Go-Browse: Training Web Agents with Structured Exploration

Apurva Gandhi **Graham Neubig**
Carnegie Mellon University
Pittsburgh, PA
{apurvag,gneubig}@cs.cmu.edu

Abstract

One of the fundamental problems in digital agents is their lack of understanding of their environment. For instance, a web browsing agent may get lost in unfamiliar websites, uncertain what pages must be visited to achieve its goals. To address this, we propose Go-Browse, a method for automatically collecting diverse and realistic web agent data at scale through structured exploration of web environments. Go-Browse achieves efficient exploration by framing data collection as a graph search, enabling reuse of information across exploration episodes. We instantiate our method on the WebArena benchmark, collecting a dataset of 10K successful task-solving trajectories and 40K interaction steps across 100 URLs. Fine-tuning a 7B parameter language model on this dataset achieves a success rate of 21.7% on the WebArena benchmark, beating GPT-4o mini by 2.4% and exceeding current state-of-the-art results for sub-10B parameter models by 2.9%.[1]

1 Introduction

Despite their impressive and often superhuman performance in other domains, most pretrained LLMs do not perform well on GUI-based web agent tasks. For instance, on the WebArena benchmark [29] where humans achieve a 78% success rate, frontier models like `GPT-4o` [11] and `GPT-4o-mini` [12] score only 38% and 19% respectively, while a smaller model like `Qwen-2.5-7B-Instruct` [26] scores only 8%. On the other hand, models trained specifically for GUI-based interaction score much better, with Anthropic's `Claude-3.7-Sonnet` [1] scoring 45.4% and OpenAI's `Computer-Using Agent` achieving 58% [13]. This gap suggests that training on agent-specific interaction data is crucial for realizing effective web agents.

But collecting high-quality web agent data presents its own set of challenges. Human-generated trajectories offer one source for quality demonstrations but are notoriously expensive, time-consuming, and ultimately unscalable to collect for the vast datasets required. One class of methods tries to automatically scale human-generated data or use humans-in-the-loop in the dataset collection process [17; 30; 8]. Another line of work attempts to improve scalability further by proposing fully unsupervised and automatic methods for data generation; for example, by generating synthetic demonstrations from general wikiHow-style tutorial articles [14] or by building an exploration policy that collects data by actively interacting with websites [9; 10].

Among these unsupervised methods, the latter ones that directly explore web environments of interest perform significantly better than those that use indirect and more generic knowledge from the internet (16% [10] vs. 6% [14] success rate). This gap underscores a fundamental problem in digital agents: their lack of prior understanding of the environments they are deployed on. Learning from a tutorial

[1] We release our code, dataset and models at `https://github.com/ApGa/Go-Browse`.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Scaling Environments for Agents (SEA).

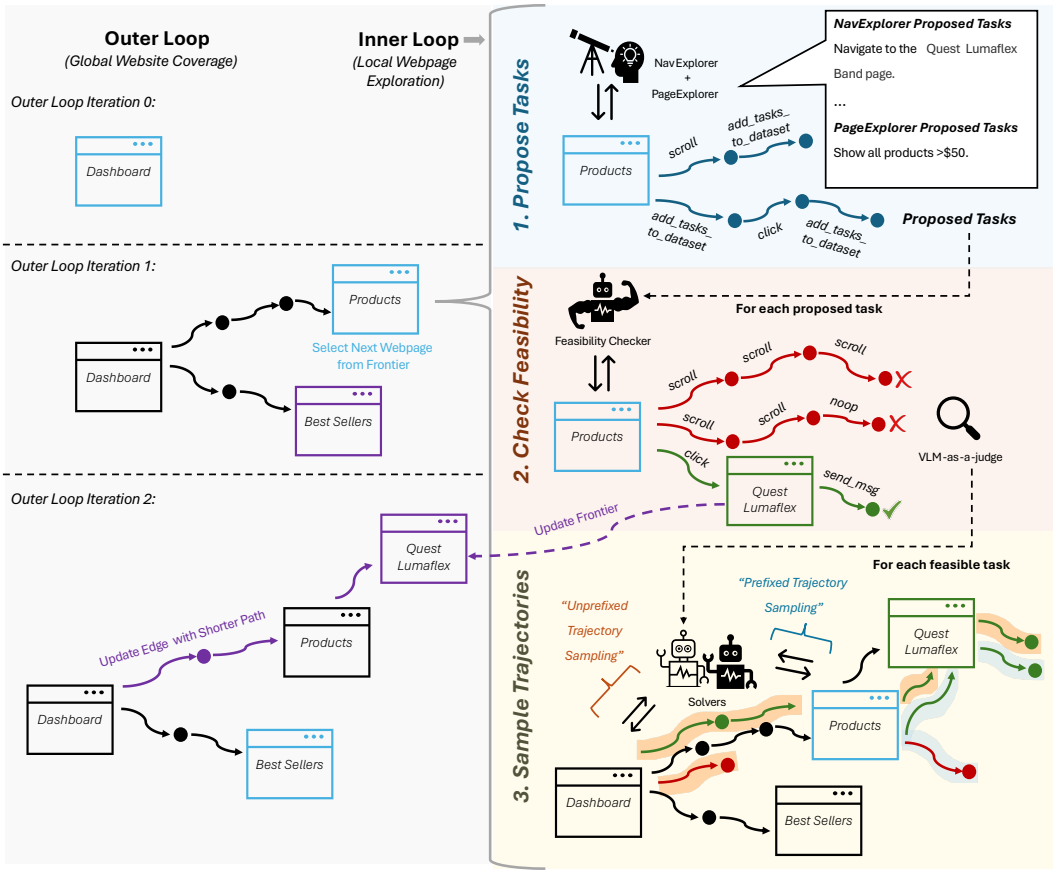


Figure 1: Overview of the `Go-Browse` algorithm for web agent data collection for a website. `Go-Browse`'s outer-loop (left) maintains an exploration frontier of discovered but not yet fully explored webpages. `Go-Browse`'s inner loop (right) explores each webpage in the frontier by (1) Proposing tasks for that webpage that are grounded in interaction; (2) Checking the feasibility of those tasks; and (3) Sampling trajectories and discovering new webpages by solving feasible tasks.

or even a human-generated demonstration on how to cancel an ongoing order on Amazon is unlikely to transfer to the myriad of other websites that a web agent may need to interact with. Instead, agents are likely to be more successful if they learn directly from environments they will encounter.

In this work, we introduce `Go-Browse`, a method that automatically collects diverse, realistic, and tailored web agent data through systematic and structured exploration of websites. In particular, `Go-Browse` iteratively builds up a graph of previously visited URLs while collecting data. This allows us to reuse information across exploration episodes, resetting new episodes to previously discovered promising webpages, and continuing exploration from there. This improves exploration efficiency over previous unsupervised data collection methods [10; 9], which have minimal reuse of information across episodes. Furthermore, resetting to previously discovered webpages allows `Go-Browse` to decouple the challenge of web navigation (finding the correct page) from that of local task solving (performing actions on that page). We demonstrate that this decoupling facilitates a *bootstrapping* effect, enabling even weaker pretrained LLMs to collect higher-quality data, since local task execution is often less demanding than website navigation, which often requires domain-specific knowledge. `Go-Browse` draws inspiration from previous RL works like `Go-Explore` [6; 7], which used analagous reset-then-explore strategies to solve games like Montezuma's Revenge that have notoriously difficult exploration challenges.

To evaluate `Go-Browse`, we instantiate it on the WebArena benchmark, collecting a dataset of $\sim$10K successful task-solving trajectories as well as $\sim$17K unsuccessful trajectories across 100 distinct URLs. Finetuning `Qwen-2.5-7B-Instruct` on our dataset achieves a task success rate of 21.7% on WebArena. This result surpasses `NNetNav` [10], the current state-of-the-art results for sub-10B parameter models by 2.9% and beats `GPT-4o-mini` by 2.4%.

2 Background

2.1 LLM Web Agents

Following previous work [4; 5; 29], our web agents are implemented using the ReAct pattern [27] where at each timestep t we prompt the LLM with a state s_t and ask it to produce an action a_t. Executing a_t in the browser environment generates a new state which we observe as s_{t+1}.

We include several components in each s_t: the task or goal g, a flattened accessibility tree representation of the current webpage, a description of the action space, the history of previous actions and any errors encountered when executing the last action. The action space includes primitive operations represented as python functions like `click(id)`, `scroll(dx, dy)`, `type(text)` and `send_msg_to_usr(msg)`, which the agent uses to interact with the browser environment. Each action a_t produced by the LLM consists of a chain-of-thought and a python function call. The complete action space and prompt template are detailed in the Appendix A.

A trajectory $\tau = \{s_1, a_1, s_2, a_2, ..., s_T, a_T\}$ represents a sequence of states and actions taken by the agent in attempting to complete a task. Trajectories terminate either when a maximum horizon length T is reached or when the agent performs a terminal action (such as `send_msg_to_usr(msg)`). For each trajectory, we define a reward model $R(g, \tau) \in \{0, 1\}$ that evaluates task completion success. The binary reward indicates whether the agent successfully completed the specified task ($R(g, \tau) = 1$) or failed ($R(g, \tau) = 0$) by the end of the trajectory. In this work, we leverage the `BrowserGym` [4; 5] python package to implement our web agent.

2.2 Exploration Policies for Data Collection

To collect web agent data in an environment through direct interaction, we need to design a method that can explore the environment effectively to gather diverse and high-quality demonstrations, which we refer to as an *exploration policy*. We can classify past work on building exploration policies into two main categories: *interaction-first* and *instruction-first*.

Interaction-first Exploration Policy. Interaction-first approaches [9; 10], such as `NNetNav` roll out an agent (e.g., a prompted pretrained LLM) to explore a website with general exploration instructions (e.g., persona simulation) instead of a concrete task (e.g., *Add Nintendo Switch to cart*). The collected trajectories are then labeled with concrete tasks in retrospect using another prompted LLM, which we call a Labeler (L). We call each rollout here an exploration episode. Algorithm 1 provides pseudocode for this process.

A benefit of interaction-first approaches is that the collected trajectories may potentially explore deeper parts of the website that might not be immediately apparent in the initial state. But since each exploration episode operates independently, there's significant redundancy in exploration—agents may often revisit the same parts of websites across different episodes, leading to similar task demonstrations. Additionally, without specific task guidance, agents may spend considerable time collecting trajectories that do not yield interesting or useful tasks.

Instruction-first Exploration Policy. Unlike interaction-first approaches, instruction-first exploration policies [8; 9; 30] first generate potential tasks and then attempt to solve them. In this approach, a prompted LLM task proposer P observes a state s_t and generates a set of plausible tasks $\mathcal{G} = \{g_1, g_2, ..., g_K\}$ grounded in the webpage's observed content and functionality. A pretrained policy A then attempts to solve each task g_i, generating trajectories $\tau_i = \{s_0, a_0, s_1, a_1, ...\}$ for each proposed task. Finally, a reward model $R(g_i, \tau_i) \in \{0, 1\}$ evaluates whether each trajectory successfully completes its corresponding task, and successful pairs (g_i, τ_i) where $R(g_i, \tau_i) = 1$ are added to the dataset $\mathcal{D}$. Algorithm 2 provides pseudocode for this approach.

This approach leverages an LLM's prior knowledge to efficiently generate diverse, useful and contextually relevant tasks, but has limitations: proposed tasks are typically limited to the currently observed page, and the LLM may occasionally hallucinate infeasible tasks about unobserved parts of the website. This is because task proposal in these methods is typically anchored to an initial static observation. In order to address this, works like PAE [30] require screenshots of human demonstrations across the website to gain additional context for task proposal; instead, our work implements the task proposer P with agents that can explore and gather their own context automatically.

Algorithm 1 Interaction-first Exploration

$A \leftarrow$ Agent()
$L \leftarrow$ Labeler()
$\mathcal{D} \leftarrow \emptyset$

for website $W \in \mathcal{W}$ **do**
  for $1 \dots N$ iterations **do**
    $s_0 \leftarrow$ InitialState(W)
    $g \leftarrow$ Exploration instructions.
    $\tau \leftarrow$ SampleTrajectory(A, s_0, g)
    $\mathcal{D}_\tau \leftarrow L(\tau)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\tau$
  end for
end for

return $\mathcal{D}$

Algorithm 2 Instruction-first Exploration

$P \leftarrow$ TaskProposer()
$A \leftarrow$ Agent()
$R \leftarrow$ RewardModel()
$\mathcal{D} \leftarrow \emptyset$
for website $W \in \mathcal{W}$ **do**
  $s_0 \leftarrow$ InitialState(W)
  $\mathcal{G} \leftarrow P(s_0,)$
  for task $g \in \mathcal{G}$ **do**
    $\tau \leftarrow$ SampleTrajectory(A, s_0, g)
    if $R(g, \tau) = 1$ **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(g, \tau)\}$
    end if
  end for
end for
return $\mathcal{D}$

Figure 2: Comparison of common styles of exploration policies for web agent data collection.

3 Go-Browse

We propose `Go-Browse`, which addresses the limitations of past interaction-first and instruction-first approaches. For each website of interest, `Go-Browse` builds a systematic map of previously discovered webpages by treating exploration as a graph traversal problem. It maintains an exploration frontier of discovered but not yet fully explored webpages and progressively explores and expands this frontier by proposing and solving tasks that encourage both local webpage exploration and finding new webpages for global frontier expansion.

Fig. 1 illustrates the `Go-Browse` algorithm and Algorithm 3 provides pseudocode. Specifically, `Go-Browse` builds up a graph $G = (\mathcal{V}, \mathcal{E})$, where nodes $v \in \mathcal{V}$ are unique URLs and edges $e \in \mathcal{E}$ are trajectories between them. As shown in Fig. 1, its outer loop (left) resembles graph traversal (e.g., breadth-first search), while an inner loop (right) resembles instruction-first exploration.

In each outer loop iteration, `Go-Browse` first selects a webpage v from the frontier and then performs the following inner loop to collect data and explore v: (1) Propose navigational and local tasks for v using the `NavExplorer` and `PageExplorer` modules, (2) Check feasibility of proposed tasks with the `FeasibilityChecker` module, and (3) Sample trajectories by solving feasible tasks with the `Solvers` module. We describe these modules in detail below. `Go-Browse`'s outer loop enforces global coverage of the website while the inner loop thoroughly explores each discovered webpage.

NavExplorer: Frontier Expansion and Navigational Task Collection. The `NavExplorer` module is responsible for proposing navigational tasks to webpages that neighbor the current webpage v in the graph. This is similar to the `TaskProposer` module in instruction-first approaches, but instead of just asking the LLM to propose tasks from a static observation, we instead implement NavExplorer as a web agent itself. We instruct it with a goal g to find neighboring webpages through interaction with the current webpage, and propose navigational tasks to reach them. We do the latter by extending the action space of the `NavExplorer` agent with an `add_tasks_to_dataset(tasks: tuple[str])` function. Designing `NavExplorer` as a web agent empowers it to perform its own purposeful exploration and ground proposed tasks on dynamically obtained observations. To keep the exploration process efficient and prioritize nodes added to the frontier, `NavExplorer` is asked to prioritize adding tasks for navigating to new webpages that are likely to have common and useful tasks that a user might want to perform. Full prompts for the `NavExplorer` modules are provided in Appendix A.2.

PageExplorer: Local Page Exploration and Task Collection. The `PageExplorer` module is similar to the `NavExplorer`, except that it is responsible for proposing tasks local to the current webpage v. It does so by asking an LLM to generate a set of plausible tasks that a user may want to perform on the current webpage (prompts in Appendix A.2) The tasks generated by the `PageExplorer` help generate training data that thoroughly explore the functionality of each webpage.

FeasibilityChecker: Task Filtering and Trajectory Sampling. The `FeasibilityChecker` module filters the tasks proposed by the previous two modules. It does so by (1) using a strong pretrained LLM agent (e.g., computer-use trained LLM) to try and solve each task, and (2) using a pretrained

Algorithm 3 Go-Browse

Initialize Dataset $\mathcal{D} \leftarrow \emptyset$, Graph $G = (\mathcal{V}, \mathcal{E})$, Frontier $F \leftarrow \emptyset$
Initialize Modules: NavExplorer, PageExplorer, FeasibilityChecker, Solvers, RewardModel R
for each website $W_i \in \mathcal{W}$ **do**
    $v_{\text{root}} \leftarrow$ GetRootURL(W_i); Add v_{root} to F and $\mathcal{V}$
    while F is not empty **do**
        $v \leftarrow$ SelectAndRemoveFromFrontier(F)
        $s_v \leftarrow$ GetCurrentState(v)
                                                ▷ Propose navigation and local tasks
        $\mathcal{G}_{\text{nav}} \leftarrow$ NavExplorer.propose_tasks(s_v)
        $\mathcal{G}_{\text{local}} \leftarrow$ PageExplorer.propose_tasks(s_v)
        $\mathcal{G}_{\text{proposed}} \leftarrow \mathcal{G}_{\text{nav}} \cup \mathcal{G}_{\text{local}}$
        $\mathcal{G}_{\text{feasible}} \leftarrow \emptyset$
                                                ▷ Filter for feasible tasks and collect initial trajectories
        for task $g \in \mathcal{G}_{\text{proposed}}$ **do**
            (is_feasible, $\tau_{\text{fc}}, v_{\text{new}}$) $\leftarrow$ FeasibilityChecker.check_and_collect(g, s_v, R, N_{max})
            if is_feasible **then**
                Add (g, τ_{fc}) to $\mathcal{D}$; Add g to $\mathcal{G}_{\text{feasible}}$
                if v_{new} is a new discovered URL **then**
                    Add v_{new} to $\mathcal{V}$ and F; Add new edges to $\mathcal{E}$
                end if
            end if
        end for
                                                ▷ Sample additional prefixed and unprefixed trajectories
        for feasible task $g \in \mathcal{G}_{\text{feasible}}$ **do**
            $\mathcal{T}_{\text{prefixed}} \leftarrow$ Solvers.sample$(g, s_v, R,$ prefixed=True$)$
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(g, \tau) \mid \tau \in \mathcal{T}_{\text{prefixed}}\}$
            $s_{\text{root}} \leftarrow$ GetState(GetRootURL(W_i))
            $\mathcal{T}_{\text{unprefixed}} \leftarrow$ Solvers.sample$(g, s_{\text{root}}, R,$ prefixed=False$)$
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(g, \tau) \mid \tau \in \mathcal{T}_{\text{unprefixed}}\}$
        end for
    end while
end for

return $\mathcal{D}$

VLM-as-a-judge to check if the sampled trajectory solves the task. We sample up to N_{max} trajectories, stopping if we sample a success. Proposed tasks with at least one successful trajectory are considered feasible and kept in the dataset along with their corresponding trajectories, while the rest are discarded.

Solvers: Prefixed and Unprefixed Sampling. The `Solvers` sample additional trajectories for the filtered, feasible tasks, but can use cheaper models to sample a larger number of trajectories. Additionally, `Solvers` perform a mix of *prefixed* and *unprefixed* sampling. In prefixed sampling, the agent tries to solve g starting from the current webpage v, while in unprefixed sampling, the agent has to solve g starting from the root node of the webpage (e.g., usually the homepage or dashboard). Prefixed sampling makes the agent's job easier by decoupling navigation (finding the webpage) from task solving locally on that webpage. As we discuss in Section 6, prefixed sampling has higher success rates, letting us bootstrap from even weaker pretrained models. Still, it is useful to sample unprefixed trajectories to instill long-horizon task-solving and exploratory behaviors in the agent.

Relation to Instruction-First and Interaction-First Approaches. We can think of `Go-Browse`'s inner-loop interaction between the `NavExplorer`, `PageExplorer` and `FeasibilityChecker` as a form of *instruction-first* exploration. But unlike typical instruction-first approaches that only start at the root node (homepage, dashboard, etc.) of the website, `Go-Browse`'s inner-loop is initialized with new pages from the frontier at every iteration. This addresses the localized exploration of instruction-first approaches by enforcing global website coverage. Furthermore, by using web agents for task proposal, `Go-Browse` enables more grounded task proposal based on real observations. `Go-Browse` also addresses the exploration efficiency limitation of interaction-first approaches by reusing information from past episodes. Since each iteration of the outer-loop resets exploration to a previously discovered webpage, `Go-Browse` can reduce redundancy and instead spend more budget on exploring novel parts of the website.

4 Data Collection

We collect a dataset (`Go-Browse-WA`) by running `Go-Browse` on the WebArena benchmark [29], which consists of 5 self-hosted websites, representing clones of common domains: Shopping Admin (CMS), Shopping, Reddit, Gitlab, and Map. We explore 20 different URLs for each of the five domains, collecting tasks across 100 distinct URLs in total. While our evaluation focuses on these specific domains, we note that `Go-Browse` is a general-purpose algorithm that makes no assumptions tying it to WebArena: the same data collection pipeline can be run on other websites of interest.

For the `NavExplorer` we perform up to 15 steps of interaction with `Claude-3.7-Sonnet` [1]. For the `PageExplorer` we perform up to 20 steps with `GPT-4o` [11] and 10 steps with `Claude-3.7-Sonnet`. Appendix B.1 provides analysis of some of the complementary differences in behavior between these models. The `FeasibilityChecker` uses `Claude-3.7-Sonnet` to try solving proposed tasks, with a maximum of 3 tries, and uses a `GPT-4o`-based *"VLM-as-a-judge"* reward model (adopted from [24; 19]). We keep a maximum of 30 feasible tasks per URL. For the `Solvers` we use `GPT-4o-mini` and `Qwen-2.5-7B-Instruct`. Task solving is limited to a maximum horizon length of 10 steps. The `Solvers` sample 2 prefixed trajectories and 2 unprefixed ones. For all interaction steps in the dataset collection process we use a temperature of 0.7. Overall, `Go-Browse-WA` took $\sim \$975.57$ to collect; for a detailed cost analysis, see Appendix B.2.

Table 1 shows the composition statistics of the `Go-Browse-WA` dataset. The dataset contains roughly similar proportions of successful trajectories from each model we use to sample trajectories (Fig. 3). While here we only finetune on the success steps, we release all steps in our dataset, including failures. The dataset also includes multiple alternate representations of webpage observations (accessibility tree, HTML, and screenshots); although, we only use the accessibility tree for our finetuning experiments.

Table 1: Dataset statistics on the 5 WebArena domains (20 pages explored/domain).

	Success	Failure	Total
Trajectories	9,504	17,245	26,749
Steps	39,339	157,123	196,462
Unique tasks		3,422	

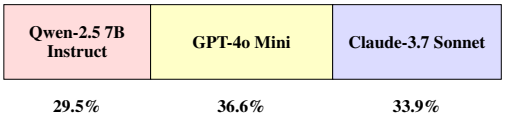


Figure 3: Proportion of successful trajectories from each model in the Go-Browse-WA dataset.

5 Fine-tuning Setup and Results

For our experiments, we train `Qwen-2.5-7B-Instruct` by performing supervised finetuning on only the success trajectories in our collected dataset. Finetuning hyperparameters are provided in Appendix C. For fair comparison, we also train a model using the same parameters on the `NNetNav-WA` dataset [10] which consists of 45K interaction steps across the 5 WebArena domains.

We benchmark the finetuned models on the 812 WebArena benchmark using `BrowserGym` [4]. Correctness of each task is evaluated using task-specific reward functions provided by WebArena. We use a temperature of 0 for the models when benchmarking.

Table 3 shows the success rates of the finetuned models on WebArena as well as other pretrained models. Our model, `Go-Browse-7B`, achieves a success rate of 21.7% on the overall WebArena tasks, outperforming the other models in the table. The `Go-Browse-7B` model outperforms the pretrained `Qwen-2.5-7B-Instruct` model by 13.4% and the finetuned `NNetNav-7B` model by 2.9%. Notably, `Go-Browse-7B` also outperforms `GPT-4o-mini` by 2.4%. Appendix B.3 provides results of performing statistical significance testing with paired bootstrap tests.

Looking at individual domains, `Go-Browse-7B` scores higher than `GPT-4o-mini` and `NNetNav-7B` in all domains except for Gitlab. Notably, `Go-Browse-7B` beats `NNetNav-7B` by 11% on the Shopping Admin domain and 7% on the Reddit domain.

We also evaluate our models on Online-Mind2Web [25], an out-of-domain benchmark with 300 tasks across 136 live websites. `Go-Browse-7B` still maintains a lead over `NNetNav-7B` even in this generalization experiment, though—as expected—models perform worse in this setting compared to the in-domain WebArena. `GPT-4o-mini` also scores much worse on Online-Mind2Web.

Model	**SR (%)**
NNetNav-7B	4.00
Go-Browse-7B	5.33
GPT-4o-mini	9.33

Table 2: Online-M2W results.

Table 3: Success rates on WebArena tasks. Bold indicates the best result in each category.

Model	Overall (%)	Admin (%)	Shopping (%)	Reddit (%)	Gitlab (%)	Map (%)
Closed Models.						
`GPT-4o-mini`	19.3	19.2	19.3	21.1	20.9	15.6
`GPT-4o`	37.6	35.7	32.3	50.9	36.7	37.5
`Claude-3.7-Sonnet`	**45.4**	**37.4**	**37.0**	**58.8**	**52.0**	**47.7**
Open-weights 7B Models.						
`Qwen-2.5-7B-Instruct`	8.3	7.1	9.4	7.9	8.7	7.8
`NNetNav-7B`	18.8	14.3	20.3	23.7	**19.9**	17.2
`Go-Browse-7B`	**21.7**	**25.3**	**22.4**	**30.7**	15.3	**17.9**

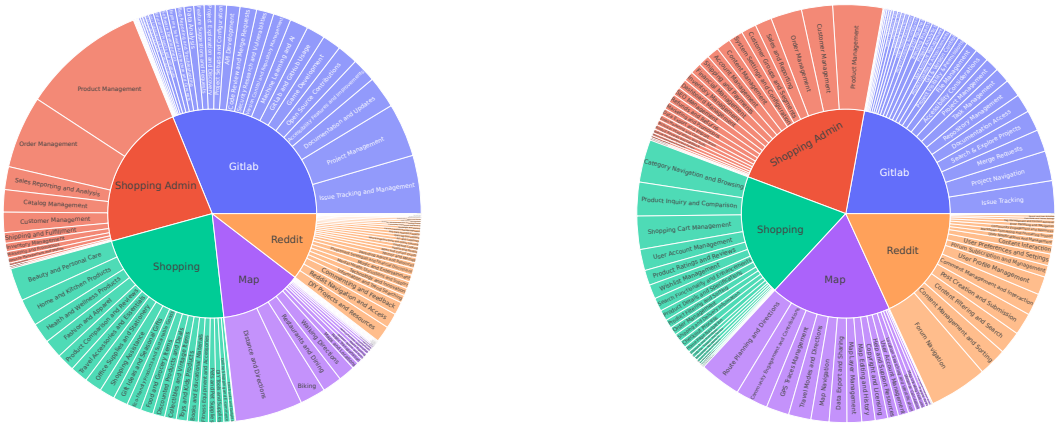
6 Analysis

Go-Browse Generates Diverse Tasks. Fig. 4 compares the distribution of tasks in the `NNetNav-WA` and `Go-Browse-WA` datasets. We follow Murty et al. [10] in using `GPT-4o-mini` to provide intent categories based on the dataset tasks and then classify each task into one of the categories. We can see that `NNetNav` shows a tendency to have larger wedges in its task distribution, indicating redundancy in exploration since each episode is independent. This pattern is particularly evident in domains that are challenging to navigate, such as Shopping Admin, because a larger number of episodes will navigate to the same easy-to-find webpages, leading to similar tasks; harder to find webpages, even if discovered by one episode, may rarely be revisited in a future episode. `Go-Browse` addresses this issue by resetting to previously encountered webpages. Even if a page is difficult to navigate to, once it is discovered, `Go-Browse` makes sure it is thoroughly explored in future episodes.

`Go-Browse-WA` also exhibits a more balanced distribution across domains; `NNetNav` contains a disproportionately large number of Gitlab tasks and relatively few Reddit tasks. These observations align with our model performance results in Table 3, where `NNetNav-7B` only outperforms `Go-Browse-7B` on the Gitlab domain, and performs notably worse on Shopping Admin and Reddit domains.

Go-Browse's Successful Trajectories Go Deeper. Fig. 5 plots how deep into the website the finetuned models go when solving tasks. On the left, when considering all trajectories, we see that both `Go-Browse` and `NNetNav` behave similarly, but on trajectories where only `Go-Browse` was successful (middle), we see that the depth distribution is more right-skewed, suggesting that `Go-Browse` owes some of its wins to its tendency to solve longer-horizon tasks. If we look at the `NNetNav`-only successful trajectories (right), we again see no significant difference in behavior, showing that going deeper is a unique characteristic of `Go-Browse`'s successes.

Table 4 shows URL patterns with the largest difference in success trajectory visits between `Go-Browse` and `NNetNav`. `Go-Browse`'s successes more frequently involve navigating to deeper URLs, such as editing specific product attributes or viewing particular order details. Notably,



(a) `NNetNav-WA` Task Distribution

(b) `Go-Browse-WA` Task Distribution

Figure 4: Task diversity of the Go-Browse and NNetNav datasets. Zoom to read sub-task labels.

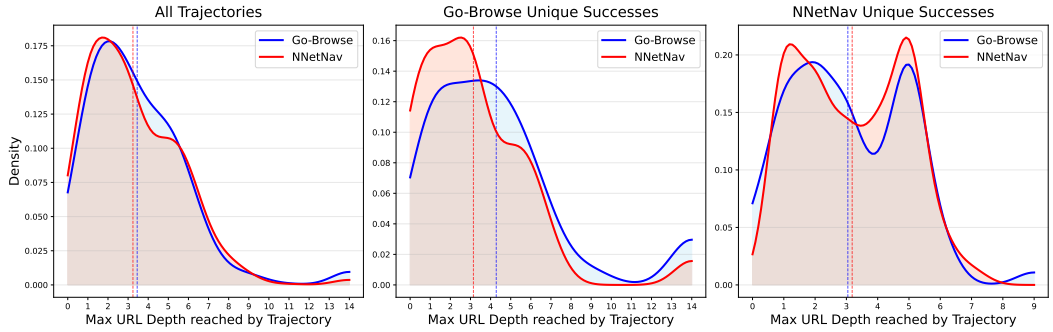


Figure 5: Distributions of maximum URL path lengths (depth) achieved by trajectories across all trajectories (left), trajectories where only Go-Browse was successful (middle), and trajectories where only NNetNav was successful (right). Go-Browse owes some of its wins to its tendency to go deeper. Note, depth is calculated as the number segments in the URL path.

Table 4: Top URLs by difference in visit count (`Go-Browse` (GB) vs. `NNetNav` (NN)).

More Visits By	URL	GB Visits	NN Visits	Diff.	Depth
GB	`<shopping_admin>/catalog/product/edit/id/{id}/`	10	1	9	5
	`<reddit>/search?query={query}?q=%7Bquery%7D`	7	0	7	2
	`<shopping_admin>/catalog/.../configurable/store/{id}/back/edit/`	5	0	5	12
	`<shopping_admin>/sales/order//view/order_id/{id}//{id}/`	6	2	4	6
	`<reddit>/user/{user}/edit_biography`	5	0	5	3
NN	`<gitlab>/projects/new`	2	6	4	2
	`<gitlab>/projects/new#blank_project`	2	5	3	2
	`<gitlab>/{user}/{repo}/-/commits/main`	2	5	3	5
	`<gitlab>/{user}/{repo}/-/forks/new`	1	4	3	5
	`<reddit>/forums/by_submissions/{id}`	0	3	3	3

`Go-Browse` exhibits significantly higher visit counts to these deeper URLs, including several that `NNetNav` never successfully visited. For instance, `Go-Browse` visited URLs for editing product attributes and searching Reddit 9 and 7 more times respectively, with `NNetNav` having 1 or 0 visits to these. Conversely, while `NNetNav` more frequently visits URLs related to creating new projects or forks in Gitlab, the difference in visitation counts is comparatively smaller. It is also interesting to observe their differing strategies for Reddit: `Go-Browse` tends to use the more direct search functionality, whereas `NNetNav` attempts to find forums by navigating to the `by_submissions` page. This aligns with our earlier finding that `Go-Browse` tends to succeed on tasks requiring deeper navigation.

Prefixed Sampling Bootstraps Weaker Models. Fig. 6 plots success rates of prefixed and unprefixed sampling against the depth of the node (URL) on which the tasks were sourced. Overall, prefixed sampling leads to higher success rates. The difference is especially pronounced as depth increases: it is harder to find deeper nodes again when starting from the root. The difference is also especially apparent for weaker models like `Qwen-2.5-7B-Instruct`. Prefixed sampling thus allows us to bootstrap from weaker pretrained models, enabling creation of higher quality data compared to what pretrained models can generate on their own.

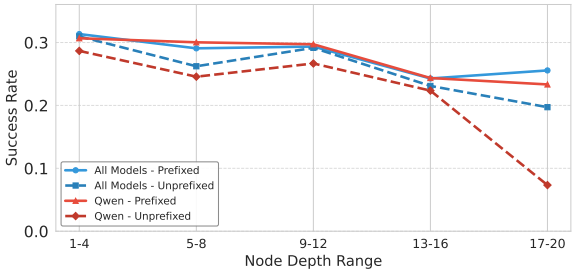


Figure 6: Prefixed sampling leads to higher success rates, especially on deeper nodes, particularly for weaker models like `Qwen-2.5-7B-Instruct`. Node depth is shortest trajectory length to reach a node from the root node, calculated using Dijkstra's algorithm.

Go-Browse's Outer Loop Improves Website Coverage. We analyze the effect of Go-Browse's outer loop by varying how many discovered URLs we reset to when proposing tasks, while fixing the total proposed tasks at 30 per domain. The $1/30$ case corresponds to Go-Browse without its outer loop (proposing 30 tasks for just a single URL per domain). As the number of reset nodes increases, the number of unique URLs visited grows steadily, demonstrating that Go-Browse's outer loop is critical for broader website coverage and more representative data.

(Resets / Tasks)	Unique URLs
(1 / 30)	183
(5 / 6)	214
(15 / 2)	260

Table 5: Unique URLs visited across 5 domains with varying # of resets.

FeasibilityChecker Improves Exploration Efficiency. During data collection, 403 tasks were filtered out by the FeasibilityChecker. This corresponds to a reduction of 3.2K trajectory rollouts ($\sim 29.4k$ steps). This is a 13% reduction in steps for the same amount of positive data.

7 Related Work

LLM Web Agents. We build on multiple works on LLM web agents like the WebArena benchmark [29] and BrowserGym [4] which both provide infrastructure for building and evaluating web agents and also provide an action space and observation features that we leverage in ReAct-based web agents [27]. There is also a line of works that augment the action space (e.g., with developer APIs or self-learned workflows) [18; 24; 28] or perform a form of in-context learning by extending the state representation with additional context gathered from past interactions [23; 9; 24]. Our work focuses instead on improving the base agentic capabilities of LLMs while keeping the scaffolding minimal.

Synthetic Data Generation for Web Agents. Past works on generating synthetic data for web agents have focused on either generating data from static indirect knowledge on the internet (e.g., tutorial articles) [14] or by logging direct interactions with websites [17; 8; 9; 10; 30]. Among the latter methods, interaction-first methods [10] seem to work unsupervised, while instruction-first methods [30; 17; 8] have typically required a human-in-the-loop to provide additional context. In our work, we build an unsupervised instruction-first method that can gather its own context via exploration. We also note more recent, concurrent instruction-first/hybrid methods for unsupervised collection of web agent data [15; 20; 22]. A key difference is Go-Browse's focus on deeply exploring websites by explicitly building and leveraging its own web graph. This helps Go-Browse collect high-quality and high-coverage data that enables training a state-of-the-art model for WebArena.

Exploration Methods in Reinforcement Learning. There is also a rich line of work from the RL community on improving exploration in agents [2; 16; 3; 6; 7]. Of these, our method takes the most inspiration from Go-Explore [6; 7] which uses analogous reset-then-explore strategies to share information across episodes to improve exploration efficiency in the context of Atari games.

8 Conclusion and Limitations

In this work, we propose Go-Browse, a fully unsupervised and scalable method for collecting web agent data through structured exploration of websites. We release, Go-Browse-WA, a dataset 9.5K successful and 39K unsuccessful task solving trajectories obtained while exploring the WebArena environments. We show that simple supervised finetuning of 7B parameter LLM on this dataset leads to significant improvements in success rates of web agents over the the previous state-of-the-art for sub-10B parameter models and also beats GPT-4o Mini. We thoroughly analyze the characteristics of our dataset and trained models, showing that Go-Browse-WA contains high-quality and diverse trajectories that lead to models that are able to better and more deeply navigate explored websites.

There are a number of limitations in our current experimental setup that open promising avenues for future research. Expanding data collection to a broader range of websites beyond the five WebArena domains would allow us to generate even larger datasets. While our current method achieves strong results using a 7B model trained on only successful trajectories, incorporating the signal from the 39K unsuccessful trajectories by exploring alternative training objectives (e.g., RL-based objectives) and scaling up model size may unlock even greater performance improvements. Finally, while using LLMs helps us scale training data, this risks introducing biases from models and prompts that may propagate to agent behavior, requiring careful auditing and mitigation before deployment.

Acknowledgements

We would like to thank Aviral Kumar, Vijay Viswanathan, Yueqi Song and Zora Zhiruo Wang for insightful discussions and feedback. We also thank the CMU Foundation and Language Model (FLAME) Center for access to their compute cluster. This work is supported in part by Amazon.

References

[1] Anthropic. Claude 3.7 Sonnet. `https://www.anthropic.com/claude/sonnet`, February 2025. API version released 24 Feb 2025, accessed 15 May 2025.

[2] Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[3] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *Seventh International Conference on Learning Representations*, pp. 1–17, 2019.

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

[5] Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., Del Verme, M., Marty, T., Vazquez, D., Chapados, N., and Lacoste, A. WorkArena: How capable are web agents at solving common knowledge work tasks? In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 21–27 Jul 2024.

[6] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[7] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847):580–586, 2021.

[8] Lai, H., Liu, X., Iong, I. L., Yao, S., Chen, Y., Shen, P., Yu, H., Zhang, H., Zhang, X., Dong, Y., et al. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024.

[9] Murty, S., Manning, C. D., Shaw, P., Joshi, M., and Lee, K. Bagel: bootstrapping agents by guiding exploration with language. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 36894–36910, 2024.

[10] Murty, S., Zhu, H., Bahdanau, D., and Manning, C. D. Nnetnav: Unsupervised learning of browser agents through environment interaction in the wild. In *Scaling Self-Improving Foundation Models without Human Supervision*, 2024.

[11] OpenAI. GPT-4o: omni-modal flagship model. `https://openai.com/index/hello-gpt-4o/`, May 2024. Model announced 13 May 2024, accessed 15 May 2025.

[12] OpenAI. GPT-4o mini: advancing cost-efficient intelligence. `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`, July 2024. Model announced 18 Jul 2024, accessed 15 May 2025.

[13] OpenAI. Computer-Using Agent: a universal interface for AI to interact with the digital world. `https://openai.com/index/computer-using-agent/`, January 2025. Research preview released 23 Jan 2025, accessed 15 May 2025.

[14] Ou, T., Xu, F. F., Madaan, A., Liu, J., Lo, R., Sridhar, A., Sengupta, S., Roth, D., Neubig, G., and Zhou, S. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[15] Pahuja, V., Lu, Y., Rosset, C., Gou, B., Mitra, A., Whitehead, S., Su, Y., and Awadallah, A. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. *arXiv preprint arXiv:2502.11357*, 2025.

[16] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

[17] Shen, J., Jain, A., Xiao, Z., Amlekar, I., Hadji, M., Podolny, A., and Talwalkar, A. Scribeagent: Towards specialized web agents using production-scale workflow data. *arXiv preprint arXiv:2411.15004*, 2024.

[18] Song, Y., Xu, F. F., Zhou, S., and Neubig, G. Beyond browsing: Api-based web agents. *CoRR*, 2024.

[19] Sun, J., Hua, Z., and Xia, Y. Autoeval: A practical framework for autonomous evaluation of mobile agents. *arXiv preprint arXiv:2503.02403*, 2025.

[20] Sun, Q., Cheng, K., Ding, Z., Jin, C., Wang, Y., Xu, F., Wu, Z., Jia, C., Chen, L., Liu, Z., et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*, 2024.

[21] Together AI. Together ai pricing. `https://www.together.ai/pricing`. Accessed: 2025-08-29.

[22] Trabucco, B., Sigurdsson, G., Piramuthu, R., and Salakhutdinov, R. Insta: Towards internet-scale training for agents. *arXiv preprint arXiv:2502.06776*, 2025.

[23] Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[24] Wang, Z. Z., Gandhi, A., Neubig, G., and Fried, D. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025.

[25] Xue, T., Qi, W., Shi, T., Song, C. H., Gou, B., Song, D., Sun, H., and Su, Y. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*, 2025.

[26] Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al. Qwen2 technical report. *CoRR*, 2024.

[27] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[28] Zheng, B., Fatemi, M. Y., Jin, X., Wang, Z. Z., Gandhi, A., Song, Y., Gu, Y., Srinivasa, J., Liu, G., Neubig, G., et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

[29] Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

[30] Zhou, Y., Yang, Q., Lin, K., Bai, M., Zhou, X., Wang, Y.-X., Levine, S., and Li, E. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. *arXiv preprint arXiv:2412.13194*, 2024.

A Web Agent Implementation Details

A.1 Web Agent Action Space

Table 6 shows the action space used for web agent experiments, adopted from the BrowserGym framework [5].

A.2 Prompts for LM Components

Action Type	Description
`noop(wait_ms)`	Do nothing for specified time.
`click(elem)`	Click at an element.
`hover(elem)`	Hover on an element.
`fill(elem, value)`	Type into an element.
`keyboard_press(key_comb)`	Press a key combination.
`scroll(x, y)`	Scroll horizontally or vertically.
`select_option(elem, options)`	Select one or multiple options.
`goto(url)`	Navigate to a url.
`go_back()`	Navigate to the previous page.
`go_forward()`	Navigate to the next page.
`new_tab()`	Open a new tab.
`tab_close()`	Close the current tab.
`tab_focus(index)`	Bring tab to front.
`send_msg_to_user(text)`	Send a message to the user.
`report_infeasible(reason)`	Notify user that instructions are infeasible.

Table 6: Web Agent action space.

Prompt Template for Web Agents

Instructions
You are a UI Assistant, your goal is to help the user perform tasks using a web browser. Review the instructions from the user, the current state of the page and all other information to find the best possible next action to accomplish your goal. Your answer will be interpreted and executed by a program, make sure to follow the formatting instructions.

Goal **{Goal}**

#Action Space
{Action space description from Table 6}

Here are examples of actions with chain-of-thought reasoning:
{"thought": "I now need to click on the Submit button to send the form. I will use the click action on the button, which has bid 12.", "action": "click('12')"}
{"thought": "I found the information requested by the user, I will send it to the chat.", "action": "send_msg_to_user('The price for a 15 inch laptop is 1499 USD.')"}
{"thought": "I have finished navigating to the Products page. I will inform the user that I have completed the task.", "action": "send_msg_to_user('I have finished navigating to the Products page.')"}

Current Accessibility Tree
{Axtree Text}

Error Message from Last Action
{Last Action Error}

History of Past Actions
{Past Actions}

Next Action
You will now think step by step and produce your next best action. Reflect on your past actions, any resulting error message, the current state of the page before deciding on your next action. Provide your output as a single json with a thought and an action. All reasoning must be contained within the thought key of the json output, and only a single action must be provided for the action key. Future actions will be taken subsequently. If you have finished performing the request, send a message to the user in a concise and to the point manner.

Goal for NavExplorer

I am trying to collect a dataset to train a better web browser agent that can perform actions for users in a web browser. For this, we are particularly interested to collect **navigation tasks** that are feasible to perform from the current web page.
Navigation tasks are tasks requiring navigating to a specific page.

Collect navigation tasks that require navigating to another webpage from this current page. You may click on links to try finding other interesting pages to collect tasks from. But if you do navigate to another page, instead of collecting tasks on that page, make sure to navigate back to the previous page using ‘go_back‘ or ‘goto‘. We will collect tasks from these new pages later. When collecting navigation tasks, prioritize those that would likely have interesting/useful tasks on them over ones that likely won’t give many useful tasks to collect.

As you are exploring, you can add navigation tasks to the dataset using the ‘add_tasks_to_dataset‘ function.

When you are done exploring the current page, send a message to the user using ‘send_msg_to_user‘ confirming this.

Be sure to prioritize adding navigation tasks to pages that a typical user of this web page would most often want to navigate to, over niche pages that the typical user would rarely frequent.

Important Remember that if you are successful at navigating to a new page, you should add a corresponding task to the dataset as your next action before finding new pages.

Goal for PageExplorer

I am trying to collect a dataset to train a better web browser agent that can perform actions for users in a web browser. For this, I need to first collect tasks that are feasible to perform on the current web page. The tasks should be concrete (e.g., on an amazon product page for product X, an appropriate task could be "Leave a positive review for X" or on a maps website a task could be "Show me driving directions from X to Y." where X and Y are specific locations).
You may explore by performing actions on this web page if that helps to determine concrete tasks that are feasible.

Find the tasks that are possible to perform on the current web page itself, without have to navigate to other links/urls. Though, you may find it helpful to navigate through menus on this page to get a better idea of what types of tasks are feasible. If you accidentally go to a new url while trying to navigate items on the page, you can go back to the previous page using the ‘go_back‘ function.

Tasks are usually of three types:
1. Information seeking: The user wants to obtain certain information from the webpage, such as the information of a product, reviews, map info, comparison of map routes, etc.
2. Site navigation: The user wants to navigate to a specific page.
3. Content modification: The user wants to modify the content of a webpage or configuration.

Be as specific as you can while creating tasks. The web agent may start from a different web page when asked to complete the task and so may not have the current page context to understand the task. So, for example, avoid creating generic tasks like "Add item to cart" or "Print receipt for this order." Instead you want to create specific tasks like "Add a Sony PS5 to cart" or "Print a receipt for Martha Jone’s order of the Nike Velocity Sweatpants from May 21, 2021"

I recommend the following order to collecting tasks:
1. First look for information seeking/extraction tasks that can be answered simply using information on the current page, requiring no additional actions.
2. Collect navigation tasks that require navigating to another webpage from this current page. You may click to links to try finding other interesting pages to collect tasks from. But if you do navigate to another page, instead of collecting tasks on that page, make sure to navigate back to the previous page using ‘go_back‘. We will collect tasks from these new pages later. When collecting navigation tasks, prioritize those that would likely have interesting/useful tasks on them over ones that likely won’t give many useful tasks to collect.

3. Finally, you can try to find content modification tasks on the current page that require performing actions on the current page itself.

As you are exploring the page, you may find it helpful to click on buttons, links, and other elements on the page to see if they reveal any additional information or options that could lead to new tasks. You can also hover over elements to see if they provide any tooltips or additional context.

Important:
When collecting tasks, focus more on the common tasks that a typical user of this webpage would want to perform. Avoid niche tasks that are unlikely to be relevant to the typical user of this website.
For most common styles of tasks, it may be useful to include a few variants or related tasks to help the web agent learn frequently used skills.

As you are exploring, you can add tasks to the dataset using the ‘add_tasks_to_dataset‘ function.

When you are done exploring, send a message to the user using ‘send_msg_to_user‘ confirming this.

Prompt for VLM-as-a-judge Reward Model

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, the final state of the webpage, and the agent's response to the user, your goal is to decide whether the agent's execution is successful or not. Please be careful of each detail and strict about the evaluation process.

There are three types of tasks:

1. Information seeking: The user wants to obtain certain information from the webpage, such as the 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 respond 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.

2. 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.

3. 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.

User Intent: **{Goal}**

Action History:
{Last Actions}

The final state of the webpage provided as an accessibility tree:
{Axtree Text}

The last snapshot of the web page is shown in the image.
{Screenshot}

B Additional Analyses

B.1 Design Choices for Task Proposal

B.1.1 GPT-4o vs. Claude-3.7-Sonnet for PageExplorer Task Proposal

To understand how task proposal behavior differs based on model choice, we tag proposed PageExplorer tasks using an LLM as navigational (Nav), information-seeking (Info), or state/content-modifying (Mod) tasks (the same three categories mentioned in the PageExplorer goal). We also perform clustering of these tasks to measure diversity, similar to Section 6.

The models differ in task proposal behavior as shown in Table 7: (1) Claude-3.7-Sonnet proposes almost almost double the number of tasks with half the max step budget; (2) GPT-4o generates a more diverse set of tasks for the quantity proposed, especially Mod tasks, where GPT-4o has many more task clusters.

The efficiency of Claude allows us to give it a smaller max step budget when used as a PageExplorer. On the other hand, GPT-4o's diversity of Mod tasks justifies using it as well to complement Claude.

Table 7: Comparison of GPT-4o and Claude-3.7-Sonnet for PageExplorer agents.

Model	# Tasks			# Clusters			Max # Steps (Per Node)
	Nav	Info	Mod	Nav	Info	Mod	
GPT-4o	274	227	243	24	18	34	20
Claude-3.7-Sonnet	415	508	516	23	19	19	10

B.1.2 NavExplorer vs. PageExplorer Tasks

Since navigational tasks are important for website coverage and are linked to Go-Browse's outer loop, we also explicitly add a NavExplorer agent with Claude (chosen for its efficiency) in addition to the PageExplorer agents. This more than doubles the number of navigational tasks in the dataset.

Table 8: Comparison of Explorer types on navigation tasks.

Explorer Type	# Nav Tasks	# Nav Task Clusters
NavExplorer	925	32
PageExplorer	689	31

B.2 Dataset Collection Cost Analysis

Table 9 provides the cost per model during rollouts (both data collection and task proposal - Panel A) and also the cost of trajectory evaluation using GPT-4o (Panel B). The overall cost of collecting Go-Browse-WA is $\$975.57$. We note that for trajectory rollouts, the cost of Claude-3.7-Sonnet, GPT-4o and GPT-4o-mini is significantly reduced due to lower prices for cached tokens. We observe that ($\sim 53\%$ of input tokens are cache reads on average). We use the official OpenAI and Anthropic API pricing to compute their costs and use Together AI [21] to estimate API cost for Qwen-2.5-7B-Instruct.

B.3 Paired Bootstrap Test for WebArena Results

We measure statistical significance using the paired bootstrap test with 10,000 bootstrap samples of the WebArena benchmark results. Our model is statistically significantly better than Qwen-2.5-7B-Instruct ($p < 0.001$). It was also judged as better than GPT-4o-mini ($p = 0.108$) and NNetNav-7B ($p = 0.094$). These results demonstrate a moderate degree of confidence in our model's improvement over these baselines, with high win ratios for Go-Browse-7B.

Table 9: Go-Browse cost analysis for rollouts (agents) and trajectory evaluation.

Panel A: Rollout Costs (Agents)				
Model	**Num. Trajs**	**Num. Steps**	**Avg. Cost / Step**	**Total Cost**
GPT-4o-mini	11,695	95,314	0.0008	76.25
Qwen-2.5-7B Instruct	10,203	79,209	0.0025	198.02
Claude-3.7-Sonnet	5,102	24,532	0.0190	466.11
GPT-4o	103	789	0.0181	14.28
Total (Rollouts)	**27,103**	**199,844**	**0.0037**	**754.66**

Panel B: Trajectory Evaluation Costs			
Model	**Num. Trajs**	**Avg. Cost / Traj Eval**	**Total Cost**
`GPT-4o`	11,105	0.0199	220.91
Grand Total (Rollouts + Eval)			**$975.57**

Table 10: Paired bootstrap tests of models vs. `Go-Browse-7B` on WebArena with 10K samples.

Model Compared	**Winner**	**Win Ratio (Baseline / Tie / `Go-Browse-7B`)**	**p-value**
`Claude-3.7-Sonnet`	`Claude-3.7-Sonnet`	(1.000 / 0.000 / 0.000)	0.000
`GPT-4o`	`GPT-4o`	(1.000 / 0.000 / 0.000)	0.000
`GPT-4o-mini`	`Go-Browse-7B`	(0.085 / 0.024 / 0.892)	0.108
`NNetNav-7B`	`Go-Browse-7B`	(0.076 / 0.018 / 0.906)	0.094
`Qwen-2.5-7B-Instruct`	`Go-Browse-7B`	(0.000 / 0.000 / 1.000)	0.000

C Hyperparameters and Additional Experiment Details

For our finetuning experiments, we use the following hyperparameters. We train for 2 epochs on the whole dataset with a maximum sequence length of 24K tokens. We use a learning rate of 2e-5. We use a batch size of 8 (1 per gpu) with 4 gradient accumulation steps.

We used the following computational resources. For finetuning with a single NVIDIA 8xH100 node where each H100 has 80GB of VRAM. Training took $\sim$40 hours for each finetuning run. For dataset generation, we run on 5 nodes with of a SLURM cluster in parallel, with 256GB of RAM and 8 CPUs allocated to each, one each per WebArena domain. We also ran LLM inference servers on 8 NVIDIA L40S GPUs to support inferencing with `Qwen-2.5-7B-Instruct`. Overall, dataset generation took $\sim$3 weeks to complete.

In this work, besides generating our own `Go-Browse-WA` dataset, we leverage the `NNetNav-WA` dataset to build a baseline. This dataset was released with the Apache License 2.0 license.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All results and claims in Abstract are justified through experiments and analysis in the main paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations and further opportunities are discussed Section 8.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: The paper does not include any theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes, dataset collection hyperparameters are provided in Section 4 and fine-tuning hyperparameters are provided in Section 5 and Appendix C. The paper details the dataset creation process in detail. Of course, code, data and models are also open access.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Yes, we provide a repo link which has the code, dataset, and instructions on reproducing the results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, all needed hyperparameters to reproduce and understand the results are provided in Section 4, Section 5 and Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We perform paired bootstrap tests to measure statistical significance of our evaluation on WebArena. These are reported in Appendix B.3.

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, these details are provided in Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, the conducted research conforms in every respect with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes, the Introduction 1 and Conclusion & Limitations 8 discuss broader impacts.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The data we collect and models we train are limited to simple web agent tasks such as those from the WebArena benchmark, and so do not have high risk for misuse with their current capabilities. The dataset is collected on self-hosted, simulated websites instead of actual live websites.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, all assets used (datasets, models, web agent infrastructure) are cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes, the main new assets in this paper are the generated dataset and model checkpoints. These as well as code/documentation are linked via an anonymized repo URL.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve research with human subjects or crowdsourcing.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve research with human subjects or crowdsourcing.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Yes, our method uses LLMs as an integral part of the dataset creation process and as the backbone for our web agents. LLM usage is described in detail throughout the paper when discussing algorithm design.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.