSHOT} |

Table D.7: Prompt for Task Proposer Agent.

**System Role**

What does this webpage show? Imagine you are a real user on this webpage, and your overall task is `{OVERALL_TASK}`. This is the list of actions you have performed that lead to the current page `{PREV_ACTION_LIST}`. You are also given the webpage screenshot and parsed HTML/accessibility tree.
Do the following step by step:
1. Please predict what action the user might perform next that is consistent with the previous action list in natural language.
2. Then based on the parsed HTML/accessibility tree of the webpage and the natural language action, generate the grounded action.
3. Update the overall task aligned with this set of actions.

---

**ACTION SPACE**: Your action space is: [‘click [element ID]’, ‘type [element ID] [content]’, ‘select [element ID] [content of option to select]’, ‘scroll [up]’, ‘scroll [down]’, and ‘stop’].
Action output should follow the syntax as given below:
‘click [element ID]’: This action clicks on an element with a specific id on the webpage.
‘type [element ID] [content]’: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing. Both the content and the id should be within square braces as per the syntax.
‘select [element ID] [content of option to select]’: Select an option from a dropdown menu. The content of the option to select should be within square braces. When you get (select an option) tags from the accessibility tree, you need to select the serial number (element_id) corresponding to the select tag, not the option, and select the most likely content corresponding to the option as input.
‘scroll [down]’: Scroll the page down.
‘scroll [up]’: Scroll the page up.

---

**IMPORTANT**: To be successful, it is important to STRICTLY follow the below rules:

**Action generation rules**:
1. You should generate a single atomic action at each step.
2. The action should be an atomic action from the given vocabulary - click, type, select, scroll (up or down), or stop.
3. The arguments to each action should be within square braces. For example, "click [127]", "type [43] [content to type]", "scroll [up]", "scroll [down]".
4. The natural language form of action (corresponding to the field "action_in_natural_language") should be consistent with the grounded version of the action (corresponding to the field "grounded _action"). Do NOT add any additional information in the grounded action. For example, if a particular element ID is specified in the grounded action, a description of that element must be present in the natural language action.
5. If the type action is selected, the natural language form of action ("action_in_natural_language") should always specify the actual text to be typed.
6. You should issue the “stop” action when the given list of input actions is sufficient for a web task.
7. You should issue a “stop” action if the current webpage asks to log in or for credit card information.
8. To input text, there is NO need to click the textbox first, directly type content. After typing, the system automatically hits the ‘ENTER’ key.
9. STRICTLY Avoid repeating the same action (click/type) if the webpage remains unchanged. You may have selected the wrong web element.
10. Do NOT use quotation marks in the action generation.

---

**Task proposal rules**:
1. You should propose tasks that are relevant to the website and can be completed using the website itself.
2. The overall task should be well-aligned to the entire set of actions in history plus the current generated action. It should not be focused just on the current action.
3. You should only propose tasks that do not require login to execute the task.
4. You should propose tasks that are clear and specific.
5. For each task, provide concrete information or constraints, and use mock-up information (identifier, number, personal information, name, attributes, etc.) to make the task more specific and realistic.
6. The task description should provide all the necessary information to complete the task.
7. The task should be feasible to complete by a real user and should not require any additional information that is not available on the website.

The output should be in below format:

*Continued on next page*

*Continued from previous page*

| | |
|---|---|
| | **OUTPUT FORMAT**: Please give a short analysis of the screenshot, parsed HTML/accessibility tree, and history, then put your answer within ``` ```, for example, "In summary, the proposed task and the corresponding action is: ```{"task": <TASK>:str, "action_in_natural_language":<ACTION_IN_NATURAL_LANGUAGE>:str, "grounded_action": <ACTION>:str}"``` |
| **User Role** | Website URL: {INIT_URL}<br>Parsed HTML/Accessibility Tree: {A11Y_TREE}<br>{SCREENSHOT} |

Table D.9: Prompt for Task Refiner Agent.

| | |
|---|---|
| **System Role** | Given a list of actions performed on the website `{WEBSITE_URL}` and the corresponding screenshots<br>List of actions: `{ACTION_LIST}`<br>Your task is to come up with a single task description that will be accomplished by performing these actions in the given sequence on the website.<br><br>**IMPORTANT**:<br>1. The task must contain some actions: "Buy, Book, Find, Check, Choose, show me, search, browse, get, compare, view, give me, add to cart, ...", ideally involving transactions/finding information on a specific product or service.<br>2. You should propose tasks that are clear and specific.<br>3. The task description should provide all the necessary information to complete the task.<br>4. The task description must indicate the domain of the website at the end of the task with the format: "... on task website", for instance, "Purchase a laptop on Amazon", "Book a hair appointment on Yelp", etc.<br>5. The task should be feasible to complete by a real user and should not require any additional information that is not specified in this input.<br>6. The task description should specify constraints like given budget, product features, and other specifications that can narrow down the search to a particular item/product.<br>7. Do NOT use any quotation marks (either single or double) in the task description.<br><br>The output should be in the below format:<br>**OUTPUT FORMAT**: Please first give some analysis of the actions and screenshots and then output the overall task description. put your answer within ``` ```, for example, "In summary, the answer is: ```<TASK_DESCRIPTION>:str```". |
| **User Role** | `{SCREENSHOT_LIST}` |

Table D.10: Prompt for Task Summarizer Agent.

| | |
|---|---|
| **System Role** | You are an expert in evaluating the performance of a web navigation agent. The agent is designed to help a human user navigate a website to complete a task. Given the user's intent, the agent's action history, and the final state of the webpage, your goal is to decide whether the agent's execution is successful or not.<br>There are four types of tasks:<br><br>1. **Transaction**: The user wants to perform a transaction on the webpage, such as booking a ticket, ordering a product, etc. The bot should at least initiate the add-to-cart or checkout process. It is still a success if the bot has done actions of 'add to cart' or checkout and encounters the login page. If the bot fails to do so, the task is considered a failure.<br><br>2. **Information seeking**: The user wants to obtain certain information from the webpage, such as information of a product, reviews, map info, comparison of map routes, etc. The bot's response must contain the information the user wants, or explicitly state that the information is not available. Otherwise, e.g. the bot encounters an exception and responds with the error content, the task is considered a failure. Besides, be careful about the sufficiency of the agent's actions. For example, when asked to list the top-searched items in a shop, the agent should order the items by the number of searches, and then return the top items. If the ordering action is missing, the task is likely to fail.<br><br>3. **Site navigation**: The user wants to navigate to a specific page. Carefully examine the bot's action history and the final state of the webpage to determine whether the bot successfully completes the task. No need to consider the bot's response.<br><br>4. **Content modification**: The user wants to modify the content of a webpage or configuration. Carefully examine the bot's action history and the final state of the webpage to determine whether the bot successfully completes the task. No need to consider the bot's response. |
| | **IMPORTANT**<br>- If a product has been added to the bag/cart in the action list but just the purchase is pending, it should be counted as a success.<br>- If you see the checkout page for the product you want to purchase, it should be counted as a success.<br>- Format your response into two lines as shown below:<br><br>**Thoughts**: <your thoughts and reasoning process><br>**Status**: "success" or "failure" |
| **User Role** | User Intent: `{TASK_DESCRIPTION}`<br>Action History: `{ACTION_HISTORY}`<br>The content of the last webpage in markdown format is given below: `{LAST_PAGE_MARKDOWN}`<br>The snapshots of all webpages corresponding to the actions are shown in the images: `{SCREENSHOT_LIST}` |

Table D.11: Prompt for Task Verifier Agent (adapted from Pan et al. (2024a)).

| | |
|---|---|
| **System Role** | You are an expert in evaluating whether the given webpage screenshot contains a captcha or not. Given the last snapshot of the web page, your goal is to decide whether the webpage contains a captcha or not.<br>Output "Yes" if the given webpage shows a captcha, otherwise "No". |
| | **IMPORTANT**:<br>Format your response into a line as shown below:<br>Answer: "Yes" or "No" |
| **User Role** | The screenshot of the web page is shown in the image.<br>`{SCREENSHOT}` |

Table D.12: Prompt for Captcha Detection Agent.

| | |
|---|---|
| **System Role** | You are an expert at completing instructions on webpage screens.<br>You will be presented with a screenshot image with some numeric tags.<br>If you decide to click somewhere, you should choose the numeric element index closest to the location you want to click.<br>You should decide the action to continue this instruction. You will be given the accessibility tree of the current screen in the format: `[element_idx] [role] [alt text or button name]`.<br>Here are the available actions:<br>`{"action": "goto", "action_natural_language": str, "value": <the URL to go to>}`<br>`{"action": "google_search", "action_natural_language": str, "value": <search query for google>}`<br>`{"action": "click", "action_natural_language": str, "idx": <element_idx>}`<br>`{"action": "type", "action_natural_language": str, "idx": <element_idx>, "value": <the text to enter>}`<br>`{"action": "select", "action_natural_language": str, "idx": <element_idx>, "value": <the option to select>}`<br>`{"action": "scroll [up]", "action_natural_language": str}`<br>`{"action": "scroll [down]", "action_natural_language": str}`<br>Your final answer must be in the above format. |
| **User Role** | The instruction is to {TASK_DESCRIPTION}.<br>History actions: {PREVIOUS_ACTION_LIST}<br>Here is the screen information: {A11Y_TREE}<br>Think about what you need to do with the current screen, and output the action in the required format in the end. |

Table D.13: Prompt for web agent training.

## E Trajectory Examples

Figure E.2 shows a sample trajectory executed on the IKEA website. Figure E.3 shows the set-of-mark annotations and accessibility tree inputs of the model during trajectory generation, training, and inference. The action space for our trajectory synthesis pipeline is given in Table E.14.

## F More Related Work

### F.1 LLM-based Web Agents

Recent advances in multimodal language models have facilitated the development of web agents — autonomous systems designed to interact with real-world websites to perform everyday tasks (Deng et al., 2023; Hong et al., 2024; Cheng et al., 2024; Zheng et al., 2024). Web agents have made significant progress, evolving from simulated environments (Liu et al., 2018) to complex real-world applications (Deng et al., 2023; Yao et al., 2022; Zhou et al., 2024). Key challenges for web agents include long-term planning, visual grounding, and memory management. To improve long-context understanding, WebAgent (Gur et al., 2024) utilizes multiple LLMs - one for planning, summarization, and grounded program synthesis. SeeAct (Zheng et al., 2024) adopts a two-step procedure of planning followed by grounding at each step using GPT-4 to accomplish web agent tasks. SkillWeaver (Zheng et al., 2025) introduces a self-improving agent framework that autonomously synthesizes reusable skills as APIs through iterative exploration. Another line of work employs a vision-only approach to train a GUI grounding model that directly predicts pixel coordinates for executing GUI agent tasks (Cheng et al., 2024; Kapoor et al., 2024; Gou et al., 2025). However, a significant bottleneck remains — the lack of large-scale, high-quality web trajectory data for training robust agents. Our work presents a new framework for synthesizing large-scale web trajectory data to train end-to-end web agents.

### F.2 Web Agent Benchmarks and Datasets

Early benchmarks for web tasks such as MiniWob++ (Liu et al., 2018) focused on testing low-level actions on simulated websites. However, these simulated websites fail to capture the complexity of the real-world web. Mind2Web (Deng et al., 2023) introduces a trajectory-level dataset with 2K tasks across 137 real-world websites and 31 domains. However, it employs a static evaluation method that penalizes alternative valid execution paths. To overcome this limitation, follow-up work has explored alternative evaluation approaches, including functional correctness-based evaluation in WebArena (Zhou et al., 2024) and key-node-based evaluation in Mind2Web-Live (Pan et al., 2024b). Most recently, Online-Mind2Web (Xue et al., 2025) addresses challenges in online evaluation, such as sensitivity to website updates, and demonstrates improved agreement with human judgments, providing a more reliable benchmark for real-world web agent evaluation. Towards the goal of making web agents more capable of performing realistic tasks, GAIA (Mialon et al., 2024) and AssistantBench (Yoran et al., 2024) introduce benchmarks that include time-consuming information-seeking tasks. In this work, we develop Explorer, a multimodal web agent trained on our synthetic dataset, and showcase its strong performance across online and offline benchmarks, including Mind2Web-Live, Multimodal-Mind2Web, and MiniWob++.

| Action Type | Description | Count |
|---|---|---|
| click [elem] | Click on elem. | 415K |
| type [elem] [text] | Type text | 62K |
| select [elem] [text] | Select text from dropdown list. | 5K |
| goto [url] | Go to url. | 26K |
| search_google [query] | Search for query on Google. | 4K |
| scroll [up/down] | Scroll up or down. | 213K |

Table E.14: Action space for web navigation in Explorer.

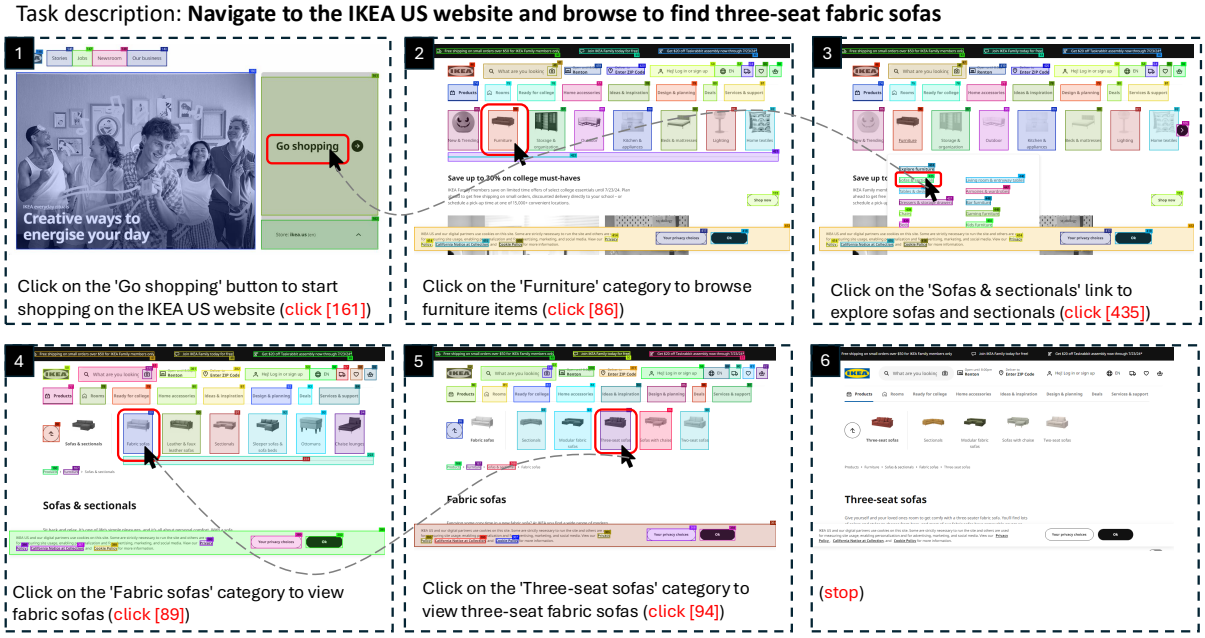
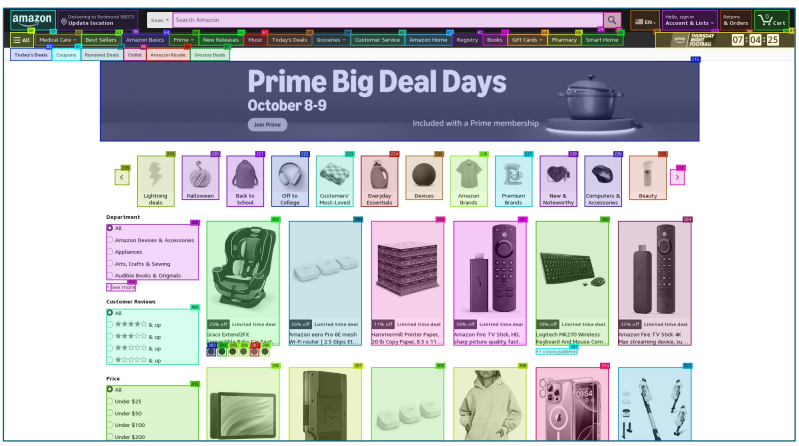



Figure E.2: Example synthetic trajectory from Explorer. Each step shows the set-of-mark annotated screenshot along with the grounded action taken by the GPT-4 agent.




(a) Set-of-mark annotated screenshot of webpage

```
[17] [A] [Amazon]
[18] [A] [Delivering to Redmond 98073 Update location]
[24] [INPUT, TYPE=TEXT] [Search Amazon]
[25] [INPUT, TYPE=SUBMIT] []
[26] [A] [Choose a language for shopping.]
[27] [A] [Hello, sign in Account & Lists]
[28] [A] [Returns & Orders]
[29] [A] [0 items in cart]
[50] [A] [Open Menu]
[52] [A] [Medical Care]
[53] [A] [Best Sellers]
[54] [A] [Amazon Basics]
[55] [A] [Prime]
[56] [A] [New Releases]
...
[289] [A] [36% off Limited time deal  Amazon eero Pro 6E
mesh Wi-Fi router | 2.5 Gbps Ethernet | Coverage up to
6,000 sq. ft | Connect 100+ devices | Ideal for streaming,
working, and gaming | 3-Pack | 2022 release Amazon eero
Pro 6E mesh Wi-Fi router | 2.5 Gbps Et\u2026]
...
[408] [DIV] [Department]
[409] [DIV] [Customer Reviews]
[410] [DIV] [Price]",
```

(b) Corresponding A11y tree

Figure E.3: Visualization of the model inputs during trajectory generation, model training, and inference. The example corresponds to step 2 of the trajectory in Figure 1.