

# NNETSCAPE NAVIGATOR: COMPLEX DEMONSTRATIONS FOR WEB AGENTS WITHOUT A DEMONSTRATOR

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

We introduce NNetscape Navigator (NNetnav), a method for training web agents entirely through synthetic demonstrations. These demonstrations are collected by first interacting with a browser to generate trajectory rollouts, which are then retroactively labeled into instructions using a language model. Most work on training browser agents has relied on expensive human supervision, and the limited previous work on such *interaction-first* synthetic data techniques has failed to provide effective search through the exponential space of exploration. In contrast, NNetnav exploits the hierarchical structure of language instructions to make this search more tractable: complex instructions are typically decomposable into simpler subtasks, allowing NNetnav to automatically prune interaction episodes when an intermediate trajectory cannot be annotated with a meaningful sub-task. We use NNetnav demonstrations from a language model for supervised fine-tuning of a smaller language model policy, and find improvements of 6 points on WebArena and over 20 points on MiniWoB++, two popular environments for web-agents. Notably, on WebArena, we observe that language model policies can be further enhanced when fine-tuned with NNetnav demonstrations derived from the *same* language model. Finally, we collect and release a dataset of over 6k NNetnav demonstrations on WebArena, spanning a diverse and complex set of instructions.

## 1 INTRODUCTION

Building grounded language agents that map human language instructions to a sequence of executable actions is a long-standing goal of artificial intelligence (Winograd, 1972), with the ultimate goal of automating mundane web tasks like flight booking. A promising new approach for building such agents is to use large language models to control policies in digital environments like browsers (Yao et al., 2022; Shinn et al., 2023; Murty et al., 2024; Wang et al., 2024, among others).

Unfortunately, such *grounded instruction following* without any training examples is challenging because LMs do not know about the myriad and ever changing interaction possibilities of different websites. For instance, for a new online shopping website, a zero-shot LM agent may not know how to make a return or change order details, without expensive test-time exploration. Even simple tasks like selecting a flight might involve typing in airport codes or selecting from a drop-down menu, and zero-shot agents cannot know this a-priori.

One way to provide LM web-agents with knowledge about new web interfaces is via expert demonstrations, that can either be used for in-context learning (Yao et al., 2022) or supervised fine-tuning (Lai et al., 2024). These demonstrations are either fully provided by human experts (Sodhi et al., 2023; Yao et al., 2022) or consist of human-generated trajectories paired with model-generated instructions (Lai et al., 2024). Of course, collecting human demonstrations that cover each possible use case for every web-site is an unattractively large, never-ending task. Thus, recent work uses entirely synthetic demonstrations by sampling a synthetic instruction, and then mapping it into a trajectory using a base LLM agent (Patel et al., 2024; Murty et al., 2024).

Such *instruction-first* methods for data collection face several challenges. First, synthetic instructions in these demonstrations are sampled from an ungrounded LM prior that generates only plausible<sup>1</sup>

<sup>1</sup>We use the term *plausible* for instructions that match a website’s genre or intended use. For example, searching for clothes on a retail site or checking notifications on a social media platform. Not all plausible instructions are feasible.

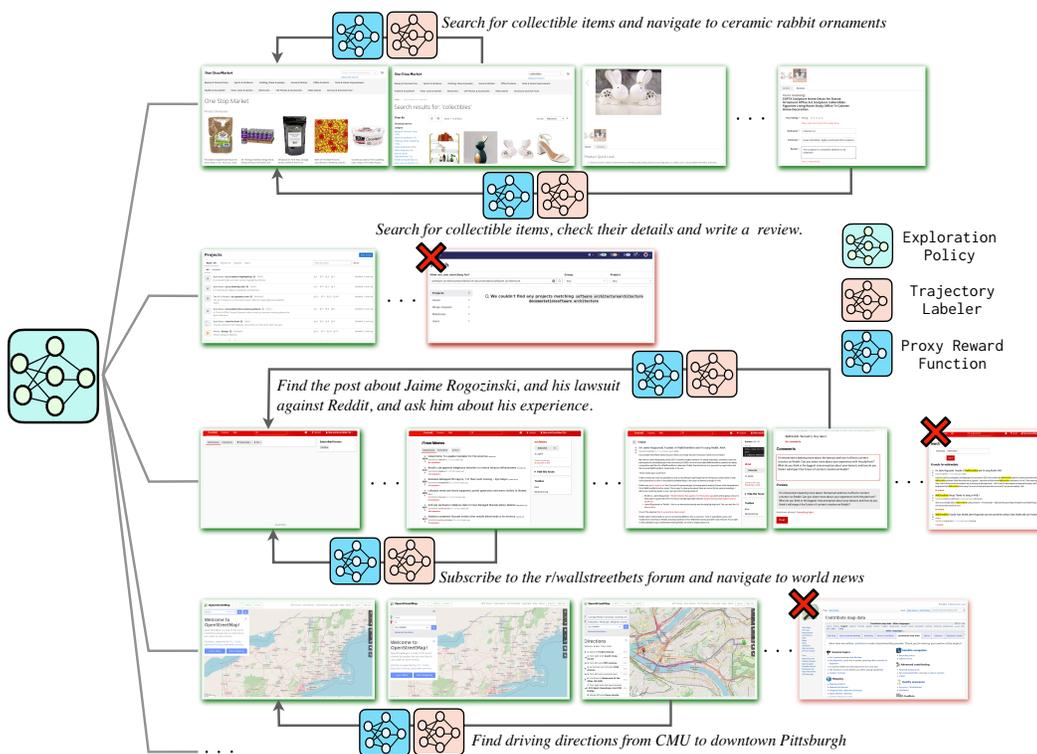


Figure 1: NNetnav produces synthetic demonstrations for training web-agents by exploring a website to create trajectories, and then labeling them into instructions. Long exploration in NNetnav is made efficient using a pruning heuristic inspired by the hierarchical structure of complex instructions. At fixed intervals during exploration, the labeling function infers an instruction for the trajectory so far, and if the resulting (instruction, trajectory) pair receives a low score from a reward function, the episode is terminated (red cross). Components in NNetnav are implemented using prompts to the same LM.

instructions without ensuring feasibility; *e.g.*, an instruction such as *Respond to the first post on r/callofdutyfans* for reddit is plausible, but not always feasible. Second, generated instructions are limited to those that reference visible features of the website; *e.g.*, given the landing page of a github-like platform, no LM prior can generate instructions like *Find information about Eric Bailey’s contributions to the byteblaze project*, which requires knowing about deeply embedded website-specific entities like *Eric Bailey*. Finally, these methods provide no control over the complexity of instructions, and rely entirely on the LM or bespoke prompts to generate complex instructions.

Instead of starting with a sampled instruction, we start by sampling an *interaction* first, and then retroactively labeling it into an instruction that is feasible by design. At a high-level, our approach *NNetscape Navigator* (NNetnav, Fig 1), uses a language model exploration policy to perform extended interactions with an environment, and another language model trajectory labeler to annotate trajectories with instructions<sup>2</sup>. To effectively control the exponential space of meaningful interactions, NNetnav uses the hierarchical structure of language instructions as a pruning heuristic: for exploration to discover a meaningfully complex task, trajectory prefixes must correspond to meaningful sub-tasks. Thus, during an exploration episode, if a language model cannot label trajectory prefixes (at set time-steps) with a sub-task, further exploration is automatically terminated. Imposing such a structure over search not only enhances efficiency, but also results in complex and hierarchical instructions (See Table 7 for examples). NNetnav prompts the same base language model for exploration, relabeling and inferring sub-tasks, and effectively addresses all limitations of instruction-first data collection.

<sup>2</sup>We will open-source our code upon acceptance.

Using GPT-4o-mini (Achiam et al., 2023) as our base language model, we use demonstrations collected via NNetnav to fine-tune a smaller Llama-3-8B-Instruct (Dubey et al., 2024) based agent on two benchmarks for web navigation, MiniWoB++ (Shi et al., 2017; Liu et al., 2018) and WebArena (Zhou et al., 2023). Compared to the base agent, performance of the fine-tuned agent improves from 28% to 48% on MiniWoB++, and from 1% to 7% on WebArena. Crucially, these improvements are higher than those from a model that’s fine-tuned with an instruction-first data collection method. Finally, we find that NNetnav can be used for self-training—fine-tuning a small LM agent with NNetnav demonstrations from the *same* LM leads to an improvement of 4% points absolute (1% to 5%) on WebArena. Further analysis reveals the benefits of retroactive labeling beyond performance improvements: When using a model-based evaluator, similar to Pan et al. (2024), hindsight trajectories from NNetnav have a higher mean reward than trajectories from an LLM agent based on the same underlying language model. Finally, we collect and release NNetnav-6k, a dataset of over 6k demonstrations covering a wide and complex range of use cases on WebArena.

## 2 BACKGROUND

Following instructions on web-browsers is a deterministic sequential decision making problem. Given an instruction  $g$ , an instruction following agent interacts with the browser by issuing a sequence of actions  $\langle a_1, a_2, \dots, a_T \rangle$  where each  $a_i \in \mathcal{A}$  is drawn in response to an observation  $o_i$ . Executing an action causes a state transition based on some unknown but deterministic environment dynamics, leading to a new observation  $o_{i+1}$ . The entire episode can be summarized as a trajectory  $\tau := \langle o_1, a_1, o_2, a_2, \dots, o_T, a_T, o_{T+1} \rangle$ . We formalize the instruction following agent as a mapping  $\pi(a_t | \tau_{<t}; g)$  where  $\tau_{<t} := \langle o_1, a_1, \dots, o_t \rangle$  is the trajectory so far. In this framework, the action space  $\mathcal{A}$  consists of a finite set of strings, while observations are represented as either flattened DOM trees or website accessibility trees.

**LLMs as Instruction Following Agents.** Recent work explores prompted large language models (LLMs) to directly parameterize  $\pi$ . These methods typically work in settings with textual observations and action spaces, and many output a reasoning string  $r_i$  before predicting the action string  $a_i$ . Concretely, we formalize an LM agent (omitting prompts) as  $\pi_{\text{LM}}(a_t | \tau_{<t}; g) := p_{\text{LM}}(a_t, | \tau_{<t}, r_t; g)$  where  $r_t \sim p_{\text{LM}}(r | \tau_{<t}; g)$  is a reasoning step drawn as a sample from the LM.

Given expert demonstrations  $\{g^i, \tau^i\}$  where  $\tau^i := \langle o_1^i, r_1^i, a_1^i, o_2^i, r_2^i, a_2^i \dots o_T^i \rangle$ , prior work adapts LM agents using demonstrations either as in-context examples (Yao et al., 2022; Shinn et al., 2023; Sun et al., 2023; Kim et al., 2023, among others) or as training data for supervised fine-tuning (Furuta et al., 2023; Lai et al., 2024; Lù et al., 2024; Patel et al., 2024). For supervised fine-tuning of  $\pi_{\text{LM}}$  on a dataset of demonstrations, we construct training instances  $\{(g^i, \tau_{<t}^i), (r_t^i, a_t^i)\}$  where  $r_t^i, a_t^i$  serves as the target reasoning step and action for an intermediate context  $(g^i, \tau_{<t}^i)$ .

**Data collection with instruction-first methods.** Collecting human demonstrations for training web-agents is time consuming and costly. Thus, recent work proposes methods for generating synthetic data for web-agents using language model components (Lai et al., 2024; Furuta et al., 2023; Murty et al., 2024). These methods start by sampling synthetic instructions from an instruction generator (a prompted LM that takes the website landing page and a persona as input), and then use a zero-shot LM policy to convert these instructions into trajectories. Resulting demonstrations are filtered using either the ground truth reward function (Furuta et al., 2023), or using another language model based reward function (Lai et al., 2024; Murty et al., 2024). Most of these methods use bigger and better language models for collecting demonstrations, and then use this data to adapt smaller models.

## 3 OUR APPROACH

NNetnav (Fig 1) is an *interaction-first* method for constructing demonstrations: An exploration policy interacts with a browser in a structured manner to sample long trajectories which are retroactively labeled into instructions (§3.2). We then post-process each trajectory to add post-hoc reasoning steps corresponding to the generated instructions, and then use this data for supervised fine-tuning (§3.3). We provide detailed pseudo-code for the exploration and relabeling steps in NNetnav in Algorithm 1.

**Algorithm 1:** Exploration and Relabeling in NNetnav within a single interaction episode

---

```

162 Input:  $\pi_{\text{explore}}, \text{Lf}_{\text{LM}}, s_{\text{LM}}, \Delta_{\text{LM}}$ 
163
164 Function run_retroactive_labeler( $\tau$ ):
165    $\delta_\tau \leftarrow [\Delta_{\text{LM}}(o_t, a_t, o_{t+1}) \mid (o_t, a_t, o_{t+1}) \in \tau]$ ;
166    $g_{\text{curr}} \leftarrow \text{Lf}_{\text{LM}}(\delta_\tau)$ ;
167    $r_{\text{curr}} \leftarrow s_{\text{LM}}(i_{\text{curr}}, \delta_\tau)$ ;
168   return  $g_{\text{curr}}, r_{\text{curr}}$ ;
169
170 Function explore( $\mathcal{W}, T_{\text{prune}}$ ):
171    $o_1 \leftarrow \mathcal{W}.\text{init-observation}$ ;
172    $t \leftarrow 1$ ;
173    $\tau \leftarrow \langle \rangle$ ;
174   demonstrations  $\leftarrow []$ ;
175   while  $t \leq T_{\text{max}}$  do
176     if  $t \in T_{\text{prune}}$  then
177        $g_{\text{curr}}, r_{\text{curr}} \leftarrow \text{run\_retroactive\_labeler}(\tau)$ ;
178       if  $r_{\text{curr}} < 1$  then
179         break;
180       else
181         demonstrations.add( $(g_{\text{curr}}, r_{\text{curr}})$ );
182      $a_t \leftarrow \pi_{\text{explore}}(o_t)$ ;
183      $o_{t+1} \leftarrow \mathcal{W}.\text{step}(a_t)$ ;
184      $\tau.\text{add}((o_t, a_t, o_{t+1}))$ ;
185      $t \leftarrow t + 1$ ;
186   return demonstrations;

```

---

### 3.1 LM COMPONENTS

We start by describing various components in NNetnav. All of these components are implemented by zero-shot prompting a language model, with different prompts (see Appendix A for details).

**Exploration Policy.** To interact with the environment, we use an exploration policy  $\pi_{\text{explore}}$ , implemented using a chain-of-thought prompted language model (Wei et al., 2022). Additionally, to simulate a diverse set of behaviors from users and improve the diversity of resulting trajectories, we seed each episode with a string description of a plausible user persona for the given website (Shanahan et al., 2023; Argyle et al., 2023).

**Summarizing Trajectory changes.** Actions issued by  $\pi_{\text{explore}}$  result in a new observation in the environment. We summarize this change as a short string description via another module  $\Delta_{\text{LM}}$ , implemented using a language model. In particular, for any state  $o_t$ , action  $a_t$  and the resulting next state  $o_{t+1}$ ,  $\delta_t = \Delta_{\text{LM}}(o_t, a_t, o_{t+1})$  produces a string-valued description of the changes in the observation as a result of the action. For a trajectory  $\tau$ , we denote the sequence of state changes as  $\delta_\tau$

**Trajectory Labeling Function.** Given state changes  $\delta_\tau$ , a labeling function  $\text{Lf}_{\text{LM}}$  produces a plausible instruction  $\hat{g} = \text{Lf}_{\text{LM}}(\delta_\tau)$  that the agent could have followed to produce the given interaction.

**Reward Function.** Given  $\hat{g}$  and  $\delta_\tau$ , the reward function module assigns a reward  $s_{\text{LM}}(\hat{g}, \delta_\tau) \in \{0, 1\}$ , based on how well the state changes correspond to the given instruction  $\hat{g}$ .

### 3.2 SAMPLING DEMONSTRATIONS VIA INTERACTIONS

Let  $t_{\text{max}}$  be the maximum episode length for each exploration rollout. At specific time-steps  $\{t_1, t_2, \dots, t_{\text{max}}\}$ , we run the pruning heuristic that attempts to annotate the trajectory so-far with a sub-task annotation. If this is successful, we continue the episode, and otherwise halt to sample another rollout. Concretely, suppose we have a partial trajectory  $\tau_{<t}$  after interacting with the environment for  $t$  time-steps. If  $t \in \{t_1, t_2, \dots, t_{\text{max}}\}$ , we first obtain a retroactive sub-task  $\hat{g} = \text{Lf}_{\text{LM}}(\delta_{\tau_{<t}})$ . We halt further exploration if  $s_{\text{LM}}(\hat{g}, \delta_{\tau_{<t}}) = 0$ . Otherwise, we add the generated  $(\hat{g}, \tau_{<t})$  to our set

of synthetic demonstrations, and continue exploring. Typically, each interaction episode results in multiple demonstrations.

### 3.3 GENERATING POST-HOC REASONING STEPS

The exploration policy in this work is implemented using a language model that generates a reasoning step, before choosing an action (§2). Since actions in our demonstration set are a result of exploration, corresponding reasoning steps are not generally related to the retroactively generated instruction. Thus, for each demonstration in our synthetic demonstration set, we post-hoc annotate every action with a new reasoning step that directly corresponds to the generated instruction. Concretely, given every  $(\hat{g}, o_i, a_i)$  tuple in our synthetic demonstration set, we prompt a language model to output a suitable reasoning step for choosing action  $a_i$  given the instruction  $\hat{g}$  and current observation  $o_i$ . We note that such a post-hoc reasoning procedure is similar to Yang et al. (2024).

## 4 EXPERIMENTAL SETUP

### 4.1 DATASETS

We fine-tune language model policies with NNetnav demonstrations on two web navigation environments, MiniWoB++ (Shi et al., 2017; Liu et al., 2018) and WebArena (Zhou et al., 2023).

1. **MiniWoB++:** A dataset of diverse synthetic web-interfaces with a shared action space. Tasks on MiniWoB++ range from clicking on buttons to complex tasks like making a booking on a website. We use a subset of 8 complex tasks from MiniWoB++ as a toy benchmark to evaluate our method. We use the `bid`-based action space from BrowserGym (Drouin et al., 2024), consisting of 12 actions, and a DOM based observation space. Due to its synthetic nature, MiniWoB++ comes with an automatic reward function. We report the mean reward over 20 random seeds for each task, similar to (Drouin et al., 2024).
2. **WebArena:** A dataset of realistic web navigation tasks over 5 websites covering domains such as e-commerce, discussion forums, maps and software development. We use the default action space from WebArena (typing, clicking, hovering, tab management) and the default accessibility tree based observation space. WebArena consists of 812 Web navigation tasks across these websites, and provides an evaluator that measures success rate (SR) in terms of functional correctness. We report the average SR across these tasks.

### 4.2 MODEL SETTINGS

All inference evaluations are conducted using the same base language model, with data collection typically performed using a larger language model unless stated otherwise. We evaluate under the following settings:

1. **Zero-Shot:** A baseline zero-shot LM policy  $\pi_{LM}$ , prompted using chain-of-thought prompting (Wei et al., 2022). Next, we consider various fine-tuned models.
2. **SFT (Instruction-First):** Supervised fine-tuning (SFT) of the base policy using data collected via instruction-first sampling. Here, we use the same reward model for filtering demonstrations as NNetnav, and also sample the same number of demonstrations for fair comparison.
3. **SFT (NNetnav):** Supervised fine-tuning of  $\pi_{LM}$  with demonstrations collected via NNetnav.
4. **SFT (NNetnav + Distil):** Ablation, where we only retain instructions found via NNetnav and re-generate trajectories by prompting the same large LM as an agent. We use this setting to isolate performance improvements attributable to NNetnav trajectories.

### 4.3 IMPLEMENTATION DETAILS

All LM components for data collection in NNetnav as well as instruction-first methods are based on GPT-4o-mini (specifically `gpt-4o-mini-2024-07-18`). We use Llama-3-8B-Instruct as the inference policy  $\pi_{LM}$ . For Instruction-first data collection, we sample 50 instructions per

Domain	Zero-Shot	SFT (Instruction-First)	SFT (NNetnav)	SFT (NNetnav + Distil.)
<i>MiniWoB++</i>				
book-flight	0.0	0.0	0.0	0.0
choose-date	0.0	0.0	0.0	0.0
click-checkboxes-soft	0.4	0.25	<b>0.65</b>	0.5
email-inbox	0.25	0.3	0.3	<b>0.35</b>
login-user	0.3	0.0	<b>1.0</b>	0.95
navigate-tree	1.0	0.95	<b>1.0</b>	0.95
phone-book	0.15	0.15	0.2	<b>0.55</b>
use-autocomplete	0.25	0.55	<b>0.7</b>	0.35
Avg.	0.28	0.28	<b>0.48</b>	0.36
<i>WebArena</i>				
Shopping	3.8	<b>7.7</b>	7.4	7.4
CMS	0	4.2	4.2	4.2
Reddit	0	0	0	0
Gitlab	0	0	0	4.5
Maps	0	9.1	<b>28.5</b>	15.4
Avg.	1.0	4.2	<b>7.2</b>	6.0

Table 1: Results for MiniWoB++ and WebArena, broken down by domain, reporting mean reward for MiniWoB++ and task success rate (SR) for WebArena. We compare the zero-shot agent with agents fine-tuned with NNetnav and instruction-first demonstrations. Overall, fine-tuning with NNetnav leads to the largest improvements: from 28% to 48% on MiniWoB++; from 1% to 7.2% on WebArena.

website for WebArena, and 80 instructions per interface in MiniWoB++, and prompt the instruction generator with the landing page as well as a persona (to improve diversity). For NNetnav, we use our exploration policy to generate 50 episodes per website for WebArena, and 80 episodes per interface for MiniWoB++. We set  $T_{\max}$  to 40 for WebArena, and 20 for MiniWoB++. For both MiniWoB++ and WebArena, we apply the pruning function every 4 time-steps. We use 16 persona types per website for WebArena, and 10 persona types per web-interface for MiniWoB++.

We use the BrowserGym framework (Drouin et al., 2024) for experiments with MiniWoB++ and prune out the full DOM to only keep visible elements. During inference, we set the max episode length for  $\pi_{LM}$  as 30 for WebArena (following Zhou et al. (2023)), and 20 for MiniWoB++. WebArena requires agents to output a special `stop` action for outputting answers. We post-process NNetnav demonstrations to add a `stop` action at the end of the trajectory using a prompted LM (See Appendix A.2 for details).

**Fine-tuning details.** We fine-tune all models for 5 epochs, truncating the max sequence length to 4096, with a learning rate of  $2e-5$ , using 4 A100 GPUs. We provide complete details of our training setup in Appendix D. We use open-instruct (Wang et al., 2023) for fine-tuning all language models, and set up local inference servers using vllm (Kwon et al., 2023).

## 5 MAIN RESULTS

**Fine-tuning agents with NNetnav leads to consistent gains.** We report results from all model settings in Table 1. We find that fine-tuning the zero-shot policy  $\pi_{LM}$  with synthetic demonstrations from NNetnav leads to consistent improvements on all tasks in MiniWoB++, leading to a 20 point improvement overall. We note an improvement of over 6 points from fine-tuning with NNetnav demonstrations on WebArena. Importantly, gains from fine-tuning with NNetnav exceeds those from using instruction-first methods by 12 points on MiniWoB++ and 1.2 points on WebArena.

Domain	Zero-Shot	SFT (Instruction-First)	SFT (NNetnav)	SFT (NNetnav + Distil.)
Shopping	1.26	1.37	2.22	<b>2.33</b>
CMS	1.21	1.29	<b>1.92</b>	1.87
Reddit	1.08	1.31	<b>2.0</b>	1.54
Gitlab	1.14	1.09	<b>1.86</b>	1.5
Maps	1.21	1.36	<b>2.29</b>	1.86
WebArena (Avg.)	1.19	1.28	<b>2.05</b>	1.87

Table 2: Model-based evaluation on WebArena, broken down by domain. For each test instruction and predicted trajectory, we prompt a GPT-4o based reward model to output a graded reward from 1 to 5 based on a manual rubric. We find that fine-tuning with NNetnav outperforms all other settings.

Domain	Zero-Shot	Self-Train (NNetnav)
Shopping	3.8	<b>15.4</b>
CMS	0.0	0.0
Reddit	0.0	0.0
Gitlab	0.0	0.0
Maps	0.0	<b>7.1</b>
Avg.	1.0	<b>5.3</b>

Table 3: We generate NNetnav demonstrations using Llama-3-8B-Instruct, which we use for supervised fine-tuning of an agent based on the same LM, and find significant improvements on WebArena from 1% to 5.3%.

Domain	Zero-Shot	Self-Train (NNetnav)
<i>in-domain</i>		
Shopping	1.26	<b>1.37</b>
CMS	1.21	<b>1.29</b>
Maps	1.21	<b>1.36</b>
<i>out-of-domain</i>		
Reddit	1.08	<b>1.31</b>
Gitlab	<b>1.14</b>	1.09

Table 4: We fine-tune  $\pi_{LM}$  with NNetnav demonstrations from 3 websites, and evaluate in-domain and out-of-domain generalization with the model based evaluator that outputs a score from 1 to 5. While improvements are higher in-domain, we still find improvements on out-of-domain data.

**Fine-grained evaluation on WebArena with LLM reward.** We observe highly non-uniform improvements on WebArena with no improvements on Reddit and Gitlab in particular. We hypothesize that this is due to the coarse nature of WebArena’s success rate (SR) evaluation, since it does not provide partial credit. Thus, inspired by Pan et al. (2024), we develop a model based evaluation using the largest publicly available GPT-4o (specifically gpt-4o-2024-08-06) model to assign a graded reward from 1 to 5 to model outputs for each test instruction (see Appendix B for full prompt). We present results from model-based evaluation in Table 2. At the level of model settings we observe the same trend: Zero-Shot < SFT (Instruction-First) < SFT (NNetnav + Distil.) < SFT (NNetnav). However since this evaluation is more graded, we find consistent improvements from using NNetnav demonstrations across all websites, including Reddit and Gitlab, where improvements of 0.92 points and 0.72 points are observed, respectively. As expected, performance rankings sometimes changes with such graded evaluation e.g. on CMS, all fine-tuned models are tied in terms of SR (Table 1), but not in terms of graded reward (Table 2). Overall, we believe WebArena evaluations should incorporate both overall SR and fine-grained model based evaluation for a more comprehensive understanding of system performance.

**The Benefit of Hindsight.** We find that SFT (NNetnav) outperforms SFT (NNetnav + Distil.) on both MiniWoB++ and WebArena. Trajectories in NNetnav are obtained via a hindsight procedure: the model acts *first*, and the instruction is inferred afterward. In contrast, for NNetnav + Distil., the instruction is provided first, and the trajectory is sampled later. To understand if hindsight trajectories offer an advantage, we use the model based evaluator to measure training data quality for these settings. Specifically, we use the evaluator to assign reward to trajectories in NNetnav and NNetnav + Distil. for WebArena, and find a win-rate of 64% for NNetnav trajectories with a mean reward of 3.52 compared to a reward of 2.44 for NNetnav + Distil. We conclude that gains from NNetnav extend beyond just providing more complex instructions.

378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431

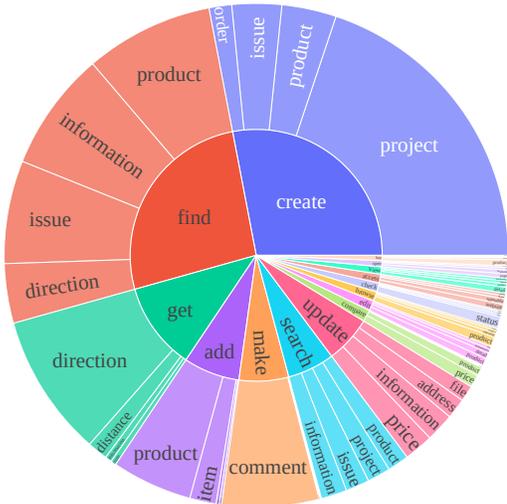


Figure 3: We plot the top 15 most common root verbs (inner circle) with their most common direct object nouns (outer circle) for instructions in NNetnav-6k. We note a high degree of diversity in intents ranging from searching for information to updating addresses and navigating for directions.

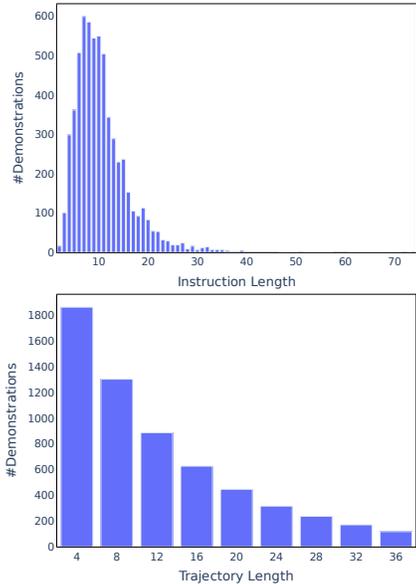


Figure 4: Length distribution of instructions and trajectories in NNetnav-6k.

**Computational savings from NNetnav pruning.** We visualize overall improvements in exploration efficiency in Fig 2. Each horizontal line depicts the fraction of interaction episodes that terminate at a specific time-step (indicated by the y-axis), with the red shaded area depicting additional actions that were prevented from early pruning. We find clear evidence of computational savings. In particular, over 60% of all exploration episodes were pruned after 16 actions for WebArena. For MiniWoB++, 65% of episodes were pruned after just 4 actions in MiniWoB++, which we identify as interactions where these first actions resulted in execution failures that our pruning heuristic successfully identified.

**Self-training with NNetnav.** Can NNetnav demonstrations from an LM be used for improving the *same* LM agent? To answer this, we collect another set of NNetnav demonstrations on WebArena, using LM components based on Llama-3-8B-Instruct. Given the limitations of this smaller model, we anticipate fewer meaningful interactions. To compensate, we increase the number of episodes to 200 episodes per website, resulting in 302 demonstrations which we use for fine-tuning the same Llama-3-8B-Instruct agent. From results in Table 3, we find improvements of 4.3 points on WebArena.

**Cross-website generalization.** Finally, we use NNetnav to conduct a small study on cross-website generalization in web-agents. Concretely, we perform supervised fine-tuning of  $\pi_{LM}$  on NNetnav demonstrations from Shopping, Maps and CMS, and evaluate generalization to Reddit and Gitlab. Here, we choose to report only model-based evaluation since success rates are 0 for these domains. From results in Table 4, we note that average reward improves by 0.06 on held out websites, and by 0.13 on in-domain websites, suggesting some potential for cross-website transfer in LLM web-agents.

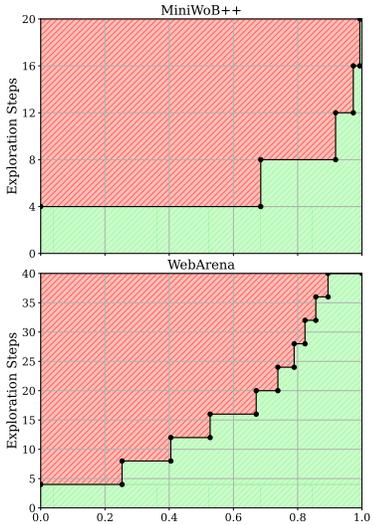


Figure 2: Horizontal lines indicate fraction of episodes terminating at corresponding y-axis exploration step. The red shaded area represents prevented actions, showing significant savings on both datasets.

Agent	#Params	WebArena SR	Open LLM?	Zero-shot?
Llama-3-8B-Instruct	8B	1.0	✓	✓
Patel et al. (2024)	72B	9.4	✓	✗
Lai et al. (2024)	7B	2.5	✓	✓
Ou et al. (2024)	7B	6.3	✓	✓
Llama-3-8B-Instruct-NNetnav	8B	<b>10.3</b>	✓	✓
Drouin et al. (2024)	Unknown (GPT-4)	23.5	✗	✓
Wang et al. (2024)	Unknown (GPT-4)	35.5	✗	✗

Table 5: We present WebArena task success rate of various prior approaches, along with key details such as model size, the use of open LLMs, and whether methods are fully zero-shot. For Lai et al. (2024), we report results from the setting that does not use human supervision. Notably, our approach, Llama-3-8B-Instruct-NNetnav, achieves a 4% improvement over the previous state-of-the-art among zero-shot agents that use open LLMs.

## 6 NNETNAV-6K

To facilitate further research on fine-tuned browser agents, we release the first large-scale dataset of over 6000 demonstrations from WebArena. Here, we use Llama-3-70B-Instruct<sup>3</sup> as the underlying LM for various components in NNetnav, and sample 3000 interactions each, with  $T_{\max}$  set to 40 as before. For each trajectory in these demonstrations, we release both accessibility tree strings as well as browser screenshots at each time-step, to support future work on text-based as well as multi-modal web agents.

To analyze diversity in these instructions, we follow methodology from Wang et al. (2022). Specifically, we use Stanza (Qi et al., 2020) to parse each instruction, identifying the verb closest to the root and its direct object. Fig 3 presents the top 15 verbs and their corresponding object nouns. Overall, we observe a diverse range of intents in the NNetnav-6k dataset. Additionally, we plot the distribution of instruction as well as trajectory lengths in Fig 4, revealing further diversity in these aspects. Table 7 provides example demonstrations from NNetnav-6k, showcasing instructions from different websites. We find a number of complex, hierarchical instructions such as *Edit the issue “Link to WCAG 2.1 instead of ...”* that refer to specific features of the website (*r/art, Swings and roundabouts*), and are plausible by design. Many of these instructions share lots of common structure (*e.g. Get walking directions from ...* and *Get cycling directions from ...*), and incorporating such structure into agents could be a promising direction for future work.

**Fine-tuning agents with NNetnav-6k demonstrations.** To demonstrate the effectiveness of NNetnav-6k in improving instruction-following in LLM web-agents, we perform supervised finetuning of the Llama-3-8B-Instruct agent on NNetnav-6k demonstrations. As described in Section 2, each demonstration expands into multiple training instances, resulting in a total of over 77,000 training examples. The results, presented in Table 5, show that our approach achieves a WebArena Success Rate (SR) of 10.3%. This marks a significant improvement over previously reported results for sub-10B models trained on synthetic datasets. To the best of our knowledge, our model sets a new state-of-the-art among agents that do not use closed-source models like GPT-4, human-annotated demonstrations, or prior knowledge of WebArena test instructions.

## 7 RELATED WORK

**Language Conditioned Digital Assistants.** Mapping instructions to action sequences in digital environments has been a long-standing goal in natural language understanding (Allen et al., 2007; Branavan et al., 2009). Most pre-LLM approaches for this rely on expert demonstrations for behavioral cloning (Chen & Mooney, 2011; Humphreys et al., 2022), along with an appropriately shaped reward

<sup>3</sup>We opted to use a locally hosted Llama-3-70B-Instruct model for collecting the larger-scale NNetnav-6k dataset, as it produced demonstrations of comparable quality to GPT-4o-mini while offering a more permissive license for downstream applications.

function (Branavan et al., 2009; Liu et al., 2018; Misra et al., 2017, among others). Here, learning is driven purely by synthetic demonstrations derived via (language model) exploration of websites.

**Linguistic Priors for Exploration.** Several prior works have used natural language priors to inform exploration for sequential decision making. Harrison et al. (2017) use a *trained model* of associations between language and state/action pairs to guide exploration during policy learning. Mu et al. (2022) use language annotations of states to train a goal generator module that provides intrinsic rewards for achieving generated goals. Similarly, Du et al. (2023) constrain exploration towards goals generated by a pre-trained LLM at each intermediate state of an agent. In contrast, NNetnav biases exploration through two new ways of using language priors. First, we use natural language as a way to filter meaningful interactions. Second, we use it as a pruning heuristic to navigate the potentially exponential search space of these interactions.

**Training Data for LLM Web-Agents.** LLMs have shown strong performance over a wide range of language understanding tasks, and are increasingly being used to interpret language in grounded contexts such as browsers (Yao et al., 2022; Lai et al., 2024; Wang et al., 2024; Patel et al., 2024; Lù et al., 2024, among others). Many of these approaches rely on human demonstrations, either for in-context learning (Yao et al., 2022; Sodhi et al., 2023; Kim et al., 2023) or for finetuning (Lù et al., 2024). However, because human demonstrations are costly, recent work trains LLM agents through synthetic demonstrations generated using instruction-first methods (Lai et al., 2024; Patel et al., 2024). One exception is Murty et al. (2024), which introduces an interaction-first method for generating synthetic demonstrations for in-context learning. Despite its novelty, their approach does not scale well to real websites due to the lack of a mechanism for effective exploration in environments with many possible interactions. In contrast, NNetnav also follows an interaction-first approach but improves efficiency by leveraging linguistically motivated pruning to navigate the space of meaningful interactions.

## 8 CONCLUSION

We propose NNetnav, a method for training web-agents with synthetic demonstrations. NNetnav flips the standard paradigm of synthetic data generation by first interacting with a website to generate trajectories and then relabeling trajectories into instructions. Real websites have a prohibitively large set of possible interactions; NNetnav searches over this space efficiently using a pruning function inspired by the hierarchical structure of language instructions: any complex instruction consists of language describable sub-tasks and so, if during an interaction a relabeling module cannot infer a meaningful sub-task for the trajectory-so-far, further exploration is pruned. We apply NNetnav to collect demonstrations on MiniWoB++ and WebArena, which are then used to fine-tune a zero-shot base LM agent. This yields significant improvements over the zero-shot baseline and outperforms standard synthetic data generation methods. In addition, we show that NNetnav enables self-training, as demonstrations collected using a base language model can improve the performance of an agent built on the same model. We find that NNetnav significantly enhances exploration efficiency due to the pruning heuristic and generates complex, realistic instructions. Lastly, we release NNetnav-6k, the largest dataset of demonstrations on WebArena to date, with over 6000 demonstrations covering broad range of intents and phenomena in WebArena.

## REPRODUCTIBILITY STATEMENT

Prompts for every LM component is provided in Appendix A, along with other details like agent action spaces. Details for model-based evaluation on WebArena are provided in Appendix B. We provide full details for post-processing demonstrations for SFT in Appendix C, and additional hyperparameters for supervised fine-tuning in Appendix D. All code and data will be available [here](#).

## REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

- 540 James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift,  
541 and William Taysom. Plow: a collaborative task learning agent. In *Proceedings of the 22nd*  
542 *National Conference on Artificial Intelligence - Volume 2, AAAI'07*, pp. 1514–1519. AAAI Press,  
543 2007. ISBN 9781577353232.
- 544 Lisa P Argyle, Ethan C Busby, Nancy Fulda, Joshua R Gubler, Christopher Rytting, and David  
545 Wingate. Out of one, many: Using language models to simulate human samples. *Political Analysis*,  
546 31(3):337–351, 2023.
- 547 S.R.K. Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning  
548 for mapping instructions to actions. In Keh-Yih Su, Jian Su, Janyce Wiebe, and Haizhou Li  
549 (eds.), *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the*  
550 *4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 82–  
551 90, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL [https://](https://aclanthology.org/P09-1010)  
552 [aclanthology.org/P09-1010](https://aclanthology.org/P09-1010).
- 553 David Chen and Raymond Mooney. Learning to interpret natural language navigation instructions  
554 from observations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 25(1):859–865,  
555 Aug. 2011. doi: 10.1609/aaai.v25i1.7974. URL [https://ojs.aaai.org/index.php/AAAI/](https://ojs.aaai.org/index.php/AAAI/article/view/7974)  
556 [article/view/7974](https://ojs.aaai.org/index.php/AAAI/article/view/7974).
- 557 Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom  
558 Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable  
559 are web agents at solving common knowledge work tasks? In Ruslan Salakhutdinov, Zico  
560 Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp  
561 (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of  
562 *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 21–27 Jul 2024. URL  
563 <https://proceedings.mlr.press/v235/drouin24a.html>.
- 564 Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek  
565 Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language  
566 models. *arXiv preprint arXiv:2302.06692*, 2023.
- 567 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha  
568 Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models.  
569 *arXiv preprint arXiv:2407.21783*, 2024.
- 570 Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin  
571 Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint*  
572 *arXiv:2305.11854*, 2023.
- 573 Brent Harrison, Upol Ehsan, and Mark O Riedl. Guiding reinforcement learning exploration using  
574 natural language. *arXiv preprint arXiv:1707.08616*, 2017.
- 575 Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair  
576 Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven  
577 approach for learning to control computers. In *International Conference on Machine Learning*, pp.  
578 9466–9482. PMLR, 2022.
- 579 Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks.  
580 *arXiv preprint arXiv:2303.17491*, 2023.
- 581 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E.  
582 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model  
583 serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating*  
584 *Systems Principles*, 2023.
- 585 Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen  
586 Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: Bootstrap and reinforce a large  
587 language model-based web navigating agent, 2024.
- 588 Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement  
589 learning on web interfaces using workflow-guided exploration. In *International Conference on*  
590 *Learning Representations*, 2018.

- 594 Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with  
595 multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024.
- 596
- 597 Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to  
598 actions with reinforcement learning. *arXiv preprint arXiv:1704.08795*, 2017.
- 599
- 600 Jesse Mu, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and  
601 Edward Grefenstette. Improving intrinsic exploration with language abstractions. *Advances in  
602 Neural Information Processing Systems*, 35:33947–33960, 2022.
- 603 Shikhar Murty, Christopher D Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. BAGEL:  
604 Bootstrapping agents by guiding exploration with language. In Ruslan Salakhutdinov, Zico  
605 Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp  
606 (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of  
607 *Proceedings of Machine Learning Research*, pp. 36894–36910. PMLR, 21–27 Jul 2024. URL  
608 <https://proceedings.mlr.press/v235/murty24a.html>.
- 609 Tianyue Ou, Frank F Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta,  
610 Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct  
611 demonstrations for digital agents at scale. *arXiv preprint arXiv:2409.15637*, 2024.
- 612 Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous  
613 evaluation and refinement of digital agents. In *First Conference on Language Modeling*, 2024.  
614 URL <https://openreview.net/forum?id=NPAQ6FKSmK>.
- 615
- 616 Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-  
617 Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks. *arXiv  
618 preprint arXiv:2405.20309*, 2024.
- 619 Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python  
620 natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual  
621 Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020. URL  
622 <https://nlp.stanford.edu/pubs/qi2020stanza.pdf>.
- 623 Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations  
624 toward training trillion parameter models. In *SC20: International Conference for High Performance  
625 Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.
- 626
- 627 Murray Shanahan, Kyle McDonell, and Laria Reynolds. Role play with large language models.  
628 *Nature*, 623(7987):493–498, 2023.
- 629 Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An  
630 open-domain platform for web-based agents. In *International Conference on Machine Learning*,  
631 pp. 3135–3144. PMLR, 2017.
- 632
- 633 Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic  
634 memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.
- 635 Paloma Sodhi, SRK Branavan, and Ryan McDonald. Heap: Hierarchical policies for web actions  
636 using llms. *arXiv preprint arXiv:2310.03720*, 2023.
- 637
- 638 Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplaner: Adaptive  
639 planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023.
- 640 Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and  
641 Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions.  
642 *arXiv preprint arXiv:2212.10560*, 2022.
- 643
- 644 Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu,  
645 David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. How far  
646 can camels go? exploring the state of instruction tuning on open resources, 2023.
- 647 Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv  
preprint arXiv:2409.07429*, 2024.

648 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V  
 649 Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models.  
 650 In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances*  
 651 *in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates,  
 652 Inc., 2022. URL [https://proceedings.neurips.cc/paper\\_files/paper/2022/file/](https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf)  
 653 [9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf).

654 Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, 1972.  
 655 ISSN 0010-0285. doi: [https://doi.org/10.1016/0010-0285\(72\)90002-3](https://doi.org/10.1016/0010-0285(72)90002-3). URL [https://www.](https://www.sciencedirect.com/science/article/pii/0010028572900023)  
 656 [sciencedirect.com/science/article/pii/0010028572900023](https://www.sciencedirect.com/science/article/pii/0010028572900023).  
 657

658 Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. ReAct meets ActRe:  
 659 Autonomous annotation of agent trajectories for contrastive self-training. In *First Conference on*  
 660 *Language Modeling*, 2024. URL <https://openreview.net/forum?id=0VLBwQGwPa>.

661 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao.  
 662 ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International*  
 663 *Conference on Learning Representations*, 2022.

664 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,  
 665 Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building  
 666 autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.  
 667

## 668 A PROMPTS FOR LM COMPONENTS

### 669 A.1 MINIWOB++

670 We start by presenting all prompts for MiniWoB++. The action space for MiniWoB++ is:

671 Listing 1: Action Space

```

672
673
674
675
676
677 noop(wait_ms: float = 1000)
678     Examples:
679         noop()
680         noop(500)
681
682 scroll(delta_x: float, delta_y: float)
683     Examples:
684         scroll(0, 200)
685         scroll(-50.2, -100.5)
686
687 fill(bid: str, value: str)
688     Examples:
689         fill('237', 'example value')
690         fill('45', 'multi-line\nexample')
691         fill('a12', 'example with "quotes"')
692
693 select_option(bid: str, options: str | list[str])
694     Examples:
695         select_option('a48', 'blue')
696         select_option('c48', ['red', 'green', 'blue'])
697
698 click(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing
699     .Literal['Alt', 'Control', 'Meta', 'Shift']] = [])
700     Examples:
701         click('a51')
702         click('b22', button='right')
703         click('48', button='middle', modifiers=['Shift'])
704
705 dblclick(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[
706     typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])
707     Examples:
708         dblclick('12')
709         dblclick('ca42', button='right')
710         dblclick('178', button='middle', modifiers=['Shift'])
711
712 hover(bid: str)
713     Examples:
714         hover('b8')
```

```

702 press (bid: str, key_comb: str)
703   Examples:
704     press('88', 'Backspace')
705     press('a26', 'Control+a')
706     press('a61', 'Meta+Shift+t')
707
708 focus (bid: str)
709   Examples:
710     focus('b455')
711
712 clear (bid: str)
713   Examples:
714     clear('996')
715
716 drag_and_drop (from_bid: str, to_bid: str)
717   Examples:
718     drag_and_drop('56', '498')
719
720 upload_file (bid: str, file: str | list[str])
721   Examples:
722     upload_file('572', 'my_receipt.pdf')
723     upload_file('63', ['/home/bob/Documents/image.jpg', '/home/bob/Documents/file.zip'])
724
725 Only a single action can be provided at once. Example:
726 fill('a12', 'example with "quotes"')
727
728 If you are done exploring, you can issue the stop action: ``stop``
729
730 Here is an example with chain of thought of a valid action when clicking on a button: "In
731 order to accomplish my goal I need to click on the button with bid 12. In summary, the next
732 action I will perform is ``click("12")``"
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

```

This is then directly used for various prompts as {action\_str}.

### Listing 2: Prompt for the Exploration Policy $\pi_{\text{explore}}$

```

731 You are an autonomous intelligent agent tasked with performing tasks on a web interface.
732 Your objective is to simulate a task that a person might request, by interacting with the
733 interface through the use of specific actions.
734
735 Here's the information you'll have:
736 DOM Representation: This is the current webpage's Document Object Model (DOM)
737 representation as a string.
738 The previous action: This is the action you just performed. It may be helpful to track your
739 progress.
740 Trajectory: This is a sequence of natural language descriptions of the agent's interaction
741 with the web-browser.
742 Person Description: The description of a specific kind of person whose task you are
743 supposed to simulate.
744
745 You can perform the following actions: {action_str}
746
747 To be successful, it is very important to follow the following rules:
748 1. You should only issue an action that is valid given the current observation.
749 2. You should only issue one action at a time.
750 3. You should reason step by step and then issue the next action.
751 4. Make sure to wrap your action in a code block using triple backticks.
752 5. The DOM / Accessibility Tree only shows the visible part of the webpage. If you need to
753 interact with elements that are not visible, you can scroll to them using the scroll action
754 . Often submit buttons are not visible and are at the bottom of the page. To scroll to the
755 bottom of the page, use the scroll action with a large value for the y coordinate.
756 6. To generate an interesting task, make sure you issue atleast 4 actions before stopping.
757 More interesting tasks typically involve more interactions with the browser.
758 7. You can issue atmost 20 actions before stopping, but feel free to output the stop action
759 early if you want to stop exploring. Don't generate anything after stop.
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775

```

### Listing 3: Prompt for $\Delta_{\text{LM}}$

```

776 You are given the output of an action taken by an autonomous intelligent agent navigating a
777 web-interface to fulfill a task given by a user. Your objective is to produce a
778 description of the changes made to the state of the browser.
779
780
781
782
783
784
785

```

756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809

Here's the information you'll have:  
Initial state of the browser as a DOM representation: This is the webpage's Document Object Model (DOM) representation as a string.  
Final state of the browser as a DOM representation: This is the DOM representation after the agent took the action.

The action taken by the agent: This is the action taken by the agent to change the state of the browser.

The actions the agent can take come from the following categories: {action\_str}

To be successful, it is very important to follow the following rules:

1. Explicitly think about the various features on the website and how the interaction with the website changed these features
2. Provide the description of changes in one or two sentences.
3. Strictly follow the format "State change: <your-answer>" for your output

#### Listing 4: Prompt for the Trajectory Labeling function $Lf_{LM}$

Given a task from a user, an autonomous intelligent agent carries out a sequence of actions on a web-interface.

The actions the agent can take fall under the following categories: {action\_str}

Your objective is to guess the instruction the user gave, given the following information:  
Trajectory: This is a sequence of natural language descriptions of the agent's interaction with the web-browser.

To be successful, it is very important to follow the following rules:

1. Explicitly think about how the trajectory is a valid way to achieve the instruction, before outputting the instruction.
2. Start by thinking by outputting Thought: <your-reasoning>.
3. End your answer by strictly following the format "Instruction: <your-answer>" for your output.

#### Listing 5: Prompt for the reward function $s_{LM}$

An autonomous intelligent agent navigating a web browser is given an instruction by a user. Your objective is to give a score to the agent based on how well it completed its task. Your score must be on the scale of 1 to 5. Give a score of 5 only when there are no errors. To do this task you are provided with the following information:

Instruction: This is the natural language instruction given to the agent.  
Trajectory: This is a sequence of natural language descriptions of the agent's interaction with the web-browser.

To be successful, it is very important to follow the following rules:

1. Explicitly think about what is needed to follow the instruction correctly on the website and if the trajectory reflects these steps.
- 2 Give a score of 4 if there are very minor errors, or if the task was more than 70% completed. Give a score of 3 (or below) if the model made very little progress towards the given instruction or if there are major errors.
3. Start by thinking by outputting Thought: <your-reasoning>.
4. End your answer by strictly following the format "Reward: <your-answer>" for your output

#### Listing 6: Prompt for the base LLM agent $\pi_{LM}$

You are an autonomous intelligent agent tasked with performing tasks on a web interface. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:  
DOM Representation: This is the current webpage's Document Object Model (DOM) representation as a string.  
The user's objective: This is the task you're trying to complete.  
The previous action: This is the action you just performed. It may be helpful to track your progress.

You can perform the following actions: {action\_str}

To be successful, it is very important to follow the following rules:

810  
811  
812  
813  
814  
815  
816  
817

1. You should only issue an action that is valid given the current observation
2. You should only issue one action at a time.
3. You should follow the examples to reason step by step and then issue the next action.
4. Make sure to wrap your action in a code block using triple backticks.
5. The DOM / Accessibility Tree only shows the visible part of the webpage. If you need to interact with elements that are not visible, you can scroll to them using the scroll action. Often submit buttons are not visible and are at the bottom of the page. To scroll to the bottom of the page, use the scroll action with a large value for the y coordinate.
6. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

818  
819  
820

### Listing 7: Prompt for adding reasoning steps retroactively to filtered trajectories

821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851

```
You are an autonomous intelligent agent that carries out a sequence of actions on a web-
interface, given an instruction from a user. The actions you can take fall under the
following categories: {action_str}

You are given the user instruction, an intermediate state of the web-page (in the form of a
DOM string), and the action you took for that intermediate state. Your objective is to
output your reasoning for choosing that specific action. In summary, you are given the
following
Instruction: This is the instruction given by the user.
Intermediate State: This is the state of the web-page at some time-step t.
Action: This is the action taken by the agent at time-step t.

Here are some example reasoning outputs for some random tasks:

Instruction: select_option('checkbox_group', ['appetizing', 'bunnies', 'weird', 'dull'])
Intermediate State: {state}
Action: click('21')

Output: Let's think step by step. The user instruction is to select multiple options from a
checkbox group, specifically 'appetizing', 'bunnies', 'weird', and 'dull'. In the current
intermediate state, I can see the checkboxes for these options, and they are represented by
input elements with specific bid attributes. The action I took was to click on the
checkbox corresponding to 'appetizing', which is represented by the input with bid '18'.
Now I will check the 'bunnies' checkbox, moving closer to fulfilling the user's request to
select the specified options. In summary, the next action I will perform is ``click('21')
```

Instruction: click('Macie')
Intermediate State: {state}
Action: click('21')

Output: Let's think step by step. The user instruction is to click on 'Macie', which is
represented by the HTML element with bid '21'. In the intermediate state of the web page, '
Macie' is identified as a folder that is expandable and clickable. By executing the click
action on this element, I will open the folder associated with 'Macie', which aligns with
the user's intention to interact with it. In summary, the next action I will perform is ``
click('21')``

To be successful, it is very important to follow the following rules:
1. Explicitly think about how executing the given action will change the web-page in a way
that gets the agent closer to achieving the user instruction
2. You should strictly follow the format "Output: Let's think step by step. <agent
reasoning>. In summary, the next action I will perform is ``[action]`` ". Here make sure
to replace [action] with the provided action.
```

852  
853

## A.2 WEBARENA

854  
855  
856  
857

Next, we present all prompts for WebArena. The action space for WebArena is:

858  
859  
860  
861  
862  
863

### Listing 8: Action Space

```
Page Operation Actions:
`click [id]`: This action clicks on an element with a specific id on the webpage.
`type [id] [content] [press_enter_after=0|1]`: Use this to type the content into the field
with id. By default, the "Enter" key is pressed after typing unless press_enter_after is
set to 0.
`hover [id]`: Hover over an element with id.
`press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g.,
Ctrl+v).
```

864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879

```

'scroll [direction=down|up]': Scroll the page up or down.

Tab Management Actions:
'new_tab': Open a new, empty browser tab.
'tab_focus [tab_index]': Switch the browser's focus to a specific tab using its index.
'close_tab': Close the currently active tab.

URL Navigation Actions:
'goto [url]': Navigate to a specific URL.
'go_back': Navigate to the previously viewed page.
'go_forward': Navigate to the next page (if a previous 'go_back' action was performed).

Completion Action:
'stop ["done"]': Issue this action when you are done.

Homepage:
If you want to visit other websites, check out the homepage at http://homepage.com. It has
a list of websites you can visit.

```

880 This is then directly used for various prompts as {action\_str}.

### 881 Listing 9: Prompt for the Exploration Policy $\pi_{\text{explore}}$

882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912

```

You are an autonomous intelligent agent tasked with navigating a web browser. Your
objective is to simulate a task that a person might perform, by interacting with the
browser through the use of specific actions.

Here's the information you'll have:

The current web page's accessibility tree: This is a simplified representation of the
webpage, providing key information.
The current web page's URL: This is the page you're currently navigating.
The open tabs: These are the tabs you have open.
The previous action: This is the action you just performed. It may be helpful to track your
progress.
Trajectory: This is a sequence of natural language descriptions of the agent's interaction
with the web-browser.
Person Description: The description of a specific kind of person whose task you are
supposed to simulate.

The actions you can perform fall into several categories: {action_str}

To be successful, it is very important to follow the following rules:
1. You should only issue an action that is valid given the current observation
2. You should only issue one action at a time.
3. You should follow the examples to reason step by step and then issue the next action.
4. Generate the action in the correct format. Start by reasoning out the current situation.
End with "In summary, the next action I will perform is" phrase, followed by action inside
``` . For example, "Let's think step-by-step. Given the current state, I need to click
on the like button which has id 1234. In summary, the next action I will perform is ```
click [1234]```".
5. To generate an interesting task, make sure you issue atleast 4 actions before stopping.
More interesting tasks typically involve more interactions with the browser.
6. You can issue atmost 40 actions before stopping, but feel free to output the stop action
early if you want to stop exploring. Don't generate anything after stop.

Here are some example outputs for some random tasks:
1. Let's think step-by-step. This page list the information of HP Inkjet Fax Machine, which
is the product identified in the objective. Its price is $279.49. I think I have achieved
the objective. I will issue the stop action with the answer. In summary, the next action I
will perform is ```stop [$279.49]```
2. Let's think step-by-step. This page has a search box whose ID is [164]. According to the
nominatim rule of openstreetmap, I can search for the restaurants near a location by "
restaurants near". I can submit my typing by pressing the Enter afterwards. In summary, the
next action I will perform is ```type [164] [restaurants near CMU] [1]```

```

### 913 Listing 10: Prompt for $\Delta_{LM}$

914  
915  
916  
917

```

You are given the output of an action taken by an autonomous intelligent agent navigating a
web browser. Your objective is to produce a description of the changes made to the state
of the browser.

```

918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971

```

Here's the information you'll have:

Initial state of the browser as an accessibility tree: This is a simplified representation
of the webpage, providing key information.
Final state of the browser: This is the accessibility tree representation after the agent
took the action

The action taken by the web agent: The agent can take actions that fall under the following
categories: {action_str}

To be successful, it is very important to follow the following rules:
1. Explicitly think about the various features on the website and how the interaction with
the website changed these features
2. Provide the description of changes in one or two sentences.
3. Strictly follow the format "State change: <your-answer>" for your output

```

### Listing 11: Prompt for the Trajectory Labeling function $Lf_{LM}$

```

Given an instruction from a user, an autonomous intelligent agent carries out a sequence of
actions on a web-browser. The actions the agent can take fall under the following
categories: {action_str}

Your objective is to guess the instruction the user gave, given the following information:
Trajectory: This is a sequence of natural language descriptions of the agent's interaction
with the web-browser.

Here are some examples of user instructions:
1. Get the distance from SF airport to Palo Alto.
2. Find out the price of Apple airpods
3. Add 5 items to cart
4. Make a comment on the first post in the r/gaming subreddit.

To be successful, it is very important to follow the following rules:
1. Explicitly think about how the trajectory is a valid way to achieve the instruction,
before outputting the instruction.
2. Start by thinking by outputting Thought: <your-reasoning>.
3. End your answer by strictly following the format "Instruction: <your-answer>" for your
output.

```

### Listing 12: Prompt for the reward function $s_{LM}$

```

An autonomous intelligent agent navigating a web browser is given an instruction by a user.
Your objective is to give a score to the agent based on how well it completed its task.
Your score must be on the scale of 1 to 5. Give a score of 5 only when there are no errors.
To do this task you are provided with the following information:

Instruction: This is the natural language instruction given to the agent.
Trajectory: This is a sequence of natural language descriptions of the agent's interaction
with the web-browser.

To be successful, it is very important to follow the following rules:
1. Explicitly think about what is needed to follow the instruction correctly on the website
and if the trajectory reflects these steps.
2 Give a score of 4 if there are minor errors, or if the task was more than 70% completed.
Give a score of 3 (or below) if the model made very little progress towards the given
instruction.
3. Start by thinking by outputting Thought: <your-reasoning>.
4. End your answer by strictly following the format "Reward: <your-answer>" for your output

```

### Listing 13: Prompt for the base LLM agent $\pi_{LM}$

```

You are an autonomous intelligent agent tasked with navigating a web browser. You will be
given web-based tasks. These tasks will be accomplished through the use of specific actions
you can issue.

Here's the information you'll have:
The user's objective: This is the task you're trying to complete.
The current web page's accessibility tree: This is a simplified representation of the
webpage, providing key information.
The current web page's URL: This is the page you're currently navigating.
The open tabs: These are the tabs you have open.

```

972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

The previous actions: These are all the action you have performed. It may be helpful to track your progress.

The actions you can perform fall into several categories: {action\_str}

To be successful, it is very important to follow the following rules:

1. You should only issue an action that is valid given the current observation
2. You should only issue one action at a time.
3. You should follow the examples to reason step by step and then issue the next action.
4. You are strictly forbidden from issuing a goto action to a URL that is not on the homepage.
5. Generate the action in the correct format. Start by reasoning about the current situation. End with "In summary, the next action I will perform is" phrase, followed by action inside `````. For example, "Let's think step-by-step. Given the current state, I need to click on the like button which has id 1234. In summary, the next action I will perform is ``click [1234]``".
6. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some example outputs for some random tasks:

1. Let's think step-by-step. This page list the information of HP Inkjet Fax Machine, which is the product identified in the objective. Its price is \$279.49. I think I have achieved the objective. I will issue the stop action with the answer. In summary, the next action I will perform is ``stop [\$279.49]``
2. Let's think step-by-step. This page has a search box whose ID is [164]. According to the nominatim rule of openstreetmap, I can search for the restaurants near a location by "restaurants near". I can submit my typing by pressing the Enter afterwards. In summary, the next action I will perform is ``type [164] [restaurants near CMU] [1]``

#### Listing 14: Prompt for adding reasoning steps retroactively to filtered trajectories

You are an autonomous intelligent agent that carries out a sequence of actions on a web-interface, given an instruction from a user.

The actions you can take fall under the following categories: {action\_str}

You are given the user instruction, an intermediate state of the web-page (in the form of an accessibility tree), and the action you took for that intermediate state. Your objective is to output your reasoning for choosing that specific action. In summary, you are given the following

Instruction: This is the instruction given by the user.

Intermediate State: This is the state of the web-page at some time-step  $t$ .

Action: This is the action taken by the agent at time-step  $t$ .

Here are some example reasoning outputs for some random tasks

Instruction: Find directions from CMU to Downtown Pittsburgh

Intermediate State: {state}

Action: click [482]

Output: Let's think step-by-step. Since my previous attempt to click the 'Go' button failed, I will try clicking the 'Find directions between two points' link instead. This could help in planning public transportation routes effectively. In summary, the next action I will perform is ``click [482]``

Instruction: Navigate to the 'Orders' section and create a new customer for order processing.

Intermediate State: {state}

Action: click [1583]

Output: Let's think step-by-step. Currently, I am in the Orders section of the Magento Admin panel. I see a button labeled 'Create New Order,' which is likely the next step for creating orders. In summary, the next action I will perform is ``click [1583]`` to create a new order.

To be successful, it is very important to follow the following rules:

1. Explicitly think about how executing the given action will change the web-page in a way that gets the agent closer to achieving the user instruction
2. Make sure to wrap the action inside triple backticks (such as ``click [1234]``, ``type [12] [Hotels near CMU]``) as shown in the examples and strictly follow the format "Output: Let's think step by step. <agent reasoning>. In summary, the next action I will perform is ``[action]``". Here make sure to replace [action] with the provided action.

#### Listing 15: Prompt for appending the special [stop] action in WebArena

1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079

```

Given an instruction from a user, an autonomous intelligent agent carries out a sequence of
actions on a web-browser. The actions the agent can take fall under the following
categories (we also provide the descriptions of each action): {action_str}

You are given the user instruction, and the final webpage after the agent finished its task
. Unfortunately, we forgot to collect the final stop action from the agent. Your objective
is to guess the agent's stop action. To do this, you are given the following
Instruction: This is the instruction given by the user.
Final State: This is the final state of the web-page after the agent executed its actions
on the browser.

Here are some examples of valid outputs:
1. Let's think step-by-step. The task requires me to find the person with the most number
of upvotes. I see the answer to that is Alice Oh. Therefore I will stop now. In summary, my
next action will be ```stop [Alice Oh]```.
2. Let's think step-by-step. The task required setting the price of Sprite to 25$ which I
have already done. Thus I will stop now. In summary, my next action will be ```stop [N/A
]```.
3. Let's think step-by-step. I was supposed to find the distance from Brad's house to the
coffee shop. I see this info on the map as 0.3 miles. Thus I will issue the stop action. In
summary, my next action will be ```stop [0.3 miles]```

To be successful, it is very important to follow the following rules:
1. Explicitly think about what kind of a stop action was needed. For instance, if the user
requests information (e.g. Search for airports near CMU or Find developers with more than 5
merge requests), then the stop action should have the answer based on the final web-page (
e.g. ```stop [Pittsburgh Airport]``` or ```stop [Don Knuth, Alan Turing]```). Otherwise,
the stop action should be without any arguments (e.g. ```stop```).
2. Your output should include reasoning steps. Also make sure to wrap the stop action in
triple backticks for e.g. ```stop [<your answer>]```. Overall, follow the following format
for your output: "Let's think step by step. <your reasoning>. In summary, my next action
should be ```stop [<your answer>]```."

```

## B MODEL-BASED EVALUATION: DETAILS

For each  $(g, \tau)$  pair we first use  $\Delta_{LM}$  to compute the sequence of changes  $\delta_r$ , which is then passed into the reward module along with  $g$ . We implement the reward module as a prompted LM, using the largest GPT-4o (specifically gpt-4o-2024-08-06) with the following prompt:

Listing 16: Prompt for the model-based evaluator

```

An autonomous intelligent agent navigating a web browser is given an instruction by a user.
Your objective is to give a score to the agent based on how well it completed its task.
Your score must be on the scale of 1 to 5. Give a score of 5 only when there are no errors.
To do this task you are provided with the following information:

Instruction: Natural language instruction given to the agent.
Trajectory: Sequence of language descriptions of the agent's interaction with the browser.

Here are some guidelines for scoring:
1. Give a score of 5 if there are no errors.
2. Give a score of 4 if the task was almost correctly done (i.e. for form filling, most of
the fields are filled or for a search task, a query was correctly typed, and the agent
navigated to the right links).
3. Give a score of 3 if the task was only partially completed (i.e for form filling, less
than half the fields are filled out) and if there are other minor execution errors.
4. Give a score of 1 or 2 if there are major execution errors, or the task was hardly
completed, or if the agent did something completely unrelated.

To be successful, it is very important to follow the following rules:
1. Explicitly think about what is needed to follow the instruction correctly on the website
and if the trajectory reflects these steps.
2. Start by thinking by outputting Thought: <your-reasoning>.
3. End your answer by strictly following the format "Reward: <your-answer>" for your output

```

## C PROCESSING DEMONSTRATIONS FOR SFT

As mentioned in §2, for supervised finetuning each demonstration is converted into multiple training instances. We perform this conversion differently based on input features of  $\pi_{LM}$ .

Dataset	NNetnav	NNetnav (self-train)	Instruction-First
MiniWoB++	2288	-	8559
WebArena	9737	2204	1681

Table 6: Number of instances for supervised training experiments of §5 under various settings. Between NNetnav and Instruction-First, we only control for the number of episodes for a fair comparison, which results in different number of training instances.

**MiniWoB++.** For MiniWoB++,  $\pi_{LM}$  conditions on the current observation  $o_t$ , the goal  $g$  and the previous action  $a_{t-1}$  (see prompt in §A.1). Thus, we pre-process each  $(g, \tau)$  demonstration into inputs  $(g, o_t, a_{t-1})$  with the corresponding reasoning step and action  $(r_t, a_t)$  as the target output.

**WebArena.** For WebArena,  $\pi_{LM}$  conditions on the current observation  $o_t$ , the goal  $g$  and all previous actions  $\{a_1, a_2, \dots, a_{t-1}\}$  (see prompt in §A.2). Thus, we pre-process each  $(g, \tau)$  demonstration into inputs  $(g, o_t, \{a_{<t}\})$  with  $(r_t, a_t)$  as the target output.

We report number of training instances from NNetnav and instruction-first generation for both environments in Table 6.

## D TRAINING DETAILS

**Additional Hyperparameters.** For all Llama-3-8B-Instruct finetuning experiments, we set the batch size for training as  $128 \times 4096$ , train for 5 epochs, with a learning rate of  $2e-5$  that linearly warms up from 0 over 3% of total training steps. We use 4 A100 GPUs with 80GB GPU memory, and additionally use DeepSpeed ZeRO-3 (Rajbhandari et al., 2020) to speed up training and manage memory.

## E NNETNAV-6K EXAMPLES

1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187

<i>Shopping</i>	
Find a kitchen utensil organizer.	
Find a kitchen utensil organizer within a certain budget.	
Write a review for the product “Citric Acid 2 Pounds 100% Pure Organic Food Grade”.	
Find the price of kitchen gadgets that can be used for dining and entertaining, and add them to the cart.	
Browse for women’s clothing items, specifically jumpsuits, and add some to cart.	
<i>CMS</i>	
Change the stock status of the Sprite Stasis Ball 65 cm to In Stock.	
Create a new product in the Magento Admin panel with the name 'New Fashionable Watch', SKU 'New Fashionable WatchFW101', price \$100.00, and set as new from 2024-01-01.	
Update the price of Sprite Stasis Ball 55 cm to \$24.50 and set its quantity to 50.	
Add two products, “Abominable Hoodie” and “Samsung Smart TV”, with respective prices \$99.99 and \$50.00, and then start the process of adding a new customer.	
<i>Reddit</i>	
Create a new forum called “Funny Stuff” with the title “Memes and LOLs”, description “A place for sharing and discussing funny memes and LOLs”, and sidebar “Memes of the day”.	
Find a webpage related to intraday trading strategies on the wallstreetbets forum.	
Find and participate in a discussion on the wallstreetbets forum about intraday trading strategy, specifically on a post titled “Swings and roundabouts”.	
Change my profile settings to use Deutsch as the language and Africa/Accra as the time zone, and then view the search results for “r/art”.	
<i>Maps</i>	
Get walking directions from Logan Street, Pittsburgh, PA to Carnegie Mellon University on OpenStreetMap.	
Get the cycling directions from Brooklyn to Manhattan.	
Find the driving directions from TLC Medical Transportation Services in Syracuse to Times Square in Manhattan.	
<i>Gitlab</i>	
Create a new project named 'My Blog Post Project' and add an Apache License 2.0 file.	
Create a new project, add a LICENSE file with Apache License 2.0, and approve the “Add verification functions” merge request.	
Search for a README.md file within the “My New Project” project and verify its contents.	
Edit the issue “Link to WCAG 2.1 instead of 2.0?” in the First Contributions project on GitLab by updating its title and description to point to WCAG 2.1 guidelines instead of 2.0 guidelines.	
Investigate the node-http-proxy project’s issue #992 regarding connection headers and determine its relevance to the Byte Blaze project.	
Investigate and comment on the “Outdated dependencies” issue in the “Byte BlazeByte BlazeByte Blaze / accessible-html-content-patterns” project.	

Table 7: Some Example demonstrations obtained from NNetnav-6k. We note that these instructions are hierarchical, refer to concrete features and entities and plausible by design.