
WebArena: A Realistic Web Environment for Building Autonomous Agents

Shuyan Zhou^{♣*} Frank F. Xu^{♣*}
Hao Zhu^{♣†} Xuhui Zhou^{♣†} Robert Lo^{♣†} Abishek Sridhar^{♣†}
Xianyi Cheng[♣] Yonatan Bisk[♣] Daniel Fried[♣] Uri Alon[♣] Graham Neubig^{♣♣}
♣Carnegie Mellon University ♣Inspired Cognition
{shuyanzh, fangzhex, gneubig}@cs.cmu.edu

Abstract

With generative AI advances, the exciting potential for autonomous agents to manage daily tasks via natural language commands has emerged. However, current agents are primarily created and tested in simplified synthetic environments, substantially limiting real-world scenario representation. In this paper, we build an environment for agent command and control that is *highly realistic* and *reproducible*. Specifically, we focus on agents that perform tasks on the web, and we create an environment with fully functional websites from four common domains: e-commerce, social forum discussions, collaborative software development, and content management. Our environment is enriched with tools (*e.g.*, a map) and external knowledge bases (*e.g.*, user manuals) to encourage human-like task-solving. Building upon our environment, we release a set of benchmark tasks focusing on evaluating the *functional correctness* of task completions. The tasks in our benchmark are diverse, long-horizon, and designed to emulate tasks that humans routinely perform on the internet. We design and implement several autonomous agents, integrating recent techniques such as reasoning before acting. The results demonstrate that solving complex tasks is challenging: our best GPT-4-based agent only achieves an end-to-end task success rate of 10.59%. These results highlight the need for further development of robust agents, that current state-of-the-art LMs are far from perfect performance in these real-life tasks, and that WebArena can be used to measure such progress.

Our code, data, environment reproduction resources, and video demonstrations are publicly available at <https://webarena.dev/>.

1 Introduction

Autonomous agents that could perform everyday tasks via human natural language commands could significantly augment human capabilities, given their potential to enhance efficiency and promote broader accessibility. Nonetheless, to fully leverage the power of these autonomous agents, it is crucial to understand their behavior within an environment that is both *authentic* and *reproducible*. This will measure the ability of agents on real tasks that human users care about and allow them to be evaluated in a fair and consistent manner.

Current environments tend to *over-simplify* real-world situations. As a result, the functionality of many environments is a limited version of their real-world counterparts, leading to a lack of task diversity within the environment [27, 1, 10, 23, 29, 30, 37]. In addition, these simplifications often

*Lead contributors.

†Equal contribution.

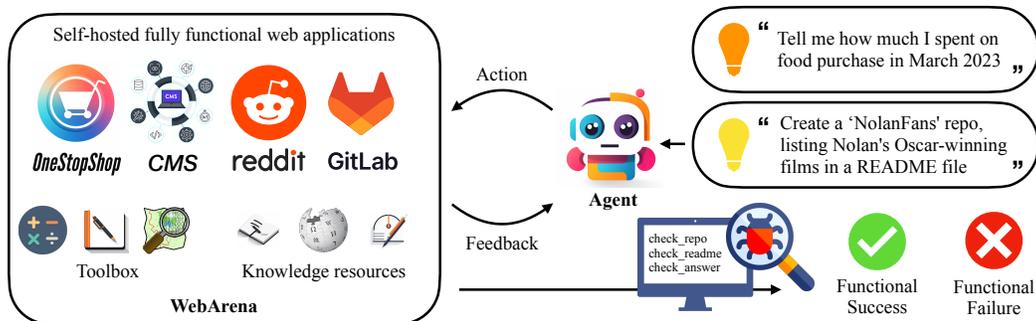


Figure 1: WebArena is a standalone, self-hostable web environment for building autonomous agents. WebArena creates websites from four popular categories with functionality and data mimicking their real-world equivalents. To emulate human problem-solving, WebArena also embeds tools and knowledge resources as independent websites. WebArena introduces a benchmark on interpreting *high-level realistic* natural language command to concrete web-based interactions. We provide annotated programs designed to programmatically validate the functional correctness of each task.

lower the complexity of tasks as compared to their execution in the real world [25, 29, 37]. Finally, some environments are presented as a static resource [27, 7] where agents are confined to accessing only those states that were previously cached during data collection, thus limiting the breadth and diversity of exploration. On the evaluation aspect, many environments focus on comparing the *surface form* of the predicted action sequences with reference action sequences, disregarding the functional correctness of the executions and possible alternative solutions [25, 12, 35, 19, 7]. These limitations often result in a discrepancy between simulated environments and the real world. Such constraints can potentially impact the generalizability of AI agents to successfully understand, adapt, and operate within complex real-world situations.

In this work, we introduce WebArena, a *realistic* and *reproducible* web environment designed to facilitate the development of autonomous agents capable of executing tasks (§2). An overview of WebArena is in Figure 1. Our environment comprises four fully operational, self-hosted web applications, each representing a distinctive domain prevalent on the internet: online shopping, discussion forums, collaborative development, and business content management. Furthermore, WebArena incorporates several utility tools, such as map, calculator, and scratchpad, to best support possible human-like task executions. Lastly, WebArena is complemented by an extensive collection of documentation and knowledge bases that vary from general resources like English Wikipedia to more domain-specific references, such as manuals for using the integrated development tool [8]. The content populating these websites is extracted from their real-world counterparts, preserving the authenticity of the content served on each platform. We deliver the hosting services using Docker containers with gym-APIs [4], ensuring both the usability and the reproducibility of WebArena.

Along with WebArena, we release a ready-to-use benchmark with 812 long-horizon web-based tasks (§3). Each task is described as a high-level natural language intent, emulating the abstract language usage patterns typically employed by humans [2]. Two example intents are shown in the upper left of Figure 1. We focus on evaluating the *functional correctness* of these tasks, *i.e.*, does the result of the execution actually achieve the desired goal (§3.3). For instance, to evaluate the example in Figure 2, our evaluation method verifies the concrete contents in the designated repository. This evaluation is not only more reliable [40, 5, 33] than comparing the vanilla action sequences [25, 7] but also accommodates a range of potential valid paths to achieve the same goal, which is a ubiquitous phenomenon in sufficiently complex tasks.

We use this benchmark to evaluate several agents that could follow NL command and perform web-based tasks (§4). These agents are implemented with a variety of approaches, from those predicting subsequent action directly based on current observations and history to more sophisticated agents using step-by-step reasoning [34, 39]. These agents are implemented in a few-shot in-context learning fashion with powerful large language models (LLMs) such as GPT-3.5 and GPT-4. Experiment results show that the best GPT-4 agent performance is somewhat limited, with an end-to-end task success rate of 10.59%. We hypothesize that the limited performance of current LLMs stems from a lack of

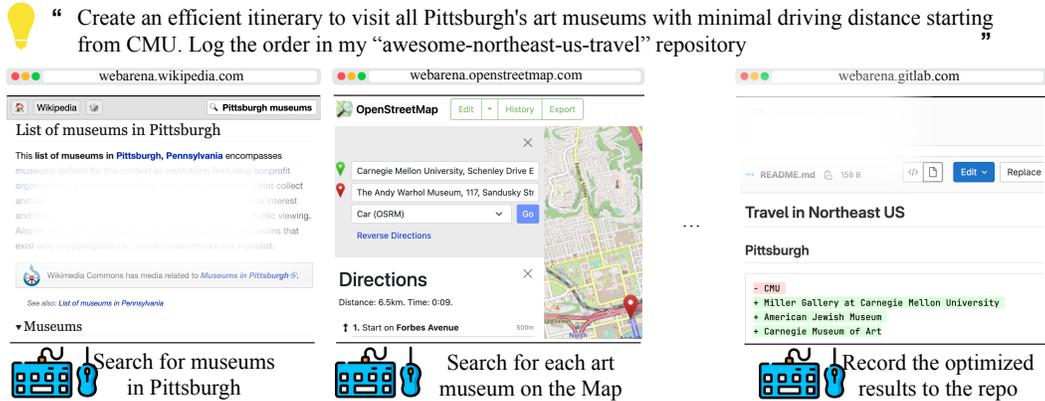


Figure 2: A high-level task that can be fully executed in WebArena. Completing such tasks requires sophisticated, long-term planning and reasoning capability. To accomplish the goal stated at the top, an agent needs to find out what art museums are located in Pittsburgh by searching Wikipedia. Next, it should identify the location of each museum on a map, optimizing the itinerary based on the information collected. Finally, the agent needs to update the README file in the appropriate repository with the planned route.

crucial capabilities such as active exploration and failure recovery to successfully perform complex task (§5.2). These outcomes underscore the necessity for further development towards robust and effective agents [18] in WebArena.

2 WebArena: Web Sites as an Environment for Autonomous Agents

Our goal is to create a *realistic* and *reproducible* web environment. We achieve reproducibility by having the environment be standalone, not relying on live websites. This circumvents technical challenges such as bots being subject to CAPTCHAs, unpredictable content modifications, and configuration changes, which obstruct a fair comparison across different systems over time. We achieve realism by using open-source libraries that underlie many in-use sites from several popular categories and importing data to our environment from their real-world counterparts.

2.1 Website Selection

In order to decide which categories of web site to use, we first conducted an analysis of approximately 200 examples from the authors’ actual web browser histories. Each author delved into their browsing histories, summarizing the goal of particular segments of their browser session. Based on this, we classified the visited websites into abstract categories. We then identified the four most salient categories and implemented one instance per category based on this analysis: (1) E-commerce platforms supporting online shopping activities (*e.g.*, Amazon, eBay), (2) social forum platforms for opinion exchanges (*e.g.*, Reddit, StackExchange), (3) collaborative development platforms for software development (*e.g.*, GitLab), and (4) content management systems (CMS) that manage the creation and revision of the digital content (*e.g.*, online store management).

In addition to these platforms, we selected three utility-style tools that are frequently used in web-based tasks: (1) a map for navigation and searching for information about points of interest (POIs) such as institutions or locations (2) a calculator, and (3) a scratchpad for taking notes.

Recognizing the critical role of information-seeking and knowledge acquisition in web-based tasks, we also incorporated various knowledge resources into our environment. These resources range from general information repositories, such as the English Wikipedia, to more specialized knowledge bases, such as the website user manuals.

Implementation We leveraged open-source libraries relevant to each category to build our own versions of an E-commerce website (OneStopShop), GitLab, Reddit, an online store content manage-

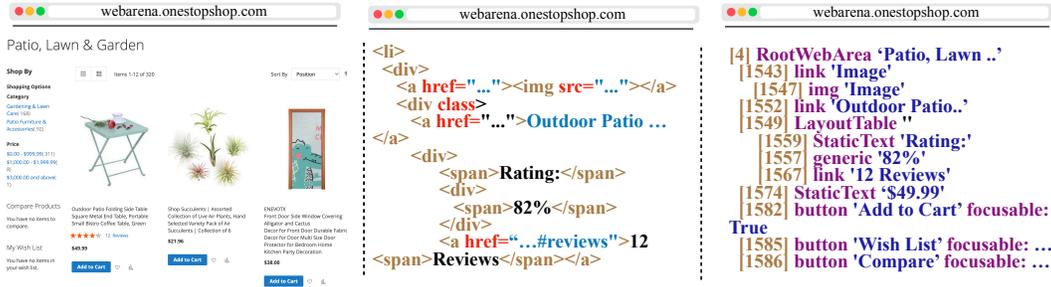


Figure 3: We design the observation to be the URL and the content of a web page, with options to represent the content as a screenshot (left), HTML DOM tree (middle) and accessibility tree (right). The content of the middle and right figures are trimmed to save space.

ment system (CMS), a map, and an English Wikipedia. Then we imported data from their real-world counterparts via a sampling method. As an example, our version of GitLab was developed based on the actual GitLab project.³ We carefully emulated the features of a typical code repository by including both popular projects with many issues and pull requests and smaller, personal projects. Details of all websites in WebArena can be found in Appendix A.1. We deliver the environment as dockers and provide scripts to reset the environment to a deterministic initial state. These details are in Appendix A.2.

2.2 Observation Space

We design the observation space to roughly mimic the web browser experience: a web page URL, the opened tabs and the web page content of the focused tab. WebArena is the first web environment to consider multi-tab web-based tasks to promote tool usage, direct comparisons and references across tabs, and other functionalities. The multi-tab functionality offers a more authentic replication of human web browsing habits compared to maintaining everything in a single tab. We provide flexible configuration to render the page content in many modes: (see Figure 3 for an example): (1) the raw web page HTML, composed of a Document Object Model (DOM) tree, as commonly used in past work [27, 7, 19]; (2) the screenshot, a pixel-based representation that represents the current web page as an RGB array and (3) the accessibility tree of the web page.⁴ The accessibility tree is a subset of the DOM tree with elements that are *relevant* and *useful* for displaying the contents of a web page. Every element is represented as its role (*e.g.*, a link), its text content, and its properties (*e.g.*, whether it is focusable). Accessibility trees largely retain the *structured* information of a web page while being more compact than the DOM representation.

We provide an option to limit the content to the contents within a viewport for all modes. This ensures that the observation can be input into a text-based model with limited context length or an image-based model with image size or resolution requirements.

2.3 Action Space

Following previous work on navigation and operation in web and embodied environments [21, 8], we design a compound action space which emulates the keyboard and mouse operations available on web pages.

Table 1 lists all the available actions categorized into three distinct groups. The first category includes elemental operations such as clicking, hovering, typing, and key combination pressing. The second comprises tab management actions such as opening, closing, and switching between tabs. The third category consists of URL navigation actions, such as visiting a specific URL or navigating forwards and backward in the browsing history.

WebArena allows agents to refer to elements on web pages with different approaches. Elements can be selected either by their on-screen coordinates, represented as (x, y) , or by a unique element ID.

³<https://gitlab.com/gitlab-org/gitlab>

⁴https://developer.mozilla.org/en-US/docs/Glossary/Accessibility_tree

Action Type	Description
noop	Do nothing
click(element)	Click at an element
hover(element)	Hover on the element
type(element, text)	Input the text to the element
key_press(key comb)	Press the key combination (<i>e.g.</i> , ctrl+v)
tab_focus(page number)	Bring the open tab with page number to the front
new tab	Open a new tab
tab close	Close the current page and focus the last open page
go back	Visit the last URL visited
go forward	Undo go back operation
goto(URL)	Go to a URL in the current page

Table 1: Action Space of WebArena

This ID is an artifact generated when traversing the Document Object Model (DOM) tree or the accessibility tree. With element IDs, the element selection is transformed into an n -way classification problem, thereby eliminating any disambiguation efforts required from the agent or the underlying implementation. An accessibility tree with the assigned element IDs is presented in Figure 3. For example, when the agent issues an action of `textttclick [1582]`, where `[1582]` is the unique ID for the element “Add to Cart”. The underlying implementation of WebArena will perform the clicking action, and the web page will update accordingly. This flexibility in element selection enables WebArena to support agents designed in various ways (*e.g.*, accepting input from different modalities) without compromising fair comparison metrics such as the number of steps taken.

2.4 User Roles Simulation

Users of the same website often have disparate experiences due to their distinct *roles*, *permissions*, and *interaction histories*. For instance, within an E-commerce CMS, a shop owner might possess full read and write permissions across all content, whereas an employee might only be granted write permissions for products but not for customer data. We aim to emulate this scenario by generating unique user profiles on each platform.

On the shopping site, we created a customer profile that has over 35 orders within a span of two years. On GitLab, we selected a user who maintains several popular open-source projects with numerous merge requests and issues. This user also manages a handful of personal projects privately. On Reddit, our chosen profile was a user who actively participates in discussions, with many posts and comments. Lastly, on our E-commerce CMS, we set up a user profile for a shop owner who has full read-and-write access to all system contents.

All users are automatically logged into their accounts using a pre-cached cookie. To our best knowledge, this is the first publicly available agent evaluation environment to implement such a characteristic. Existing literature typically operates under the assumption of universally identical user roles [27, 21, 7].

3 Benchmark Suite of Web-based Tasks

We provide a benchmark with 812 test examples on grounding high-level natural language instructions to interactions in WebArena. Each example comes with a metric to evaluate the functional correctness of the task execution. In this section, we first formally define the task of controlling an autonomous agent through natural language. Then we introduce the annotation process of our benchmark.

3.1 Controlling Agents through High-level Natural Language

The WebArena environment is denoted as \mathcal{E} with state space \mathcal{S} , action space \mathcal{A} (§2.3) and observation space \mathcal{O} (§2.2). The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}$ is deterministic, and it is defined by the underlying implementation of each website in the environment. Performing a task described by a natural language intent \mathbf{i} can be formulated as a partially observable Markov decision process (POMDP): at each time

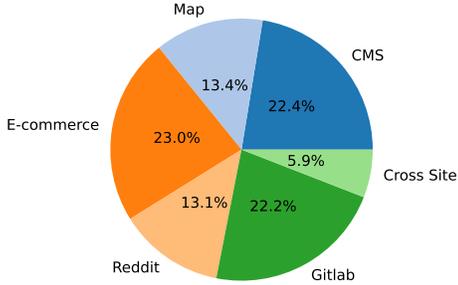


Figure 4: The intent distribution across different websites. Cross-site intents necessitate interacting with multiple websites. Notably, regardless of the website, all user intents require interactions with multiple web pages.

Category	Example
Information Seeking	When was the last time I bought shampoo
	Compare walking and driving time from AMC Waterfront to Randyland
Site Navigation	Checkout merge requests assigned to me
	Show me the ergonomic chair with the best rating
Content & Config	Post to ask “whether I need a car in NYC”
	Delete the reviews from the scammer Yoke

Figure 5: Example intents from three categories.

step t , an agent issues an action a_t given the partial observation o_t . Consequently, the action results in a new state s_{t+1} and its corresponding observation o_{t+1} . We propose a reward function $r(\mathbf{a}, \mathbf{s})$ to measure the success of a task execution, where \mathbf{a} represents the sequence of actions, and \mathbf{s} denotes all intermediate states. This reward function assesses if state transitions align with the expectations of the intents. For example, with an intent to place an order, it verifies whether an order has been placed. Additionally, it evaluates the accuracy of the agent’s actions, such as checking the correctness of the predicted answer.

3.2 Intent Collection

We focus on curating *realistic* intents to carry out *complex* and *creative* tasks within WebArena. To start with, our annotators were guided to spend a few minutes exploring the websites to familiarize themselves with the websites’ content and functionalities. As most of our websites are virtually identical to their open-web counterparts, despite having sampled data, most annotators can quickly comprehend the websites.

Next, we instructed the annotators to formulate intents based on the following criteria:

- (1) The intent should be *abstract* and *high-level*, implying that the task cannot be fulfilled with merely one or two actions. As an example, instead of “*click the science subreddit*”, we encouraged annotators to come up with something more complex like “*post a greeting message on science subreddit*”, which involves performing multiple actions.
- (2) The intent should be *creative*. Common tasks such as account creation can be easily thought of. We encouraged the annotators to add constraints (e.g., “*create a Reddit account identical to my GitLab one*”) to make the intents more unique.
- (3) The intent should be formulated as a *template* by making replaceable elements as variables. The annotators were also responsible for developing several instantiations for each variable. For example, the intent “*create a Reddit account identical to my GitLab one*” can be converted into “*create a {{site1}} account identical to my {{site2}} one*”, with an instantiation like “*{site1: Reddit, site2: GitLab}*” and another like “*{site1: GitLab, site2: OneStop-Shopping}*”. Notably, tasks derived from the same template can have distinct execution traces. The similarity resides primarily in the high-level semantics rather than the specific implementation.

We additionally provided a prompt for the annotators to use to prompt ChatGPT⁵ for inspiration. This prompt contains an overview of each website and instructs the model to describe potential tasks to be performed on these sites. Furthermore, we offered a curated list of examples for annotators to reference.

⁵<https://chat.openai.com/>

Intent Analysis In total, we curated 241 templates and 812 instantiated intents. On average, each template is instantiated to 3.3 examples. The intent distribution is shown in Figure 4.

Furthermore, we classify the intents into three primary categories with examples shown in Figure 5:

- (1) Information-seeking tasks: These are tasks where a textual response is expected. Importantly, the information-seeking tasks in WebArena often require navigation across multiple pages or focus on *user-centric* content. This makes them distinct from open-domain question-answering tasks [36, 17], which focus on querying general knowledge with a simple retrieval step. For instance, to answer “*When was the last time I bought the shampoo*”, an agent must traverse to the user’s purchase history, checking individual order details to identify the most recent shampoo purchase.
- (2) Site navigation tasks: This category is composed of tasks that require navigating through web pages using a variety of interactive elements such as search functions and links. The objective is often to locate specific information or navigate to a particular section of a site.
- (3) Content and configuration operation tasks: This category encapsulates tasks that require operating in the web environment to create, revise, or configure content or settings. This includes adjusting settings, managing accounts, performing online transactions, generating new web content, and modifying existing content. Examples range from updating a social media status or README file to conducting online purchases and configuring privacy settings.

3.3 Evaluation Annotation

Evaluating Information Seeking Tasks To measure the correctness of information-seeking tasks where a textual answer is expected, we provide the annotated answer a^* for each intent. The a^* is further compared with the predicted answer \hat{a} with one of the following scoring functions $r_{\text{info}}(\hat{a}, a^*)$.

First, we define `exact_match` where only \hat{a} that is identical with a^* will receive a score of one. This function is primarily applicable to those intent types whose responses follow a more standardized format, similar to the evaluation on question answering literature [26, 36].

Second, we create `must_include` where any \hat{a} containing a^* receives a score of one. This function is primarily used in scenarios where an unordered list of text is expected or where the emphasis of evaluation is on certain key concepts. In the second example in Table 2, we expect both the correct name and the email address to be presented, irrespective of the precise wording used to convey the answer.

Finally, we introduce `fuzzy_match` where we utilize a language model to assess whether \hat{a} accurately captures the key concepts presented in a^* . Specifically, in this work, we use `gpt-3.5-turbo-0613` to assess whether \hat{a} could entail each of the key concepts in a^* [6]. The corresponding prompt details are provided in Appendix A.3. The `fuzzy_match` function is applicable to situations where the key concepts are interrelated or where the format of the answer is diverse. For instance, in responding to “*Compare the time for walking and driving route from AMC Waterfront to Randyland*”, it is essential to ensure that driving time and walking time are accurately linked with the correct terms. The `fuzzy_match` function could also flexibly match the time “2h58min” with different forms such as “2 hour 58 minutes”, “2:58” and others.

Our evaluation methods emphasize the *recall* of key concepts within the reference answer rather than precision or overall answer similarity, as measured by metrics such as ROUGE or METEOR. This is mainly due to the fact that the questions in our dataset often require the inclusion of specific content, while the precise choice of terminology and phrasing is less critical in our context. By emphasizing recall, we attempt to ensure the helpfulness of the predicted answer, irrespective of its syntactic conformity to the reference. Existing works that measure answer recall in long-form QA share a similar spirit [13].

Evaluating Site Navigation and Content & Config Tasks The tasks in these categories require accessing web pages that meet certain conditions or performing operations that modify the underlying data storage of the respective websites. To assess these, we establish reward functions $r_{\text{prog}}(\mathbf{s})$ that programmatically examine the intermediate states \mathbf{s} within an execution trajectory to ascertain

Function	ID	Intent	Eval Implementation
$r_{\text{info}}(a^*, \hat{a})$	1	Tell me the name of the customer who has the most cancellations in the history	<code>exact_match(\hat{a}, "Samantha Jones")</code>
	2	Find the customer name and email with phone number 8015551212	<code>must_include(\hat{a}, "Sean Miller")</code> <code>must_include(\hat{a}, "sean@gmail.com")</code>
	3	Compare walking and driving time from AMC Waterfront to Randyland	<code>fuzzy_match(\hat{a}, "Walking: 2h58min")</code> <code>fuzzy_match(\hat{a}, "Driving: 21min")</code>
$r_{\text{prog}}(s)$	4	Checkout merge requests assigned to me	<code>url = locate_last_url(s)</code> <code>exact_match(URL, "gitlab.com/merge_requests?assignee_username=byteblaze")</code>
	5	Post to ask "whether I need a car in NYC"	<code>url = locate_latest_post_url(s)</code> <code>body = locate_latest_post_body(s)</code> <code>must_include(URL, "/f/nyc")</code> <code>must_include(body, "whether I need a car in NYC")</code>

Table 2: We introduce two evaluation approaches. r_{info} (top) measures the correctness of performing information-seeking tasks. It compares the predicted answer \hat{a} with the annotated reference a^* with three implementations. r_{prog} (bottom) programmatically checks whether the intermediate states during the executions possess the anticipated properties specified by the intent.

whether the outcome aligns with the intended result. These intermediate states are often the underlying databases of the websites, the status, and the content of a web page at each step of the execution.

Evaluation of each instance involves two primary components. First, we provide a `locator`, tasked with retrieving the critical content pertinent to each intent. The implementation of this locator may vary from a database query, a website-supported API call, to a JavaScript element selection on the relevant web page, depending on implementation feasibility. For example, in evaluating the intent of the fifth example in Table 2, the evaluation process first obtains the URL of the latest post by examining the last state in the state sequence s . Then it navigates to the corresponding post page and runs Javascript code `document.querySelector('.submission__inner').outerText` to obtain the content of the post.

Subsequently, we annotate keywords that need to exist within the located content. For example, the evaluation verifies if the post is correctly posted in the "nyc" subreddit by examining the URL of the post and if the post contains the requested content by examining the post content. We reuse the `exact_match` and `must_include` functions from information-seeking tasks for this purpose.

Unachievable Tasks Due to constraints such as inadequate evidence, user permissions (§2.4), or the absence of necessary functional support in the website, some tasks ($\sim 4\%$) in our benchmark are not possible to complete. For instance, fulfilling an intent like *"Tell me the contact number of OneStopShop"* is impracticable in `WebArena`, given that the website does not provide such contact information. We label such instances as "N/A" and expect that an agent should produce an equivalent response. These examples allow us to assess an agent’s ability to avoid making unfounded claims and its adherence to factual accuracy.

Annotation Process The intents were contributed by the authors following the annotation guideline in §3.2. Every author has extensive experience with web-based tasks. The reference answers of the information-seeking tasks were curated by the authors and an external annotator. To ensure consistency and accuracy, each question was annotated twice. If the two annotators disagreed, a third annotator finalized the annotation. The programs to evaluate the remaining examples were contributed by three of the authors who are proficient in JavaScript programming. Difficult tasks were often discussed collectively to ensure the correctness of the annotation. The annotation required the annotator to undertake the full execution and scrutinize the intermediate states.

Agent	Model	Success Rate
Direct	gpt-3.5	7.14
Reasoning	gpt-3.5	7.38
Reasoning	gpt-4	10.59

Table 3: The end-to-end task success rate (%) on WebArena. The direct agent directly predicts the next action given the observation, and the reasoning agent performs the reasoning on the observation, progress, and others *before* the action prediction.

4 Baseline Web Agents

Prompts We design two LLM-prompt-based web agents as our baselines where the input observations and the predicted actions are all in text: (1) *Direct Agent*: the agent takes as input the observation and directly predict the next action. (2) *Reasoning Agent*: the agent first performs a sequence of reasoning steps in text, then issues the next action [34, 39]. The reasoning often involves interpreting the intent, understanding the current state, and estimating the task progress, which could aid task completion. In both prompts, we first elaborate the definition of web-based tasks, then provide a full list of available actions and their explanations. We also include in the prompt a few notes on avoiding common failures, which we observed during our initial experiments on simple examples. For instance, we found that the agent tends to generate a sequence of actions instead of one action at a time, and therefore provide a hint of “*You should only issue one action at a time.*” Finally, we provide two demonstrations of observation \rightarrow action pairs in the prompt. The full prompts can be found in Appendix A.4.

Observations & Actions Primarily, we use an accessibility tree with element IDs as the observation space, as shown in the rightmost example of Figure 3. Although we provide careful instructions on avoiding common failures, the actions are not flawless. For example, the agent frequently issues the actions in the wrong format, which results in parsing failures. To address this, we incorporate these error messages in the observation, thereby enabling the agent to make dynamic adjustments and learn from the failures. The agent can identify which element to interact with by the ID of the element. For instance, the agent can issue `click [1582]` to click the “Add to Cart” button. As part of our future research, we intend to explore the possibility of multi-modal agents that accept image-based observations and perform pixel-based actions.

Configurations We experiment with `gpt-3.5-turbo-0613` and `gpt-4-0613` with a temperature of 1.0^6 and a top- p parameter of 0.9. The maximum number of state transitions is set to 30. We halt execution if the same action is repeated more than three times on the same observation or if the agent generates three consecutive invalid actions. These situations typically indicate a high likelihood of execution failure and hence warrant early termination.

5 Results

5.1 Main Results

The main results are shown in Table 3. The reasoning agent powered by GPT-4 achieves a relatively modest end-to-end task success rate of 10.63%. When the same reasoning agent is powered by GPT-3.5, the success rate drops to 7.38%. Interestingly, the reasoning agent that explicitly engages in a reasoning process prior to providing the response does not exhibit a marked performance improvement when compared to its simpler counterpart that directly predicts the subsequent action without the intermediate reasoning step. The GPT-3.5 reasoning agent outperforms the direct agent only by successfully completing a marginal number of examples. These results underline the inherent challenges and complexities of executing tasks that span long horizons, particularly in realistic environments as WebArena.

⁶We use a high temperature to encourage exploration as we observed frequent early termination during agent development. More details are in §5.2.

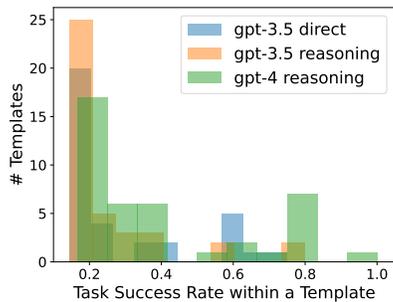


Figure 6: Distribution of success rate within a template on templates with at least one successful execution

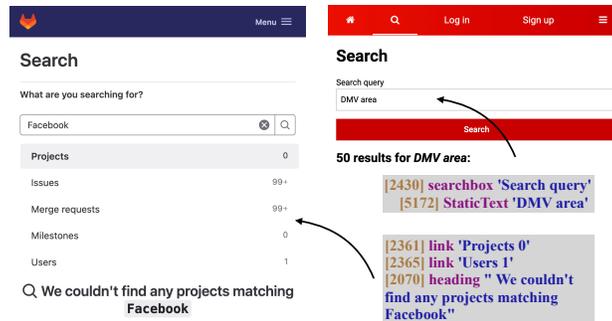


Figure 7: Two examples where the GPT-4 agent failed, along with their screenshot and the accessibility tree of the relevant sections (grey). On the left, the agent fails to proceed to the “Users” section; on the right, the agent repeats entering the same search query.

Consistency of Performing Tasks From the Same Template Tasks derived from the same template typically share similar reasoning and planning procedures, albeit manifesting distinct executions and observations. We visualize the task success rate for templates with at least one successful completion in Figure 6. Of the 41 such templates, the GPT-4 agent manages to achieve a 100% task success rate on only one template, while GPT-3.5 agents do not achieve full task completion for any of the templates. In several cases, the agents are only able to complete one task variation among others. The discrepancy in performance arises from two reasons. First, the complexity of tasks can vary, even within the same template. For instance, the agent is able to correctly perform “*Fork metaseq*” while it fails to “*Fork all repos from Facebook*”. The latter task necessitates more repetitive operations, therefore increasing its complexity. Second, the prediction may have larger variances due to the high temperature we use in the experiment. Nevertheless, these results suggest the current instability of agents in handling complex tasks. More refined techniques, such as procedural skill libraries [41, 32], may enhance the performance as they could enable the *reuse* of successful past experiences and therefore lead to more consistent outcomes.

5.2 Error Analysis

We perform an analysis by looking into the recorded execution trajectories from the GPT-4 reasoning agent and identify the following common failure modes.

Incorrect Infeasibility Determination A prevalent issue with GPT-4 agents is the early stop of tasks due to a conclusion of non-achievability. This issue often manifests as the agent outputting a `stop [N/A]` action. This behavior is observed in instances where the GPT-4 agent erroneously identifies approximately 54.9% of feasible tasks (428 tasks) as impossible. For instance, as on the left of Figure 7, to achieve the goal of “*Fork all repos from Facebook*”, the agent first correctly performs the search of Facebook. However, the default search lists the possible projects named Facebook, while the agent needs to further click the user column to find the Facebook account. The agent decides to stop because it thinks that “It appears that there are no available projects matching Facebook. Hence, it is not possible to fork any Facebook repositories from GitLab.”.

Observation Bias Realistic websites frequently present information on similar topics across various sections to ensure optimal user accessibility. However, a GPT-4 agent often demonstrates a tendency to latch onto the first related piece of information it encounters without sufficiently verifying its relevance or accuracy. For instance, the homepage of the E-Commerce CMS displays the best-selling items based on *recent purchases*, while historical best-seller data is typically accessed via a separate report. Presented with the task of “*What is the top-1 best-selling product in 2022*”, the GPT-4 agent defaults to leveraging the readily available information on the homepage, bypassing the necessary step of generating the report to obtain the accurate data.

Benchmark	Dynamic Interaction?	Realistic Environment?	Diverse Human Tasks?	Functional Correctness?
Mind2Web [7]	✗	✓	✓	✗
Form/QAWoB [27]	✗	✓	✓	✗
MiniWoB++ [21]	✓	✗	✗	✓
Webshop [37]	✓	✗	✗	✓
ALFRED [29]	✓	✗	✗	✓
VirtualHome [25]	✗	✗	✓	✗
AndroidEnv [31]	✓	✓	✗	✗
WebArena	✓	✓	✓	✓

Table 4: The comparison between our benchmark and existing benchmarks on grounding natural language instructions to concrete executions. Our benchmark is implemented in our fully interactable highly-realistic environment. It features diverse tasks humans may encounter in their daily routines. We design evaluation metrics to assess the functional correctness of task executions.

Failures in Observation Interpretation Interestingly, while a GPT-4 agent is capable of summarizing the observations, it occasionally overlooks more granular information, such as the previously entered input. As in the right-hand example of Figure 7, [5172] `StaticText` indicates that the search term “DMV area” has already been entered. However, the agent disregards this detail and continuously issues the command `type [2430] [DMV area]` until it reaches the maximum step limit. Furthermore, the agent often neglects the previous action information that is provided alongside the observation.

We hypothesize that these observed failures are related to the current pretraining and supervised fine-tuning on dialogues employed in GPT models [24]. These models are primarily trained to execute instructions given *immediate* observations (*i.e.*, the dialogue history); thereby, they may exhibit a lack of explorations. Furthermore, in dialogue scenarios, subtle differences in NL expressions often have less impact on the overall conversation. As a result, models may tend to overlook minor variations in their observations.

6 Related Work

Benchmarks for Controlling Agents through Natural Language Using computer and smartphone platforms as an environment to study controlling agents through natural language has been studied in the literature [3, 27, 21, 31, 7, 19, 35]. However, achieving a balance between *functionality*, *authenticity*, and *support for environmental dynamics* is challenging. Existing works often necessitate a compromise between these aspects, as shown in Table 4. Some works [27, 7] depend on a static cache of states during data collection, which constrains the agents’ explorations on unfamiliar states and assess functional correctness. Other works [37, 21] adopt an overly simplistic representation of real-world environmental complexities, which narrows the breadth and depth of feasible tasks, restricting the potential for more complex or varied task execution. While AndroidEnv [31] aim to emulate an independent Android setup, they fall short in providing human high-level intents and task evaluations. Such simplifications are observed in synthetic environments such as AI2THOR [15, 29] and VirtualHome [25]. Finally, gaming environments [8, 16] are constrained by their inherent design principles which diverge from human objectives (*e.g.*, agent survives for n days). WebArena addresses these limitations by introducing a self-hostable web environment conducive to performing realistic day-to-day tasks. The feature of self-hosting permits comprehensive end-to-end task execution evaluations, providing a more holistic and robust benchmark suit for future agent developments.

Natural Language Command and Control Executing complex tasks described in a high-level natural language requires several capabilities, including hierarchical planning, state tracking, error recovery, and others. Large Language Models (LLMs) have been observed to break down high-level assignments into more manageable sub-tasks [11]. To link these broken-down plans to tangible executions and encourage the reuse of skills, several studies advocate for the representation of task executions as programs [41, 20, 32, 9]. The introduction of structured planning and backtracking methodologies, such as the tree-of-thoughts [38, 22], further allow for a more structured task planning,

as well as the capacity to reconsider previous decisions. Moreover, it is crucial for LLMs to include built-in failure recovery and self-correction mechanisms to increase their robustness [28, 14]. However, while these approaches have shown promise, their applications have largely been tested in relatively simple settings. For instance, in environments where an erroneous action can be reversed by a single undo command. However, actions in complex environments such as WebArena may not be easily reversible. For example, canceling an incorrectly placed order requires navigating through a series of complex cancellation steps rather than just executing a simple undo command.

7 Conclusion

We present WebArena, a highly-realistic, standalone, and reproducible web environment designed for the development and testing of autonomous agents. WebArena includes fully functional web applications and genuine data from four major categories, providing a realistic platform for agent interaction. It further supports a wide range of tools and external knowledge bases, fostering a focus on human-like problem-solving. Additionally, we curate a comprehensive benchmark consisting of 812 examples that focus on translating high-level natural language intents into specific web interactions. We also offer metrics to programmatically ascertain whether tasks have been completed according to the desired objectives.

When tested with the state-of-the-art GPT-4 language model, the best-performing agent achieved a modest end-to-end task success rate of 10.59%. Our error analysis suggests that while the agent could steer the interactions toward the task completion, there are several areas for improvement. Common errors include an overestimation of task infeasibility and an over-reliance on readily available context at the expense of necessary exploration. Additionally, the agent often fails to distinguish nuances in observations, leading to repeated actions that ultimately result in task termination.

These findings underscore the need for future research to focus on enhancing the robustness and efficacy of autonomous agents within the WebArena environment.

References

- [1] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3674–3683. IEEE Computer Society, 2018.
- [2] Yonatan Bisk, Jan Buys, Karl Pichotta, and Yejin Choi. Benchmarking hierarchical script knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4077–4085, Minneapolis, Minnesota, 2019. Association for Computational Linguistics.
- [3] S.R.K. Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90, Suntec, Singapore, 2009. Association for Computational Linguistics.
- [4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- [5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *ArXiv preprint*, abs/2107.03374, 2021.
- [6] Dorottya Demszky, Kelvin Guu, and Percy Liang. Transforming question answering datasets into natural language inference datasets. *ArXiv preprint*, abs/1809.02922, 2018.
- [7] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023.

- [8] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.
- [9] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023.
- [10] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. IQA: visual question answering in interactive environments. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4089–4098. IEEE Computer Society, 2018.
- [11] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.
- [12] Yacine Jernite, Kavya Srinet, Jonathan Gray, and Arthur Szlam. CraftAssist Instruction Parsing: Semantic Parsing for a Minecraft Assistant. *ArXiv preprint*, abs/1905.01978, 2019.
- [13] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *ArXiv preprint*, abs/2305.06983, 2023.
- [14] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *ArXiv preprint*, abs/2303.17491, 2023.
- [15] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.
- [16] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In Hugo Larochelle, Marc’ Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [17] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.
- [18] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.
- [19] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile UI action sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8198–8210, Online, 2020. Association for Computational Linguistics.
- [20] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.
- [21] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- [22] Jieyi Long. Large language model guided tree-of-thought. *ArXiv preprint*, abs/2305.08291, 2023.

- [23] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016.
- [24] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [25] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8494–8502. IEEE Computer Society, 2018.
- [26] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, 2016. Association for Computational Linguistics.
- [27] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 2017.
- [28] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *ArXiv preprint*, abs/2303.11366, 2023.
- [29] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10737–10746. IEEE, 2020.
- [30] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [31] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *ArXiv preprint*, abs/2105.13231, 2021.
- [32] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *ArXiv preprint*, abs/2305.16291, 2023.
- [33] Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. Execution-based evaluation for open-domain code generation. *ArXiv preprint*, abs/2212.10481, 2022.
- [34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.
- [35] Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna, Larry Heck, James Landay, and Monica Lam. Grounding open-domain instructions to automate web support tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1022–1032, Online, 2021. Association for Computational Linguistics.
- [36] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question

- answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, 2018. Association for Computational Linguistics.
- [37] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *ArXiv preprint*, abs/2207.01206, 2022.
- [38] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv preprint*, abs/2305.10601, 2023.
- [39] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv preprint*, abs/2210.03629, 2022.
- [40] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. arxiv 2017. *ArXiv preprint*, abs/1709.00103, 2017.
- [41] Shuyan Zhou, Pengcheng Yin, and Graham Neubig. Hierarchical control of situated agents through natural language. *ArXiv preprint*, abs/2109.08214, 2021.

A Appendix

A.1 Website Implementation

Given the selected websites described in §2.1, we make the best attempt to reproduce the functionality of commonly used sites in a reproducible way. To achieve this, we utilized open-source frameworks for the development of the websites across various categories and imported data from their real-world counterparts. For the E-commerce category, we constructed a shopping website with approximately 90k products, including the prices, options, detailed product descriptions, images, and reviews, spanning over 300 product categories. This website is developed using Adobe Magento, an open-source e-commerce platform⁷. Data resources were obtained from data from actual online sites, such as that included in the Webshop data dump[37]. As for the social forum platform, we deployed an open-source software Postmill⁸, the open-sourced counterpart of Reddit⁹. We sampled from the top 50 subreddits¹⁰. We then manually selected many subreddit for northeast US cities as well as subreddit for machine learning and deep learning-related topics. This manual selection encourages cross-website tasks such as seeking information related to the northeast US on both Reddit and the map. In total, we have 95 subreddits, 127390 posts, and 661781 users. For the collaborative software development platform, we choose GitLab¹¹. We heuristically simulate the code repository characteristics by sampling at least ten repositories for every programming language: 80% of them are sampled from the set of top 90 percentile wrt stars repos using a discrete probability distribution weighted proportional to their number of stars; the remaining are sampled from the bottom ten percentile set using similar weighted distribution. This is done to ensure fair representation of repos of all kinds, from popular projects with many issues and pull requests to small personal projects. In total, we have 300 repositories and more than 1000 accounts with at least one commit to a repository. For the content management system, we adapted Adobe Magento’s admin portal, deploying the sample data provided in the official guide. We employ OpenStreetMap¹² for map service implementation, confining our focus to the northeast US region due to data storage constraints. We implement a calculator and a scratchpad ourselves.

Lastly, we configure the knowledge resources as individual websites, complemented with search functionality for efficient information retrieval. Specifically, we utilize Kiwix¹³ to host an offline version of English Wikipedia with a knowledge cutoff of May 2023. The user manuals for GitLab and Adobe Commerce Merchant documentation are scraped from the official websites.

A.2 Environment Delivery and Reset

One goal for our evaluation environment is ease of use and reproducibility. As a result, we deploy our websites in separate Docker images¹⁴, one per website. The Docker images are fully self-contained with all the code of the website, database, as well as any other software dependencies. They also do not rely on external volume mounts to function, as the data of the websites are also part of the docker image. This way, the image is easy to distribution containing all the pre-populated websites for reproducible evaluation. End users can download our packaged Docker images and run them on their systems and re-deploy the exact websites together with the data used in our benchmarks for their local benchmarking.

Since some evaluation cases may require the agent to modify the data contained in the website, *e.g.*, creating a new user, deleting a post, etc., it is crucial to be able to easily reset the website environment to its initial state. With Docker images, the users could stop and delete the currently running containers for that website and start the container from our original image again to fully reset the environment to the initial state. Depending on the website, this process may take from a few seconds to one minute. However, not all evaluation cases would require an environment reset, as

⁷<https://github.com/magento/magento2>

⁸<https://postmill.xyz/>

⁹<https://www.reddit.com/>

¹⁰<https://redditlist.com/sfw.html>

¹¹<https://gitlab.com/gitlab-org/gitlab>

¹²<https://www.openstreetmap.org/>

¹³<https://www.kiwix.org/en/>

¹⁴<https://www.docker.com/>

many of the intents are information gathering and are read-only for the website data. Also, combined with the inference time cost for the agent LLMs, we argue that this environment reset method, through restarting Docker containers from the original images, will have a non-negligible but small impact on evaluation time.

A.3 Prompt for `fuzzy_match`

Given the statement “`{{prediction}}`”, would it be correct to infer “`{{reference}}`”? Yes or No.

A.4 The Prompts of the Baseline Web Agents

The system message of the reasoning agent for both GPT-3.5 and GPT-4 is in Figure 8, and two examples are in Figure 9. The system message of the direct agent for GPT-3.5 is in Figure 10 and the two examples are in Figure 11.

A.5 A Successful Trajectory

A successful trajectory is shown in Figure 12.

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.
The previous action: This is the action you just performed. It may be helpful to track your progress.

The actions you can perform fall into several categories:

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).
`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 [answer]` : Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

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.
<http://homepage.com/password.html> lists all the account names and passwords for the websites. You can use them to log in to the websites.

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 with a "In summary, the next action I will perform is" phrase, followed by action inside ``````. For example, "In summary, the next action I will perform is `` `click [1234]` ``".
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop. ""

Figure 8: The system message of the reasoning agent. This message has a general explanation of the task, the available actions, and some notes on avoiding common failures.

example_user
OBSERVATION:
[1744] link 'HP CB782A#ABA 640 Inkjet Fax Machine (Renewed)'
[1749] StaticText '\$279.49'
[1757] button 'Add to Cart'
[1760] button 'Add to Wish List'
[1761] button 'Add to Compare'
URL: <http://onestopmarket.com/office-products/office-electronics.html>
OBJECTIVE: What is the price of HP Inkjet Fax Machine
PREVIOUS ACTION: None

example_assistant
Let's think step-by-step. This page lists 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]` ``

example_user
OBSERVATION:
[164] textbox 'Search' focused: True required: False
[171] button 'Go'
[174] link 'Find directions between two points'
[212] heading 'Search Results'
[216] button 'Close'
URL: <http://openstreetmap.org>
OBJECTIVE: Show me the restaurants near CMU
PREVIOUS ACTION: None

example_assistant
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]` ``

Figure 9: The two examples provided as `example_user` and `example_assistant` for the reasoning agent. Before issuing the action, the agent first perform reasoning.

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.

The previous action: This is the action you just performed. It may be helpful to track your progress.

The actions you can perform fall into several categories:

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).
- ``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 [answer]``: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

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.

<http://homepage.com/password.html> lists all the account name and password for the websites. You can use them to log in to the websites.

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

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. Generate the action in the correct format. Always put the action inside a pair of `` ` ` ``. For example, `` ` `click [1234]` ` ` ``
4. Issue stop action when you think you have achieved the objective. Don't generate anything after `stop."`

Figure 10: The system message of the direct agent. This message has the general explanation of the task, the available actions and some notes on avoiding common failures.

```

example_user
OBSERVATION:
[1744] link 'HP CB782A#ABA 640 Inkjet Fax Machine (Renewed)'
[1749] StaticText '$279.49'
[1757] button 'Add to Cart'
[1760] button 'Add to Wish List'
[1761] button 'Add to Compare'
URL: http://onestopmarket.com/office-products/office-electronics.html
OBJECTIVE: What is the price of HP Inkjet Fax Machine
PREVIOUS ACTION: None

example_assistant
```stop [$279.49]```

example_user
OBSERVATION:
[164] textbox 'Search' focused: True required: False
[171] button 'Go'
[174] link 'Find directions between two points'
[212] heading 'Search Results'
[216] button 'Close'
URL: http://openstreetmap.org
OBJECTIVE: Show me the restaurants near CMU
PREVIOUS ACTION: None

example_assistant
```type [164] [restaurants near CMU] [1]```

```

Figure 11: The two examples provided as `example_user` and `example_assistant` for the direct agent. The agent directly emits the next action given the observation.

One Stop Market

Product Showcases



Let's think step-by-step. To check the status of an order, I need to go to the account page. The link to the account page is [1256]. In summary, the next action I will perform is `click [1256]`.

[Compare Products](#)

Let's think step-by-step. The user is on the 'My Account' page of the website and wants to check the status of their latest order. To find this information, I will need to navigate to the 'My Orders' section, which can be accessed by clicking the link with the ID [4165]. In summary, the next action I will perform is `click [4165]`.

One Stop Market

Search entire store here...

Advanced Search

Beauty & Personal Care - Sports & Outdoors - Clothing, Shoes & Jewelry - Home & Kitchen - Office Products - Tools & Home Improvement -

Health & Household - Patio, Lawn & Garden - Electronics - Cell Phones & Accessories - Video Games - Grocery & Gourmet Food -

My Account

My Orders

My Downloadable Products

My Wish List

Address Book

Account Information

Stored Payment Methods

My Product Reviews

Newsletter Subscriptions

Compare Products

Order #	Date	Order Total	Status	Action
00000170	5/17/23	\$365.42	Canceled	View Order Reorder
00000189	5/2/23	\$754.99	Pending	View Order Reorder
00000188	5/2/23	\$2,004.99	Pending	View Order Reorder
00000187	5/2/23	\$1,004.99	Pending	View Order Reorder
00000180	3/11/23	\$65.32	Complete	View Order Reorder
00000166	3/10/23	\$17.99	Complete	View Order Reorder
00000161	2/27/23	\$762.18	Complete	View Order Reorder
00000156	2/24/23	\$231.54	Canceled	View Order Reorder
00000158	2/11/23	\$174.99	Canceled	View Order Reorder

Let's think step-by-step. The latest order is the first one listed in the table. The ID of the link to view this order's details is [6342]. I will click this link to find out more about the order's status and expected arrival date. In summary, the next action I will perform is `click [6342]`

One Stop Market

Search entire store here...

Advanced Search

Beauty & Personal Care - Sports & Outdoors - Clothing, Shoes & Jewelry - Home & Kitchen - Office Products - Tools & Home Improvement -

Health & Household - Patio, Lawn & Garden - Electronics - Cell Phones & Accessories - Video Games - Grocery & Gourmet Food -

My Account

My Orders

My Downloadable Products

My Wish List

Address Book

Account Information

Stored Payment Methods

My Product Reviews

Newsletter Subscriptions

Compare Products

Order # 00000170

Canceled
Order Date: May 17, 2023

[Reorder](#) [Print Order](#)

Items Ordered

Product Name	SKU	Price	Qty	Subtotal
Uttermost Volterra Crackled Taupe-Gray Ceramic Table Lamp	B087QSCXGT	\$250.80	Ordered: 1	\$250.80
NOZE Rustic Coat Rack Wall Mounted Shelf with 4 Hooks, Hanging Entryway Organizer for Mug Coffee Cup, Holding Solid Wooden Shelf with 2 Baskets for Kitchen Living Room, Bathroom and Bedroom	B08JLHHC6	\$40.99	Ordered: 1	\$40.99
Plus Size Lingerie for Women Sexy for Sex Naughty Eyelash Lace Bodysuit Naughty	B09LQTV3RX	\$58.63	Ordered: 1	\$58.63

Let's think step-by-step. This page is the order status page for Order # 00000170. It shows that the order status is "Canceled". However, it does not provide information about the delivery time. Since the order is canceled, it will not arrive. 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 [The order is canceled and will not arrive.]`

Figure 12: The trajectory of the GPT-4 reasoning agent on performing the task of “Tell me the status of my latest order and when will it arrive”. The agent uses accessibility tree as the web page observations, however, for readability, we display the procedure using screenshots. In each screenshot, the element predicted for interaction by the agent is highlighted by a red arrow. The agent’s reasoning process and predicted actions are beneath each screenshot.