R2D2: Remembering, Replaying and Dynamic Decision Making with a Reflective Agentic Memory

Anonymous ACL submission

Abstract

The proliferation of web agents necessitates advanced navigation and interaction strategies within complex web environments. Current models often struggle with efficient navigation and action execution due to limited visibility and understanding of web structures. Our proposed **R2D2** framework addresses these challenges by integrating two paradigms: Remember and Reflect. The Remember paradigm uses a replay buffer that aids agents in reconstructing the web environment dynamically, thus enabling the formulation of a detailed "map" of previously visited pages. This helps in reducing navigational errors and optimizing the decision-making process during web interactions. Conversely, the Reflect paradigm allows agents to learn from past mistakes by providing a mechanism for error analysis and strategy refinement, enhancing overall task performance. We evaluate **R2D2** using the WebArena benchmark, demonstrating substantial improvements over existing methods, including a 50% reduction in navigation errors and a threefold increase in task completion rates. Our findings suggest that a combination of memory-enhanced navigation and reflective learning promisingly advances the capabilities of web agents, potentially benefiting various applications such as automated customer service and personal digital assistants.

1 Introduction

Web agents——autonomous AI agents designed to navigate and perform natural language-described tasks within web environments——have become increasingly integral to applications such as online customer service (Huang et al., 2024a), automated data retrieval (Huang et al., 2024b), and personalized digital assistants.[1]. These agents interact with complex web interfaces to execute user-described tasks, often emulating human actions like clicking buttons, filling forms, and extracting information (Shi et al., 2017; Liu et al., 2018; Yao et al., 2022; Zhou et al., 2023). Despite recent advancements in web agents' capabilities, a persistent challenge remains: agents frequently fail to navigate effectively within intricate web environments. As illustrated in Fig. 1, the successful resolution of a user query usually requires a long series of actions.

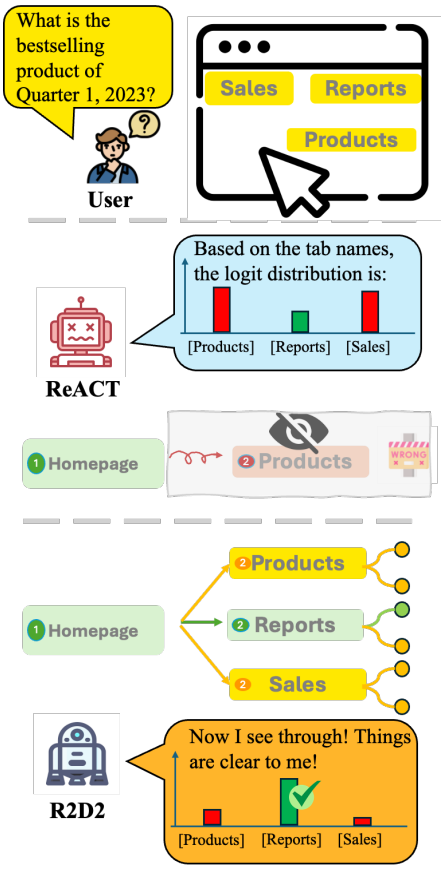

Figure 1: Traditional methodologies conceptualize web navigation within the framework of an Unknown MDP. The ReACT agent operates under high uncertainty due to incomplete information regarding the outcomes of their actions, leading to erroneous navigational paths, impeding effective task resolution. **R2D2** transforms the task into a Known MDP, improving robustness.

[1] https://www.anthropic.com/news/3-5-models-and-computer-use

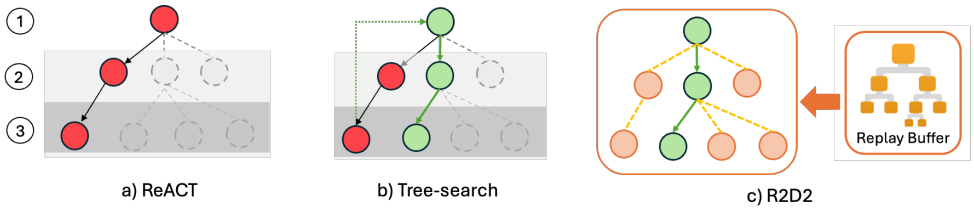

Figure 2: This diagram represents various approaches for web agents framed as a search problem, where each node symbolizes a webpage. (a) Reactive: The agent chooses the best immediate actions without any proactive strategy. (b) Tree-search with reflections: the agent investigates different routes by actively navigating websites and allows for reversing direction (shown by dashed arrows). Both a) and b) approaches are Unknown-MDP-based. At each timestep, agents' observation space is constrained, which typically results in suboptimal or inefficient results. c) **R2D2**: Our proposed framework constructs the search space, leveraging stored state information from a replay buffer. By transforming the task into a Known MDP, **R2D2** enhances agents' ability to navigate and interact with web interfaces.

The fundamental challenges associated with previous methodologies are twofold: first, these approaches model web navigation as an Unknown Markov Decision Process (MDP), wherein the agent has limited visibility into the consequences of its actions, often leading to suboptimal performance outcomes. Second, prior methods engage in complex reasoning during the inference phase while observing a stream of experiences, and in their simplest forms, they discard incoming trajectories immediately after a single episode. This rapid forgetting of potentially valuable experiences impedes the agent's capacity to leverage useful information for future decision-making.

Meanwhile, a primary obstacle in enhancing web agent performance lies in navigation-related failures, which account for approximately 60% of their operational errors (as illustrated in Fig. 6). These failures occur when agents become disoriented within the web environment, preventing them from reaching the target webpages necessary to execute desired tasks. Such navigation inefficiencies significantly hinder the overall effectiveness of web agents. The remaining 40% of errors stem from execution failures and edge cases, where agents either misinterpret user intentions or mishandle specific web elements.

Inspired by cognitive studies showing that humans excel at complex tasks by iteratively refining strategies based on feedback (Palenciano et al., 2021; Zenkri et al., 2024), as well as by approaches in robotics for structured exploration of unfamiliar spaces (Thrun, 2002), we propose the **R2D2** (**R**emembering, **R**eflecting, and **D**ynamic **D**ecision Making) framework that enhances both navigation and task execution for web agents. Our method transforms the task from Unknown-MDP into a Known MDP by introducing the Remember Paradigm. It leverages a structured replay buffer of the agent's experiences that guides the agent to more promising avenues (Blundell et al., 2016; Schaul et al., 2016; Parisotto and Salakhutdinov, 2017; Savinov et al., 2018). At a high level, our approach enables the agent to record and recall previously visited pages—essentially constructing a dynamic map of the web environment—and then leverage this knowledge to refine its strategies. By converting the agent's experience into a well-organized search space, we empower it to identify reliable navigation routes to target resources rather than re-deriving them from scratch during inference. This reduces the computational overhead at test time and helps avoid repetitive or unproductive exploration.

To enable continual improvement based on both successes and failures, **R2D2** incorporates the Reflect Paradigm. Previous efforts in this domain often focus narrowly on immediate, execution-level errors and struggle with more pervasive navigation challenges (Shinn et al., 2023; Pan et al., 2024; Wang et al., 2024a). In contrast, our method leverages the refined, known search space described earlier to minimize navigational missteps, allowing the reflection mechanism to operate more effectively on remaining execution problems. By reducing the burden of basic wayfinding, the agent's reflective capabilities can more efficiently identify and correct subtle issues, ultimately leading to a higher overall success rate on complex web tasks.

Our proposed method diverges from traditional techniques by providing a more comprehensive and structured representation of the agent's historical experiences. Instead of simply recalling past states (Agashe et al., 2024) or relying on on-the-fly reasoning (Koh et al., 2024; Zhou et al., 2024), our approach organizes the agent's accumulated expe-

riences into a coherent and reusable resource that effectively guides future decisions. We evaluate our approach using the WEBARENA benchmark (Zhou et al., 2023), where it achieves substantial gains compared to baseline models, including approximately a 50% reduction in navigation errors and a threefold increase in overall task completion rates. Moreover, **R2D2** outperforms state-of-the-art methods by 17%, thereby demonstrating a more robust and informed capability for executing complex web-based tasks.

2 Related Works

Enhancing Web Agents. Existing research has illuminated that language models, without intervention, struggle to express linguistic intent in formal instruction that can control an extra-linguistic environment, such as web site navigation (Shi et al., 2017; Liu et al., 2018; Yao et al., 2022; Zhou et al., 2023; Deng et al., 2023). This inadequacy stems primarily from the intrinsic challenges associated with perception, strategic planning, and task execution in the intricate web environments.

To mitigate these challenges, enhancements to web agents have been categorized into two following principal strategies. (1) *Perception Alignment*: This strategy aims to augment agents' capabilities in interpreting graphical user interface elements by integrating multimodal data from webpages, enhancing both textual and visual comprehension (Gou et al., 2024; Anonymous, 2024). (2) *Post-hoc Reflection*: Studies indicate that enabling agents to engage in reflective practices post interaction can facilitate learning from historical trajectories, thereby improving future task executions (Shinn et al., 2023; Song et al., 2024; Pan et al., 2024; Wang et al., 2024a). (3) *Online Search Algorithms*: This involves the adoption of sophisticated search algorithms, including Monte Carlo Tree Search and other tree-based exploration methods, integrated with high-level planning driven by LLM-derived knowledge (Zhang et al., 2024; Koh et al., 2024; Meng et al., 2024). Furthermore, Gu et al. 2024 discusses speculative planning that leverages simulations of world models.

Despite these enhancements, the performance of current web agents are constrained by the assumptions of Unknown MDP, where the potential outcomes of actions are not available. In contrast, this paper proposes a novel approach where we reconstruct the web environment's structure based on the agents' exploratory actions, thereby furnishing them with outcome information crucial for making informed and grounded decisions.

Continual Exploration of the Agentic Environment. Tasking agents to explore an unknown environment has been an active research direction in the community (Brohan et al., 2023). Recent studies have focused on how agents abstract experiences into actionable skills, a development that is becoming increasingly central to advancements in this field (Wang et al., 2023a,b; Liu et al., 2024). Within the domain of web-based agents, these skills are often conceptualized as workflows. Sodhi et al. (2024) have introduced a novel framework that leverages human-engineered workflows to compose policies to tackle web tasks. Although improving agents' performance, manually crafting workflows can be a tedious process.

Unlike previous strategies that rely solely on high-quality successful trajectories or hand-crafted workflows, **R2D2** introduces a two-part mechanism that continuously learns from the full range of agent experiences, including failed attempts. **R2D2** moves beyond the limitations of purely Unknown-MDP-based assumptions and handcrafted workflows, resulting in more informed, robust decision-making and improved overall performance.

3 Method

In this section, we present **R2D2**, a framework for tackling complex web navigation tasks by integrating two paradigms: Remember and Reflect. The Remember paradigm constructs a structured replay buffer from past observations(§3.2), while the Reflect paradigm diagnoses and corrects errors in failed trajectories(§3.3). We then introduce mechanisms of the reflective memory (§3.4). Finally, we illustrate how these paradigms interact to improve the agent's performance through retrieval and in-context learning demonstrations (§3.5).

3.1 Method Overview

Given a user's task query q and an initial observation o_0 from the environment, the agent must produce a sequence of actions to address q. We define an *episode* as the process where the agent starts from o_0 and executes a trajectory t. Let $t = \{a_1, a_2, \ldots, a_H\}$ be the trajectory of length H, where each a_h is an action at step h. After each action a_h, the agent receives an observation o_h, thereby forming an observation se-

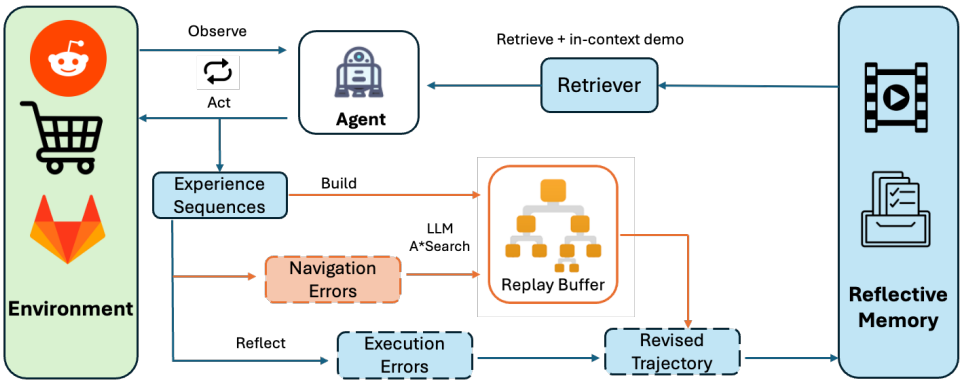

Figure 3: An overview of the **R2D2** architecture, highlighting the Remember and Reflect Paradigms. The Remember paradigm constructs a structured replay buffer from previous observations, enabling the agent to use past episodic data through A* search for navigation. Meanwhile, the Reflect paradigm diagnoses errors and generates corrective insights, which are then stored in a reflective memory for future decision-making processes.

quence $O = \{o_1, o_2, \ldots, o_H\}$. Consider N distinct user queries $\{q_1, q_2, \ldots, q_N\}$, each associated with its own episode and and observation sequence $O_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,H}\}$. Across all N user queries, **R2D2** forms the union of these observations:$\mathcal{O}_{\text{all}} = \bigcup_{i=1}^{N} O_i$.

R2D2 addresses errors in trajectories by categorizing them into two distinct types: (1) **Navigation failure.** We define O^* as the *key observations* essential for successfully addressing the query q. After performing the trajectory t, the observation sequence do not contain all the key observations, which leads to agents' navigation failure in the environment, formally $O \cap O^* \neq O^*$. The agent fails because of incomplete information or tools to address the user query. (2) **Execution failure.** After performing t, $O \cap O^* = O^*$. In other words, navigation was successful, since the proper path was followed, but the agent still failed to address the user query.

The *Remember* paradigm aims to build a replay buffer from $\mathcal{O}_{all}$, allowing the agent to store and revisit past observations and experiences (Blundell et al., 2016; Schaul et al., 2016; Parisotto and Salakhutdinov, 2017; Savinov et al., 2018). The *Reflection* paradigm then corrects trajectories that failed due to execution issues, providing explicit rationales for these execution failures. The successful and corrected trajectories and their rationales are stored in a *reflective memory* for agent's future reference. Finally, the *Retriever* leverages this reflective memory by selecting relevant corrected trajectories as in-context demonstrations, thereby continually improving the agent's performance.

3.2 Remember Paradigm

Building the Replay Buffer. To effectively represent the web environment and enable its systematic reconstruction from the observation sequence $\mathcal{O}_{\text{all}}$, **R2D2** structures the replay buffer as a directed graph $G = (O, E)$. Here, the vertex set O includes the root node o_0, which corresponds to the homepage observation, and each subsequent vertex represents another webpage observation o_i. The edge set E consists of tuples $((o_i, o_j), a)$ where each edge corresponds to an action a that transitions the agent from one observation o_i to another observation o_j. Due to the noisy and dynamic nature of web pages, **R2D2** store the differences between consecutive observations at each vertex rather than the full webpage state.

A* Search within the Buffer. The search algorithm employs a breadth-first search (A^*) strategy (Hart et al., 1968; Meng et al., 2024) to navigate and evaluate web environments effectively. Instead of expanding nodes level-by-level, **R2D2** incorporates a heuristic that guides the search toward potentially relevant and promising webpages more efficiently (Bonet and Geffner, 1999; Guez et al., 2018; Moldovan and Abbeel, 2012). This heuristic, provided by the LLM, estimates the relevance and utility of exploring a particular webpage node, thereby reducing unnecessary expansions and focusing on paths that are more likely to yield correct information.

In A^* search, each node o in the replay buffer graph is associated with both a cost (e.g., the depth from the start node or the number of steps taken) and a heuristic $h(o)$, which estimates how close o

is to a relevant webpage that can answer the query q. We derive the heuristic by prompting the LLM to assess the likelihood that the subtree rooted at o will yield information relevant to q. A^* search proceeds by maintaining a priority queue that selects which node to expand next based on the sum of the cost-to-come and the heuristic estimate. Webpages that are deemed relevant are added to a candidate queue, prioritizing content that potentially answers the query, while non-relevant pages are bypassed to streamline the search. This exploration continues until all reachable nodes have been evaluated. Subsequently, for each candidate node in the queue, paths are constructed back to the root, mapping feasible routes that could satisfy the query. The LLM then ranks these paths based on relevance and utility, culminating in the selection of the optimal trajectory P^*.

Algorithm 1 Optimized Web Search Using A^* and Language Model

Require: User query q, Initial observation o_0
Ensure: Optimal solution trajectory P^* for q
Initialize replay buffer graph $G = (O, E)$
Add root node o_0 to O
Initialize priority queue Q_{A^*} with $(o_0, f(o_0) = h(o_0))$
Initialize candidate queue $Q_{cand} \leftarrow \emptyset$
while Q_{A^*} is not empty **do**
  $(o_i, f(o_i)) \leftarrow$ dequeue(Q)
  if IsRelevant(o_i, q, LLM) **then**
    $Q_{cand} \leftarrow$ enqueue$(Q_{cand}, (o_i, f(o_i)))$
  end if
  for each action a available at o_i **do**
    $o_j \leftarrow$ Transition(o_i, a) {Obtain next observation via action a}
    if o_j not in Visited **then**
      $h(o_j) \leftarrow$ Heuristic(o_j, q)
      $f(o_j) \leftarrow f(o_i) + h(o_j)$
      $Q_{A^*} \leftarrow$ enqueue$(Q, (o_j, f(o_j)))$
    end if
  end for
end while
Initialize trajectory set $\mathcal{T} \leftarrow \emptyset$
for each o_i in Q_{cand} **do**
  $t \leftarrow$ Backtrack(o_i) {Generate trajectory from o_0 to o_i by following parent pointers}
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$
end for
$P^* \leftarrow$ RankAndSelectOptimal$(\mathcal{T}, q)$ {Use LLM to rank trajectories based on relevance and utility}
return P^*

This A^*-base approach enables more informed and targeted exploration of the replay buffer. By guiding the agent through the environment with a heuristic informed by the LLM, **R2D2** narrows down the search space and accelerates the discovery of relevant information. This ultimately results in faster and more accurate trajectory generation, effectively addressing complex user queries.

3.3 Reflect Paradigm

R2D2 first determines[2] the error type of a failed trajectory t. For all navigation failures, **R2D2** performs search within the replay buffer to correct their navigation behaviors. We discuss details of how we address navigation failure in §3.2. We now discuss using reflection techniques to address execution errors.

The reflection process is designed to enhance the system's capability to learn from mistakes within trajectories rather than only successes (Madaan et al., 2023; Shinn et al., 2023). When the failure reason of a trajectory t is classified as an execution failure, we prompt the LLM to identify the first erroneous action a_i. The trajectory is then truncated to include only the actions before the error point, $\{a_1, a_2, \ldots, a_{i-1}\}$, which are considered correct. Following this, a detailed reflection on the erroneous action a_i is generated, providing a rationale for its failure and potential strategies for avoidance in the future. This reflection, along with the truncated trajectory, is stored in the reflective memory (Weston et al., 2015; Mirowski et al., 2017; Wayne et al., 2018) that is to be introduced in §3.4.

3.4 Reflective Memory Mechanism

We introduce the reflective memory mechanism that stores corrected and truncated trajectories for future retrieval. The reflective memory is structured as a key-value store:

Key-Value Architecture. The reflective memory mechanism functions as a key-value store where each user query is encoded into a unique query vector serving as the key, encapsulating the query's semantic intent for efficient retrieval via vector similarity metrics. The corresponding value comprises a truncated and corrected trajectory, as described in §3.1, along with reflective insights. Specifically, for execution failures, only steps up to the first error are stored, while for navigation failures, corrected trajectory segments are retained once identified. During inference, a new query vector is generated and matched against existing keys to retrieve the most relevant trajectories.

Basic Operations. In alignment with conventional memory module architectures, the reflective memory mechanism defines two basic operations: (1) *Lookup.* Given a query, the memory retrieves the value(s) associated with the closest key vectors. (2)

[2]Please refer to Fig. 7 in Appendices for the detailed prompt.

Update. If a newly truncated trajectory provides a more accurate or enriched reflection for an existing query, the memory updates the current value. **R2D2** uses an LLM to make such decision. Please refer to Fig. 8 in the Appendix for an example.

3.5 Paradigm Coordination and Inference

Exploration Phase. Using a ReACT agent (Yao et al., 2023), **R2D2** processes user queries and collects observational data to build $\mathcal{O}_{\text{all}}$. Trajectories are classified and corrected via Remember and Reflect paradigms, then stored in the memory.

Inference Phase. During inference, user queries are encoded into vectors and matched against reflective memory to retrieve relevant trajectories as in-context demonstrations (Karpukhin et al., 2020; Brown et al., 2020). These demonstrations guide the agent's response. Failed trajectories undergo reflection, and the memory is updated to improve future performance. This coordination allows **R2D2** to leverage past experiences and reflections, ensuring continuous learning and enhanced handling of complex queries.

4 Experiments

In this section, we evaluate the proposed **R2D2** framework for web agent tasks and compare it with baseline methods. We first delve into the details of our experimental setup (§4.1), discuss the results obtained(§4.2), and perform ablation studies to understand the strengths of different components (§4.3). Furthermore, we provide a comprehensive error analysis (§4.5).

4.1 Experimental Setup

Benchmark. We use the WebArena benchmark (Zhou et al., 2023). This benchmark comprises diverse web interaction scenarios, ranging from web shopping to customer relationship management system (CMS). The dataset consists of 812 user queries with annotated ground truth trajectories. The Webarena benchmark further provides a set of validators to programmatically validate the functional correctness of each task.

Implementation Details. We choose gpt-4o[3] as our main LLM for both Remember and Reflect paradigm. We use the retriv[4] framework as the backbone of the reflective memory index, and select "sentence-transformers/all-MiniLM-L6-v2" as the dense embedding model for our retriever.

Baselines. We compare our framework against several represetative agent frameworks: (1) **ReACT** (Yao et al., 2023): a widely-used framework, which takes an observation of the environment as input, performs Chain-of-Thought reasoning, and then generates the next action. (2) **Tree-Search** (Koh et al., 2024): an inference-time tree-search strategy to perform best-first tree-search in web environments. It enables agents to revert to the most recently validated state upon encountering a failed trajectory. (3) **LATS** (Zhou et al., 2024): a method based on Monte Carlo tree search that employs LLMs as agents, value functions, and optimizers for decision-making. (4) **Anticipatory Reflection** (Wang et al., 2024a): a method that explicitly considers potential failures before action, alignment and backtracking after actions to maintain task objectives. (5) **AutoEval** (Pan et al., 2024): methods that boost agent performance using domain-general automatic evaluators. (6) **BrowserGym** (Chezelles et al., 2024): a framework that incorporates additional actions and observation tools for agents to interact with the environment.[5]

4.2 Main Results

Overall Performance. As shown in Tab. 1, our **R2D2** model consistently achieves higher success rates than both Tree-Search and ReACT across all tasks. For example, on the CMS and Reddit tasks, **R2D2** outperforms Tree-Search by substantial margins. These gains demonstrate the effectiveness of combining a systematic replay buffer with a reflective memory paradigm. By leveraging past interactions, **R2D2** avoids repeated mistakes, leading to more accurate and efficient decisions.

Substantial Improvements in Complex Domains. The results in the CMS and GitLab domains are particularly notable. **R2D2** achieved a 30.4% success rate in CMS and 28.0% in GitLab, considerably higher than other tested methods. These domains often require complex navigations with web interfaces, where **R2D2**'s capability to leverage past

[3] OpenAI. (2024). ChatGPT (November 20th version).
[4] https://github.com/AmenRa/retriv

[5] We also notice there are other methods that use different setups, such as SteP (Sodhi et al., 2024) that employs human-engineered workflows, and AWM (Wang et al., 2024b) that uses BrowserGym framework that customizes a larger action space than standard WebArena to interact. For the sake of direct comparison, these frameworks that depend on additional human efforts are not taken into comparison here.

Method	Tasks					Total SR
	CMS	Reddit	Shopping	Map	GitLab	
ReACT(Yao et al., 2023)†	–	–	–	–	–	13.1%
Tree-Search (Koh et al., 2024)†	16.5%	10.5%	28.1%	25.8%	13.3%	19.2%
AutoEval (Pan et al., 2024)‡	–	–	–	–	–	20.2%
LATS (Zhou et al., 2024)‡	15%	**25%**	30%	27%	17%	22.5%
AR (Wang et al., 2024a)‡	16%	24%	32%	27%	18%	23.4%
BrowserGym (Drouin et al., 2024)‡	–	–	–	–	–	23.5%
R2D2†	**30.4%**	20.8%	**35.8%**	**29.6%**	**28.0%**	**27.5%**

Table 1: Performance comparison across multiple web-based tasks. Reported success rates (SR) are organized by model and method, including baseline approaches and our proposed **R2D2**. Superscripts indicate the model used: † GPT-4o, ‡ GPT-4. The baseline results are from corresponding papers.

visited states and reflect on past actions proves especially beneficial.

Comparison with Reflection-based Frameworks. When compared to complex frameworks employing sophisticated reflection mechanisms (e.g., AR, AutoEval), **R2D2** holds its own or exceeds performance, with a total success rate (SR) of 27.5%. While AR and AutoEval offer robust reflection capabilities, **R2D2**'s integrated approach to first remembering and then reflecting allows it to preemptively correct paths and further refine strategies. The success can be attributed to the method's dual-paradigm system. We show more analysis in §4.3.

4.3 Ablation Study

Ablating rounds of execution. To better understand the strength of our proposed framework, we compare **R2D2** with advanced baselines that emphasize reflection techniques. Fig. 4 illustrates a marked increase in the success rate of **R2D2** during initial episodes. Upon manual inspection, we attribute this early performance enhancement primarily to the effective resolution of navigation failures.

By the fifth episode, **R2D2** substantially outperforms AR and LATS, confirming its methodological superiority. This highlights **R2D2**'s ability to leverage historical data and adaptive strategies effectively. While AR demonstrates commendable learning capabilities through its anticipatory reflection, it fails to match **R2D2**'s effectiveness. LATS, in contrast, shows minimal improvement. These findings support the practical superiority of the **R2D2** model in dynamic learning environments.

Ablating Remember & Reflect Paradigms. In this ablation study focused on the CMS domain, the full **R2D2** model substantially outperforms its variants, as shown in Fig. 5. The "– Reflection" variant, which lacks advanced reflection capabilities, shows moderate gains, while the "– Navigation" variant, which removes navigation, achieves only marginal improvement. Notably, the "– Reflection" variant, though initially showing some improvement, demonstrates a limited performance increase in later episodes, suggesting that while navigation capabilities can provide early benefits, their effectiveness without reflection support plateaus quickly. This observation highlights the critical role of navigation in sustaining performance improvements over time, reinforcing that reflection alone is insufficient for long-term success in complex web environments.

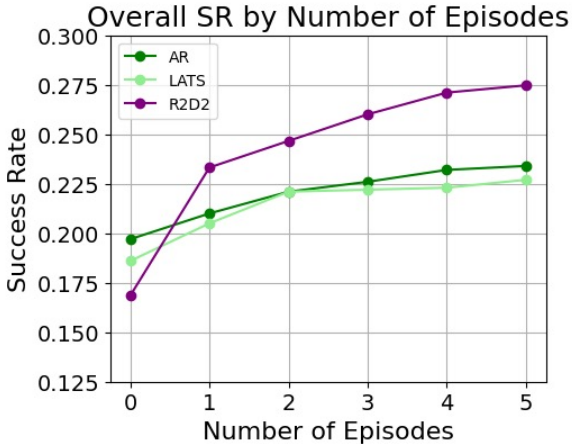

Figure 4: Performance comparison with different reflection-based methods. **R2D2** achieves marked increase at the first episode. Our manual inspection in Fig. 6 shows 75% of the initial increase is attributed to fixing the navigation failures.

Ablating Failed Trajectories. To elucidate the learning dynamics of **R2D2**, we conduct a study to isolate the impact of failed trajectories. During this study, only successful trajectories are provided as in-context demonstrations at inference time,

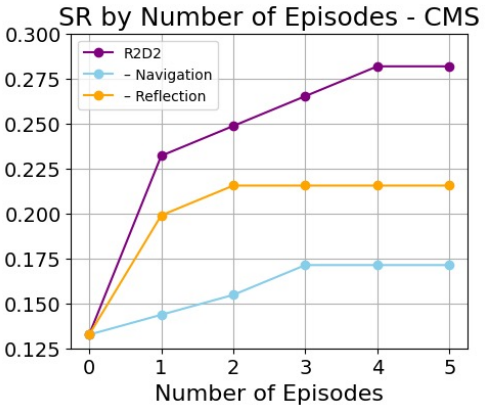

Figure 5: Performance comparison with ablation variants. Removing navigation or reflection capabilities from **R2D2** is very harmful to performance.

Method	Accuracy	Steps
Tree-Search (Koh et al., 2024)	19.2%	33.8
AutoEval (Pan et al., 2024)	20.2%	29.2
R2D2	**27.5%**	**13.1**

Table 2: Comparison of task accuracy and number of actions required. R2D2 reduces the number of online steps while maintaining a higher success rate.

thereby restricting **R2D2** to learning exclusively from positive examples. This variant falls 7.5% from the full implementation of **R2D2** to 20.5%. This also reveals a critical limitation: the number of positive examples is insufficient to provide robust navigation and reflection to the agent during inference. Consequently, if no relevant successful trajectory is identified at retrieval time. These findings substantiate the hypothesis that failed trajectories, despite not directly addressing user queries, are instrumental in enriching **R2D2**'s strategic repertoire, and **R2D2** extend beyond the mere memorization of positive examples.

4.4 Efficiency Analysis

Beyond improving task success rates, **R2D2** also demonstrates significant efficiency gains by reducing the number of online steps required per task. As shown in Table 2, we compare the average steps taken to successfully address a task between **R2D2** and open-sourced representative baselines. **R2D2** completes tasks with fewer steps on average, achieving higher success rate[6]. Because web-based tasks are often bottlenecked by interactions with the live environment rather than by language model queries, minimizing the number of online steps reduces latency and overall inference time. Consequently, **R2D2** 's ability to leverage a cached replay buffer avoids frequent back-and-forth rollouts, leading to improved efficiency alongside its superior performance.

[6]Offline memory construction for **R2D2** involves at most five actions per task, and the replay buffer creation is rule-based, making it lightweight.

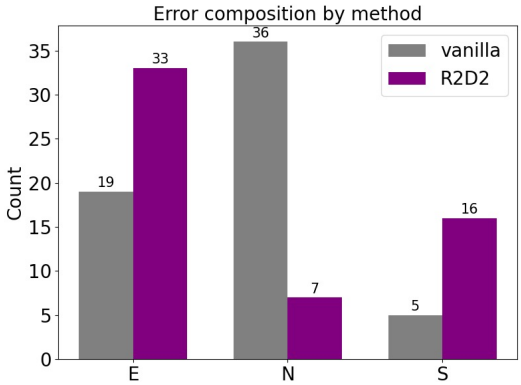

Figure 6: Error analysis of vanilla ReACT agents' and R2D2's trajectories. "E" indicates execution failures, "N" indicates navigation failures, and "S" indicates success. **R2D2** substantially reduces navigation failures, achieving higher success rate.

4.5 Error Analysis

As shown in Fig. 6, we manually inspect the trajectories of the same 60 queries executed by the vanilla ReACT agent and **R2D2** agent. About 60% of the vanilla ReACT agent trajectories stall at the navigation stage, so there is not even an opportunity to fail in execution. In contrast, the **R2D2** agent substantially reduces navigation failures, reliably guiding itself toward the right content and thereby reaching a point where it is possible to fail in execution more frequently. As a result, **R2D2** achieves a higher pass rate overall. We further annotate and discuss erroneous trajectories in Appx. §A. We also conduct a qualitative evaluation of the **R2D2** framework, as detailed in Appx. §C.

5 Conclusion

The **R2D2** framework significantly enhances web agents' capabilities by integrating Remember and Reflect paradigms, enabling more effective navigation and interaction in complex web environments. This approach leads to measurable improvements in performance, reducing errors and increasing task completion rates. **R2D2** not only outperforms existing models but also offers a scalable solution adaptable to various domains. Future work could extend its application, further optimizing agent functionality across broader scenarios.

Limitations

Language Studied. Our experiments were exclusively conducted in English. This limitation restricts our understanding of the model's efficacy across different linguistic contexts, potentially overlooking cultural and language-specific nuances that could affect the agent's performance in non-English web environments.

Focus on a Single Benchmark. Our experiments are confined to the WebArena benchmark using GPT-4o, which may raise concerns about their broader applicability. However, WebArena spans a broad set of tasks—ranging from online shopping to social media interactions—and **R2D2** 's strong performance across these varied scenarios suggests that our cached-search approach and Remember/Reflect paradigms are not restricted to a single domain.

Resource Constraints. Each complete pass through WebArena incurs a cost of approximately $200 in GPT-4o usage, making large-scale or multi-benchmark experimentation logistically challenging. That said, our approach—leveraging an external replay buffer, reflection protocols, and a search-based decision layer—does not inherently depend on GPT-4o. We anticipate that future research can replicate these methods using different LLM backends or other text-based environments, suggesting that our reliance on WebArena and GPT-4o does not fundamentally limit **R2D2** 's scope or generalizability.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent s: An open agentic framework that uses computers like a human.

Anonymous. 2024. Harnessing webpage UIs for text-rich visual understanding. In *Submitted to The Thirteenth International Conference on Learning Representations*. Under review.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. 2016. Model-free episodic control. *arXiv preprint arXiv:1606.04460*.

Blai Bonet and Hector Geffner. 1999. Planning as heuristic search: New results. In *European Conference on Planning*, pages 360–372. Springer.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2023. Rt-1: Robotics transformer for real-world control at scale.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. 2024. The browsergym ecosystem for web agent research.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. Workarena: How capable are web agents at solving common knowledge work tasks?

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for gui agents.

Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2024. Is your llm secretly a world model of the internet? model-based planning for web agents.

Arthur Guez, Théophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. 2018. Learning to search with mctsnets. In *International conference on machine learning*, pages 1822–1831. PMLR.

Peter Hart, Nils Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2024a. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments.

Wenhao Huang, Zhouhong Gu, Chenghao Peng, Jiaqing Liang, Zhixu Li, Yanghua Xiao, Liqian Wen, and Zulong Chen. 2024b. AutoScraper: A progressive understanding web agent for web scraper generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2371–2389, Miami, Florida, USA. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

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

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

Xiaogeng Liu, Peiran Li, Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. 2024. Autodan-turbo: A lifelong agent for strategy self-exploration to jailbreak llms. *arXiv preprint arXiv:2410.05295*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback.

Silin Meng, Yiwei Wang, Cheng-Fu Yang, Nanyun Peng, and Kai-Wei Chang. 2024. LLM-a*: Large language model enhanced incremental heuristic search on path planning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1087–1102, Miami, Florida, USA. Association for Computational Linguistics.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. 2017. Learning to navigate in complex environments.

Teodor Mihai Moldovan and Pieter Abbeel. 2012. Safe exploration in markov decision processes.

Ana F Palenciano, Carlos González-García, Jan De Houwer, Marcel Brass, and Baptist Liefooghe. 2021. Exploring the link between novel task proceduralization and motor simulation. *Journal of Cognition*, 4(1).

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents.

Emilio Parisotto and Ruslan Salakhutdinov. 2017. Neural map: Structured memory for deep reinforcement learning.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. 2018. Semi-parametric topological memory for navigation.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.

Paloma Sodhi, S.R.K Branavan, Yoav Artzi, and Ryan McDonald. 2024. Step: Stacked LLM policies for web actions. In *First Conference on Language Modeling*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, Bangkok, Thailand. Association for Computational Linguistics.

Sebastian Thrun. 2002. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi (Jim) Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024.

Haoyu Wang, Tao Li, Zhiwei Deng, Dan Roth, and Yang Li. 2024a. Devil's advocate: Anticipatory reflection for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 966–978, Miami, Florida, USA. Association for Computational Linguistics.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. 2023b. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. 2018. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models.

Oussama Zenkri, Florian Bolenz, Thorsten Pachur, and Oliver Brock. 2024. Extracting principles of exploration strategies with a complex ecological task. In *International Conference on Simulation of Adaptive Behavior*, pages 289–300. Springer.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2024. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning acting and planning in language models.

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

A Error Analysis

Among all the execution failures of **R2D2**, the errors can be classified as following:

Pessimistic Reflection (30.3%). When the agent makes a mistake and enters the reflection phase, it occasionally produces overly pessimistic rationales. Instead of proposing a plausible alternative action or a corrective step—such as trying a different button or re-verifying information on the same page—the agent may hastily conclude that the service is unavailable, broken, or that no solution exists. This pessimism not only mischaracterizes the underlying issue but also inhibits effective learning from the mistake. By prematurely giving up, the agent misses opportunities to refine its approach, explore subtle variations in the action sequence, or simply retry a failed step with slight modifications.

Lack of GUI understanding (24.2%). In certain scenarios, the agent struggles to properly interpret or interact with the graphical user interface (GUI) elements of the webpage. For example, when the user's query requires submitting information through an online form, the agent may fail to pinpoint the correct input fields or submission buttons, even after correctly navigating to the right page. As a result, it may click on the wrong element, repeatedly fail to submit required information, or get stuck trying to identify how to move forward.

Difficulty with Executing Complex Plan (20.2%). After reaching the desired section of a website, the agent may still falter when asked to carry out intricate, multi-step tasks. For instance, it could be instructed to iterate through a list of items, adding each one to a cart, verifying their details, and then proceeding to a checkout process. This difficulty suggests that, although navigation is now more reliable, the agent still needs improved reasoning capabilities and better long-term action planning to handle scenarios that demand careful step-by-step execution.

B Prompt Details

In this section, Fig. 7 and Fig. 8 show prompt details in the paradigms.

Error Classification Prompt

Please review the agent's attempt to answer the user's query and decide whether it failed because it never gathered all the key pieces of information (navigation failure),
{Examples}
or because it had everything it needed but still couldn't answer correctly (execution failure),
{Examples}
After making your decision, briefly explain why, and suggest how the agent might improve—either by finding the missing information or by reflecting on its reasoning steps.

Figure 7: Error classification prompt to determine agent trajectory error type.

C Qualitative Analysis

As shown in Tab. 3, an illustrative case can be observed in the user query asking for the billing name of the oldest complete order. The trajectory imple-

User Query	gpt-4o Failed Traj.	R2D2 Intermediates	R2D2 Final Traj.
Get the billing name of the oldest complete order.	1. Click 'SALES' icon 2. Click order_id/65 3. Scroll down 4. Stop action: Grace Nguyen 5. ✗	**ROUND 1** **Navigation**: 1. Click 'SALES' 2. Click 'Orders' **Reflection**: Agent fails to reach the correct page.	1. Click 'SALES' icon 2. Click 'Orders' 3. Click '↑ Purchase Date' 4. Click 'Filters' icon 5. Click 'Status' 6. Type 'Complete' 7. Click 'Apply Filters' 8. Click 'View' 9. Stop action: John Lee 10. ✓
List the top 2 search terms in my store.	1. Click 'CATALOG' 2. Click 'SALES' 3. Click 'REPORTS' 4. Click 'CATALOG' 5. ✗	**ROUND 1** **Navigation**: 1. Click 'REPORTS' 2. Click 'Search Terms' **Reflection**: Agent fails to reach the correct page. **...** **ROUND 3** **Navigation**: Correct navigation! **Reflection**: Arrives correctly but needs to sort entries.	1. Click 'REPORTS' 2. Click 'Search Terms' 3. Click 'Hits' 4. Click '↓ Hits' 5. Click '↑ Hits' 6. Stop action: Top 2 search terms are 'hollister' (19 hits) and 'Joust Bag' (10 hits). 7. ✓

Table 3: Comparison of action trajectories for resolving user queries. This table illustrates how the GPT-4o model fails to navigate complex web interfaces, whereas the proposed **R2D2** framework successfully manages both intermediate steps and final actions to accurately address the same queries. Please refer to **??** for more detailed analysis.

Trajectory Evaluation Prompt for Update Operation

Please review these two agent's trajectories to answer the user's query and decide which one is better or closer to achieve the goal.
{Examples}
After making your decision, briefly explain why.

Figure 8: Trajectory evalution prompt for Update Operation. The LLM compares two trajectory and determines which one is better for addressing the user query.

mented by gpt-4o failed to resolve the query as it did not properly navigate the complex web interface. In contrast, the trajectory of **R2D2** shows a series of steps that meticulously navigate through the interface. This targeted navigation led directly to the successful completion of the task, emphasizing the necessity of precise and thoughtful navigation strategies to effectively interact with and extract information from sophisticated web environments.

In instances where navigation is executed correctly but is insufficient to solve the task, the reflection module of **R2D2** plays a crucial role. A clear example of this is the task to list the top 2 search terms in the store. While the gpt-4o trajectory navigates correctly to the 'Search Terms' section, it does not delve deeper into analyzing or sorting the data, resulting in incomplete and inaccurate information retrieval. Conversely, **R2D2** not only accesses the correct section but also actively manipulates the data display by sorting the search terms according to their hits, thereby precisely identifying and articulating the top search terms. This demonstrates the power of **R2D2**'s reflective capabilities.