DEVIL'S ADVOCATE: Anticipatory Reflection for LLM Agents

Anonymous ACL submission

Abstract

In this work, we introduce a novel approach that equips LLM agents with introspection, enhancing consistency and adaptability in solving complex tasks. Our approach prompts LLM agents to decompose a given task into manageable subtasks (i.e., to make a plan), and to continuously introspect upon the suitability and results of their actions. We implement a three-fold introspective intervention: 1) **anticipatory reflection** on potential failures and alternative remedy *before* action execution, 2) *post*-action alignment with subtask objectives and backtracking with remedy to ensure **utmost effort in plan execution**, and 3) comprehensive review upon plan completion for **future strategy refinement**. By deploying and experimenting with this methodology—a zero-shot approach—within WebArena for practical tasks in web environments, our agent demonstrates superior performance with a success rate of 23.5% over existing zero-shot methods by 3.5%. The experimental results suggest that our introspection-driven approach not only enhances the agent's ability to navigate unanticipated challenges through a robust mechanism of plan execution, but also improves efficiency by reducing the number of trials and plan revisions by 45% needed to achieve a task.

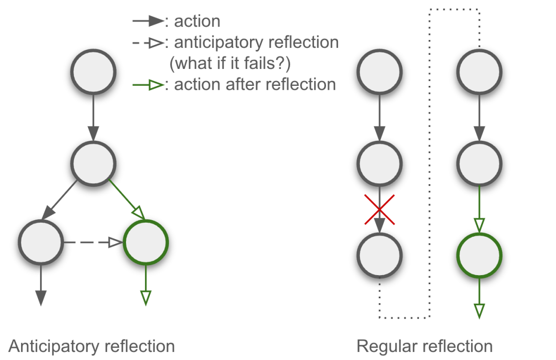

Figure 1: Conceptual difference between our anticipatory reflection and regular ones. Circles denote states and arrows actions. At the branching level, our method does not only yield the next action, but also anticipates a potential error associated with it and plans for backups. In contrast, regular reflection performs trials sequentially, correcting one error for each pass.

1 Introduction

Two roads diverged in a yellow wood,
And sorry I could not travel both
. . .
Then took the other, as just as fair,
And having perhaps the better claim

Robert Frost

The enduring appeal of Frost's emblematic poem, "The Road Not Taken," resides not just in its poetic elegance, but also in the profound lesson it imparts about decision-making. As we stand at the crossroads of a choice, it is a daunting challenge to assess probable outcomes and choose a course that best aligns with our objectives. This task becomes even more formidable when Large Language Model (LLM) agents (Huang et al., 2022b; Yao et al., 2023b; Song et al., 2023) have to navigate complex scenarios unfolding in real time, e.g., solving tasks in web environments (Liu et al., 2018; Yao et al., preprint; Deng et al., 2023; Zhou et al., 2024b), conducting simulated science experiments (Wang et al., 2022), and solving embodied household tasks (Shridhar et al., 2021).

Indeed, LLM agent decision-making has witnessed enhancement by post-hoc reflection and correction (Shinn et al., 2023; Song et al., 2024), coupled with adaptive planning (Sun et al., 2023; Prasad et al., 2023), where the agents learn from past successes and failures while concurrently mapping out flexible strategies. However, reflection usually works sequentially where only one hypothetical error can be corrected for each head-to-toe execution trajectory. Considering that such reflection is a test-time strategy, it poses a great efficiency issue. For instance, the agent could retry 10 times before concluding it still can not solve the task. Furthermore, self-reflection involves frequent shifts in plans which, albeit a mere inconvenience for humans, can lead to disorientation for AI agents. This may produce confusion, a standstill, or even an infinite loop of failure, which substantiates the importance of *thoroughly executing a set plan with utmost effort before resorting to a plan revision*. Therefore, this paper puts forward a methodology aimed at achieving an optimal balance between consistency and adaptability. This critical equilibrium mirrors the resilience and agility that is anticipated of a capable system that is prepared for curveballs but unwavering in the execution of its plan. Fig. 1 highlight our design in comparison to existing reflection strategy.

In this paper, we introduce a novel approach that integrates introspection into the fabric of LLM agents. This approach enables agents to continuously reflect on their actions, thereby stimulating a learning process that dynamically optimizes exploration paths and enhances robust decision-making under uncertainty. Our introspective intervention focuses on three principal dimensions:

1. Anticipatory reflection before action execution (similar to a devil's advocate);
2. Post-action evaluation and backtracking with remedy when necessary, to ensure the outcome aligns with subtask objectives;
3. An extensive review upon plan completion to generate finer plans for subsequent trials.

We implement this introspective methodology within WebArena (Zhou et al., 2024b), a comprehensive web environment featuring 812 tasks in five scenarios: online shopping, e-commerce management, social discussion forums, maps, and software development platforms. Experimental results demonstrate that our approach, which is zero-shot, substantially outperforms state-of-the-art zero-shot methods while improving efficiency, paving the way for a new paradigm of intelligent systems that are more consistent, adaptable, and effective[1].

2 Related Works

In this paper, we develop and expand upon several key themes within the realm of natural language processing, with a specific focus on the integration of action generation, planning, and reflection in the construction of LLM agents.

Action Generation LLMs have been employed in tasks requiring decision-making or action generation and have proven useful as agent-controlling policies in embodied environments (Huang et al., 2022b,a; Driess et al., 2023; Wang et al., 2023a; Zhu et al., 2023). They have also demonstrated effectiveness in text-based environments (Liu et al., 2018; Shridhar et al., 2021; Liu et al., 2023), where techniques like ReAct (Yao et al., 2023b) have shown notable benefits. Despite its success, ReAct's limitation lies in its inability to adjust to changes in the environment. Several improvements (Madaan et al., 2023; Shinn et al., 2023) have been proposed to counter these limitations, advocating for self-reflection to enhance decision-making and reasoning. However, these techniques primarily aim to improve single plans or trajectories without considering alternative actions, which could modify the plan in a wrong direction.

Position Bias Mitigation While comparing answer choices is generally effective, large language models used for action generation are not without flaws. They can exhibit bias, especially towards the first (or sometimes second) answer they see, regardless of its quality. This is known as position bias (Zheng et al., 2023; Wang et al., 2023b). Our method mitigates this bias by asking follow-up questions that challenge its own answer.

Planning Extensive research has explored the potential of LLMs in task planning (Dror et al., 2023; Prasad et al., 2023; Sun et al., 2023; Wu et al., 2023; Guan et al., 2023; Gur et al., 2024). The concept of decoupling planning and execution in formulating LLM agents has been validated through numerous paradigms such as ReWOO (Xu et al., 2023), ADaPT (Prasad et al., 2023), Structured Self-Reflection (Li et al., 2023), and DEFS (Wang et al., 2023c). Nonetheless, these methods exhibit a deficiency in establishing a resilient mechanism for plan execution, with agents frequently revisiting and revising their plans following each instance of adverse environmental feedback, often due to inaccurately executed actions. Our approach, conversely, emphasizes executing a previously defined plan with unwavering effort before considering any modifications. This guarantees a more stable and consistent problem-solving process. To implement this, the factor of tree search becomes crucial for exploring the best solutions. Past approaches, including ToT (Yao et al., 2023a), RAP (Hao et al., 2023), LATS (Zhou et al., 2024a), AdaPlanner (Sun et al., 2023), and ToolChain* (Zhuang et al., 2024), have incorporated tree search techniques in iden-

[1]Code to reproduce our results will be released.

tifying the optimal route to the desired solution. However, our approach distinguishes itself by engaging the LLM in preparing alternate solutions in anticipation of impending failures, ensuring more comprehensive consideration in action generation.

Reflection and Self-refinement Reflection and refinement techniques have advanced significantly through works such as Reflexion (Shinn et al., 2023), AdaPlanner (Sun et al., 2023), and Auto-Eval (Pan et al., 2024). Our methodology further enhances this by incorporating an anticipatory reflection mechanism that operates before each action rather than performing post-hoc reflection after each complete trial. This approach simplifies exploration by expediting remedial action and reducing extensive backtracking and serial plan revisions, thereby improving the overall efficiency.

3 Method

Given a task $\mathcal{T}$ and an environment $\mathcal{E}$ with which the LLM agent G interacts, our objective is to enable the agent to systematically and adaptively complete the task through introspective methods. We first present how we decompose the task and generate action regarding each state in the environment in §3.1 and §3.2. Then we introduce the introspection mechanism in §3.3.

3.1 Task Decomposition and Planning

The first step involves decomposing the task $\mathcal{T}$ into subtasks in a sequential manner, forming a plan. This decomposition is achieved through an LLM generation process. Let G_{plan} denote the agent's plan generation function, prompted by the task $\mathcal{T}$, description of the initial state S_0, and any experience from past trials, i.e., history $\mathcal{H}$:

$$\mathcal{P} \sim G_{\text{plan}}(\mathcal{T}, S_0, \mathcal{H}). \tag{1}$$

Here, the plan $\mathcal{P}$ is parsed into a sequence of ordered subtasks:

$$\mathcal{P} = (\tau_1, \tau_2, \ldots, \tau_N), \tag{2}$$

where τ_i represents the i-th subtask in the plan, and N is the number of subtasks. For instance, Fig. 2 shows a plan with 5 subtasks for solving a task in WebArena. The distribution of WebArena tasks based on the number of subtasks within each task is illustrated in Fig. 3. This also reflects the difficulty of the tasks in WebArena, where most tasks take 4-9 steps to complete.

Plan for task: *What is the color configuration of the picture frame I bought in Nov 2022*:
1. Click on the 'My Account' link to access your account details.
2. Click on the 'Order History' link to view your past orders.
3. Scroll down the page until you find the order from November 2022.
4. Click on the order details link for the order from November 2022.
5. Scroll down to the product details section to find the color configuration of the picture frame.

Figure 2: An example plan with 5 subtasks, generated by GPT-4. Subtasks are generated based on the first observation $\mathcal{S}_0$ and prior knowledge about web operation.

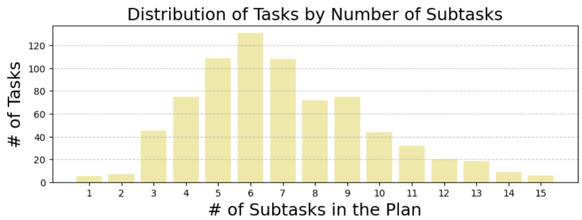

Figure 3: Distribution of WebArena tasks based on the number of subtasks within each task. The number of subtasks has a majority within 4-9 with a long tail distribution.

3.2 State and Action Representation

Let $S_t \in \mathcal{S}$ denote the current state of the environment at time t, where $\mathcal{S}$ is the set of all possible states. From state S_t, let $a_t \in \mathcal{A}$ denote the next action taken by the agent, where $\mathcal{A}$ is the set of all possible actions. The next action is generated based on the the specific subtask τ_i being addressed, current state S_t, and action history $\mathcal{H}_{t-1}$:

$$a_t \sim G_{\text{action}}(\tau_i, S_t, \mathcal{H}_{t-1}), \tag{3}$$

where G_{action} denotes the agent's action generation function. Let $\mathcal{H}_t$ denote the history of actions taken up to time t:

$$\mathcal{H}_t = \{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_t\}, \tag{4}$$

where $\hat{a}_t$ is a textual description of action a_t, along with useful information learned from this action execution, generated with function G_{describe}. The history would later be used to answer questions in the task or to revise the agent's plan. G_{describe} accepts as input the state before the action, the action itself, the state after the action:

$$\hat{a}_t \sim G_{\text{describe}}(S_t, a_t, S_{t+1}). \tag{5}$$

Algorithm 1 Introspective Agent

Input: task $\mathcal{T}$; initial observation S_{initial}; environment $\mathcal{E}$;
Initialization: time $t = 0$; state $S_t = S_{\text{initial}}$; action $a_t = \emptyset$; plan $\mathcal{P} = \emptyset$; subtask $\tau = \emptyset$; history $\mathcal{H} = \emptyset$;

1: **while** $\neg G_{\text{completed}}(\mathcal{T}, \cdot)$ **do**
2: $\quad \mathcal{P} \sim G_{\text{plan}}(\mathcal{T}, S_t, \mathcal{H})$; ▷ Plan Revision
3: $\quad \text{Stack} = [(S_t, a_t, \tau)]$;
4: $\quad$ **while** Stack **do**
5: $\quad\quad (S'_t, a_t, \tau) = \text{Stack.pop}()$
6: $\quad\quad$ **if** $S_t \neq S'_t$ **then** go_back(S'_t); $S_t = S'_t$; ▷ Backtracking
7: $\quad\quad$ **if** τ **is** $\emptyset$ **then** $\mathcal{C}_\tau = 1$; $\tau = \mathcal{P}.\text{next}()$;
8: $\quad\quad$ **else** $S_{t+1} = \mathcal{E}(a_t)$; $\mathcal{H}.\text{add}(G_{\text{describe}}(S_t, a_t, S_{t+1}))$; ▷ Grounding
9: $\quad\quad\quad \mathcal{C}_\tau \sim G_{\text{align}}(S_t, a_t, S_{t+1}, \tau)$; ▷ Alignment with Subtask Objective
10: $\quad\quad\quad$ **if** $\mathcal{C}_\tau$ **then**
11: $\quad\quad\quad\quad$ **if** $G_{\text{completed}}(\mathcal{T}, S_{t+1})$ **then** Finished; ▷ Early Stop
12: $\quad\quad\quad\quad$ **if** $G_{\text{completed}}(\tau, S_{t+1})$ **then** $\tau = \mathcal{P}.\text{next}()$; ▷ Next Subtask
13: $\quad\quad t{+}{+}$;
14: $\quad\quad$ **if** $\mathcal{C}_\tau$ **then** $a_t \sim G_{\text{action}}(\tau, S_t)$;
15: $\quad\quad\quad$ **for** $r = 1$ **to** R **do**
16: $\quad\quad\quad\quad a_t^{(r)} \sim G_{\text{remedy}}(\tau, S_t, a_t)$; ▷ Anticipatory Reflection
17: $\quad\quad\quad\quad \text{Stack.push}((S_t, a_t^{(r)}, \tau))$;
18: $\quad\quad\quad \text{Stack.push}((S_t, a_t, \tau))$; ▷ Placing a_t at the top of Stack

When the state observation is too long to fit in the context window of an LLM, the state is first summarized by the LLM into a shorter description before being fed to G_{describe} (e.g., this operation is commonly needed for solving web navigation tasks on content management platforms). Note that a subtask can involve several actions, and thus i does not necessarily equal to t. Given the possibility that the task can be finished at some time t before the completion of all subtasks, whenever the agent arrives at a new state, we ask the agent to check two things: whether the subtask is finished $\mathcal{C}_{\tau_i} \in (0, 1)$[2], and whether the task is finished $\mathcal{C}_{\mathcal{T}} \in (0, 1)$:

$$\mathcal{C}_{\tau_i} \sim G_{\text{completed}}(\tau_i, S_{t+1}, \mathcal{H}_t), \tag{6}$$
$$\mathcal{C}_{\mathcal{T}} \sim G_{\text{completed}}(\mathcal{T}, S_{t+1}, \mathcal{H}_t), \tag{7}$$

where $G_{\text{completed}}$ denotes the function for checking whether an objective is fulfilled. If $\mathcal{C}_{\tau_i} = 1$, the agent moves on to solve the next subtask τ_{i+1}; whereas when the agent determines $\mathcal{C}_{\mathcal{T}} = 1$, it finishes the current trial regardless of whether the plan $\mathcal{P}$ is finished.

3.3 Introspective Mechanisms

The sequential action generation above can potentially execute the plan and solve the task already. Nevertheless, without proper introspection and adaptation, the agent might be stuck at a certain unsolvable subtask or go into a loop of failure when unexpected problems emerge. Thus, we introduce three introspective mechanisms to enhance our LLM agent's problem-solving ability below.

3.3.1 Anticipatory Reflection (DEVIL'S ADVOCATE)

The first layer of introspection occurs before each action execution. The agent anticipates potential failures and comes up with R alternative remedies $[a_t^1, a_t^2, \cdots, a_t^R]$. Each remedy action is generated by prompting the LLM with a follow-up question:

- *"If your answer above is not correct, instead, the next action should be:"*

We use G_{remedy} to denote the generation of remedy actions, which accepts as input the subtask τ_i, the current state S_t, the action history $\mathcal{H}_{t-1}$, and the LLM predicted next action a_t at first attempt:

$$a_t^r \sim G_{\text{remedy}}(\tau_i, S_t, \mathcal{H}_{t-1}, a_t). \tag{8}$$

If later found necessary, the agent can go back to state S_t to modify the original action a_t to try the remedy action a_t^r to ensure a smooth plan execution. For example, in Fig. 4, we show a state observation where all three clicking actions align with the objective of the current subtask. The execution of any of these actions would complete the subtask; yet the agent might need to return to this state if it later determines that the action predicted at first attempt was incorrect[3].

[2] When the agent determines that a subtask is non-essential to solving the task, we also set $\mathcal{C}_{\tau_i} = 1$.

[3] The action generated at first attempt still gets the highest priority, i.e., a_t is the last one to be pushed to the stack so it can be popped and executed first (see line 18 in Alg. 1).

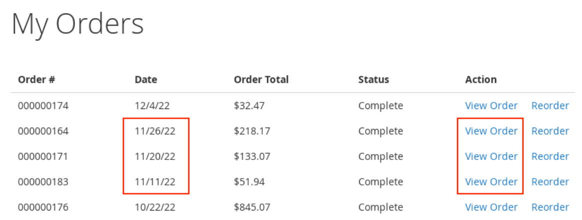

Figure 4: Screen observation at one step in solving the subtask: *Click on the order details link for the order from November 2022*. The agent might decide to click (a_t) on the "View Order" button of **any one of the three Nov 2022 orders** to see if a picture frame was purchased in that order, and it is highly probable that backtracking is needed to view the details of the other two orders (if the first chosen is not a picture frame). In our proposed approach, the other two alternative clicking actions $[a_t^1, a_t^2]$ would be pushed to stack before the agent executes action a_t.

3.3.2 Post-action Evaluation and Backtracking

The second introspective mechanism kicks in after the execution of each action. Here, the agent evaluates whether the action and the resulting state align with the subtask objective. This introspective function, denoted as G_{align}, is motivated by the state before the action S_t, the action a_t, the resulting state S_{t+1}, the current subtask τ_i:

$$\theta_t \sim G_{\text{align}}(S_t, a_t, S_{t+1}, \tau_i). \tag{9}$$

Here $\theta_t \in (0, 1)$ denotes the evaluation score reflecting how well the state S_{t+1} aligns with the subtask objective τ_i. It is a binary signal indicating whether the agent needs to stop and backtrack to some previous state and take an alternative action $a_k^r, k \leq t$, if the execution of a_t does not meet the objective of the current subtask. In our experiments with web environments, the URL of the webpage is a useful information recorded as part of S_t. When backtracking, we can easily navigate back to the URL. However, the element information on the URL might differ from the state we first encountered upon arriving at that page. To address this, we prompt the LLM to map the recorded element in the action to the new element with which we want to interact, if necessary.

3.3.3 Plan Revision

The third introspective mechanism occurs upon plan failure, i.e., when the stack is empty and $\mathcal{C}_{\mathcal{T}} = 0$. Now the agent performs a thorough review of the actions executed and the notes taken, and refines its future plan based on identified problems:

$$\mathcal{P}_{\text{new}} \sim G_{\text{plan}}(\mathcal{T}, S_0, \mathcal{H}_t). \tag{10}$$

Here, $\mathcal{P}_{\text{new}}$ is the new plan after reflecting on the past failed trials. The agent then re-enters the plan execution phase and starts a new episode.

Through these three layers of introspection, our agent is more capable of navigating the complexities of unforeseen circumstances and addressing tasks, bringing us a significant stride closer to achieving truly autonomous, adaptable, and intelligent systems. By structuring the problem in this manner, we have established a clear framework for enabling LLM agents to perform tasks autonomously and adaptively through introspection. Alg. 1 shows a pseudo code of our approach.

4 Experiments

In this section, we demonstrate how introspection enhances consistency and adaptability of LLM agents in solving complex tasks in web environments. We first introduce the experimental setup for evaluation (§4.1), followed by evaluation results (§4.2). Detailed error analysis is provided in §5, which highlights the directions for future endeavor.

4.1 Experimental Setup

Live Environments We evaluate our proposed method in the simulated web environments of WebArena (Zhou et al., 2024b), a dataset of human-annotated web browsing tasks designed to evaluate the ability of LLMs to perform complex, real-world actions on the internet[4]. The 812 tasks in WebArena involve five websites: an online shopping website, a software development website, a social forum platform, a map, and an e-commerce management platform; and these tasks can be categorized into three classes: information seeking tasks, site navigation and content & config tasks, and unachievable tasks. Though WebArena provides visual observation (screenshots), in this work we use the text observation only. The observation at each step is the accessibility tree of the webpage, and the elements in the accessibility tree are all within the current viewport of a 1280×720 screen. The action space of our LLM agent includes actions that interact with environment: *click*, *type*, *scroll*, *goto*, *go_back*, *go_forward*, and also a *note_down*

[4]Webarena (`https://webarena.dev`) is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

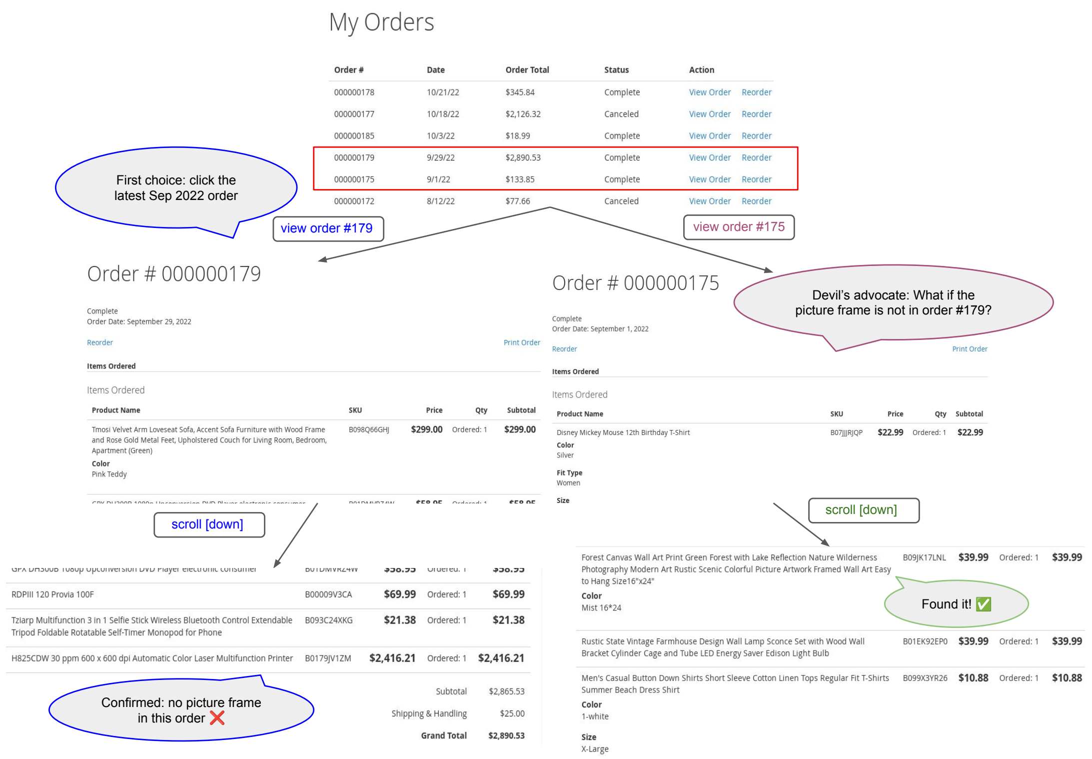

Figure 5: Decision making process of our agent in solving the task: *What is the color configuration of the picture frame that I bought in Sep 2022?* Before execution of the predicted action, the agent asks a follow-up question to itself regarding its decision: *what if the picture frame is not in order #179? what should be the alternative remedy?* And after finding out that order #179 contains no picture frame at all, the agent backtracks to the previous state to view order #175 and continue.

action that takes down useful snippet/summary for answering information-seeking questions.

Baselines We employ `gpt-4-0613`[5] (Achiam et al., 2023) with a context window of 8k tokens to build the agents and compare our method with three other agent construction strategies: planning and sequential decision making (Plan + Act w/o reflexion), similar to ReWOO (Xu et al., 2023); planning and sequential decision making with reflection (Plan + Act), similar to AdaPlanner (Sun et al., 2023); and tree search based planning, similar to LATS (Zhou et al., 2024a), but with reflection. In all methods, we set the upper limit on the number of actions to 30, i.e., after the agent executes 30 actions for a given task, it has to stop. In all three methods, we adopt the same prompts for action generation G_{action}, plan generation G_{plan}, and evaluator G_{align} and $G_{\text{completed}}$ to ensure a fair comparison[6]. In our experiments, we set the LLM temperature to 1.0 and max_tokens to 512, and keep all other parameters as default.

[5] `https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4`

[6] Detailed prompts are shown in the Appendix.

Metrics We follow the evaluation metric "Success Rate" in (Zhou et al., 2024b), and count the number of actions per trial and the number of plan revisions per task. To determine whether a task is successfully completed, the `exact_match` metric is used for some site navigation and information seeking tasks. However, this can sometimes be overly stringent. For instance, consider the URLs below that display the same content (under 'electronics', the category id of 'headphones' is 60). In fact, both of them point to exactly the same webpage. However, when evaluating for task completion, only the one that exactly matches a predefined finish URL is considered correct[7]. To address this issue, we manually review the evaluation process and correct such misjudgements in our results[8].

- `http://localhost:7770/electronics/headphones.html`
- `http://localhost:7770/electronics.html?cat=60`

[7] In WebArena, only the first URL link is used as the ground truth thus agent that reaches the second URL is judged as task incomplete.

[8] Our manual correction will also be released together with our code.

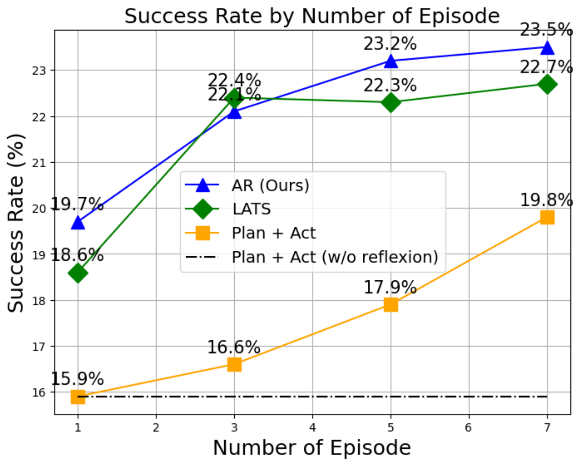

Figure 6: Results of different agent construction strategies on WebArena. AR is short for our method, anticipatory reflection; LATS represents our in-house implementation of the approach proposed by Zhou et al. (2024a); Plan + Act is a method of decomposition of task and execution of each subtask, similar to ReWOO (Xu et al., 2023). All three methods are equipped with plan revision (post-failure reflection).

4.2 Results

The experimental results, depicted in Fig. 6, demonstrate the efficacy of our introspection-driven approach in enhancing the consistency and adaptability of LLM agents in web environments. We compare the success rates of various agent construction strategies across multiple episodes. Our method, anticipatory reflection (AR), consistently outperforms the others, achieving a success rate of 23.5% after seven episodes, closely followed by LATS with 22.7%. In contrast, the Plan + Act method shows gradual improvement, reaching 19.8%, but remains significantly lower than the tree-search-based AR and LATS methods. Taking a closer look at the performance curve of LATS, there is an inconsistent pattern as success rate even drops at round 5. This is likely due to the homogeneous generated actions through direct sampling. In comparison, AR benefits from the "devil's advocate" approach, enabling more thorough planning and execution due to introspective follow-up questions. This trend underscores the importance of incorporating introspection mechanisms for both plan execution and revision, highlighting their critical role in enhancing consistency and efficiency.

Further insights can be gleaned from Tab. 1, which compares the average number of actions in the first and last trials across different methods. Our AR method shows an increase in the average number of actions from 6.39 in the first trial to 7.07 in the last trial, indicating a robust learning and adaptation process. In comparison, the average number of actions in the first trial of the Plan+Act method is only 4.01, suggesting that it stops at an early stage without completing full plan execution. Thus, our method effectively leverages a greater number of actions to achieve better outcomes, thereby reducing the number of plan revisions by 45% and improving overall efficiency.

	# of Actions First Trial	# of Actions Last Trial	# of Plan Revisions
Plan+Act	4.01	4.47	2.03
LATS	6.08	6.45	1.16
AR	6.39	7.07	0.64

Table 1: Statistics of the trajectory of different agents solving tasks on WebArena. We report the number of actions in the first and last trial, and also the number of plan revisions, i.e., trials.

5 Error Analyses

The subsequent sections shed light on an analysis of errors we observed from the agent's behavior when executing tasks. Two key areas have been identified for detailed discussion: an agent's occasional inability to fully learn from past failures, and inefficiencies in solving specific kinds of tasks due to a sequential planning scheme.

5.1 Agent Only Takes Partial Lesson from Past Failures

One category of common errors we notice is that the agent is not taking full lesson from past failure when generating a new plan. As illustrated in Fig. 7, the agent is at the final step of drafting a refund message for a Bluetooth speaker, after a series of steps taken to seek information for the order. From the screen, we know that the agent should consolidate all the information gathered from previous steps and type one piece of text into the (only) box titled *"What's on your mind?"*. However, as can be seen from the plans at the lower right corner in Fig. 7, while some improvements were made by adding the date of purchase and a more detailed explanation in the revised plan, the agent still failed to optimize the input process, repeating the typing actions separately for fields that do not exist. This inefficiency in the agent's behavior showcases the need for either an LLM with stronger reasoning

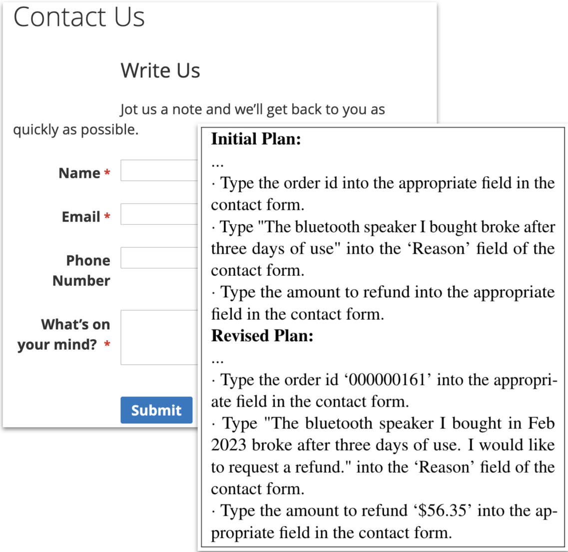

Figure 7: Screen observation at the last step to solve the task: *Draft a refund message via their "contact us" form for the bluetooth speaker I bought Feb 2023. It broke after three days of use. The shop requires the order id, the reason and the amount to refund in the message. Don't submit yet.*

ability or a better mechanism to solicit more comprehensive and accurate reflection.

5.2 Sequential Planning is Not Enough

In our analysis, we observed a recurrent error pertaining to the design of the agent's planning process. The proposed methodology structures a plan as a sequence of tasks that are executed in a specific order. Though it is effective in a decent amount of use cases, it seems to falter when faced with tasks necessitating more sophisticated logic. Specifically, tasks that mandate implementing a reusable function encapsulating several actions and employing a loop construct tend to challenge the model's current configuration. For example:

- *List out reviewers, if exist, who mention about average print quality.*
- *Give me the SKU of the products that have 1-3 units left.*
- *Like all submissions created by CameronKelsey in subreddit earthporn.*

Performing such tasks is analogous to executing SQL commands without a direct query API, but instead, in a realistic environment. The ability to process these tasks effectively would necessitate the incorporation of additional cognitive constructs into the planning model—e.g., memory, loops, repetitive actions, or encapsulation of a group of actions into callable functions. Though taking notes can help the agent eliminate wrong choices, these systemic extensions would add crucial capabilities to the web agent, significantly enhancing its navigation and problem-solving competence in realistic web environments. Moreover, while the current agent can succeed in the limited search space of simple tasks, it often struggles to review and introspect upon more descriptive tasks that require dynamic problem-solving. By addressing these limitations in future work, i.e., effectively converting textual description of a plan into robust execution of callable functions and loops, we believe that the reasoning capability of our agent can be substantially improved, leading to better outcomes in understanding and solving tasks that involve dynamic cognition in web environments.

6 Conclusions

In this work, we introduce a novel introspective methodology that significantly enhances the problem-solving capabilities of LLMs in complex environments, as demonstrated through comprehensive evaluations in the WebArena setting. Our approach strategically decomposes tasks into actionable subtasks and incorporates a three-tiered introspection process, which includes anticipatory reflection, robust post-action evaluation, and episode-level plan revision. This setup not only allows LLM agents to adapt their strategies in real time but also fosters long-term learning, reducing the need for frequent interventions as experience accumulates. The application of our introspective agent design in the WebArena benchmark demonstrates substantial performance gain (3.5%) over state-of-the-art zero-shot approach, along with stable performance curve with increasing number of rounds. Such benefits are accompanied by almost halving the number of plan revisions (45%) during error handling. In summary, by enabling LLM agents to proactively contemplate potential failures, evaluate actions post-execution, and continuously refine their strategy based on experiential insights, our approach equips AI systems with a human-like strategic thinking capability.

Broader Impact

Looking forward, the integration of multi-modal data inputs could further enhance the contextual understanding and decision-making accuracy of these agents. The principles and findings from our approach provide a robust foundation for future research in AI, particularly in aspects of autonomous decision-making, learning efficiency, and adaptability. As AI continues to integrate into diverse aspects of decision-making, embedding introspective capabilities will be essential to ensure these systems operate not only with precision but with an understanding akin to strategic human cognition.

Ethics Statement

As the capabilities of LLM agents enhance and their deployment in real-world applications increases, it is crucial to address potential ethical concerns, particularly regarding data privacy, bias, and transparency. Our work focuses on improving agent introspection to enhance task performance and decision-making explanations, aiming to develop more transparent and trustworthy AI systems. We emphasize the importance of human oversight to monitor and mitigate unforeseen consequences and encourage the responsible use of this technology for societal benefit. By promoting continuous evaluation and fair practices, we seek to minimize biases and ensure that the deployment of these agents does not exacerbate social inequalities. Furthermore, we are committed to optimizing computational resources to reduce the environmental impact, advocating for sustainable AI practices.

Limitations

Despite substantial progress made with our current design, limitations persist that inhibit optimal performance. Notably, the agent lacks a full learning mechanism to capitalize on past failures when generating a new plan, resulting in inefficient execution and recurring mistakes. Furthermore, while the sequential planning approach is effective for simpler tasks, it falls short for more sophisticated operations, such as those requiring encapsulated actions or loop constructs. Additionally, the agent struggles with tasks that expand beyond a simple search space, suggesting obstacles in handling dynamic problem-solving. Last but not least, our agent needs significant amounts of LLM generation (i.e., API calling), consequently requiring substantial time and computational resources, which dents its efficiency. Therefore, future work needs to concentrate on improving the agent's ability to fully learn from prior shortcomings, adapt to handle complex tasks, enhance dynamic problem-solving capabilities, and optimize time and resource utilization with more efficient LLM calling.

References

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

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.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. PaLM-E: An Embodied Multimodal Language Model. In *arXiv preprint arXiv:2303.03378*.

Rotem Dror, Haoyu Wang, and Dan Roth. 2023. Zero-Shot On-the-Fly Event Schema Induction. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 705–725, Dubrovnik, Croatia. Association for Computational Linguistics.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. In *The Twelfth International Conference on Learning Representations*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore. Association for Computational Linguistics.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv preprint arXiv:2201.07207*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022b. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *arXiv preprint arXiv:2207.05608*.

Tao Li, Gang Li, Zhiwei Deng, Bryan Wang, and Yang Li. 2023. A Zero-Shot Language Agent for Computer Control with Structured Reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11261–11274, Singapore. Association for Computational Linguistics.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv: 2308.03688*.

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.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous Evaluation and Refinement of Digital Agents.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2023. ADaPT: As-Needed Decomposition and Planning with Language Models. *arXiv*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and Error: Exploration-Based Trajectory Optimization for LLM Agents. *arXiv preprint arXiv:2403.02502*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. AdaPlanner: Adaptive Planning from Feedback with Language Models.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv: Arxiv-2305.16291*.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023b. Large Language Models are not Fair Evaluators.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your Agent Smarter than a 5th Grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023c. Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. 2023. Embodied Task Planning with Large Language Models. *arXiv preprint arXiv:2305.03716*.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. preprint. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In *ArXiv*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023a. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024a. 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, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024b. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *The Twelfth International Conference on Learning Representations*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. 2023. Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory. *arXiv preprint arXiv:2305.17144*.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2024. ToolChain*: Efficient Action Space Navigation in Large Language Models with A* Search. In *The Twelfth International Conference on Learning Representations*.

Appendix

Prompt for Plan Generation (G_{plan})

Imagine that you are imitating humans doing a task on a website step by step. You can click an element with the mouse, scroll up or down, go to a certain URL or go back to previous page, or type some text with the keyboard (e.g., click(), scroll(), goto(), go_back(), and type() functions in playwright). One step means one operation within any of the mentioned actions.

You are within a sandbox and only have access to the following websites to work with:

- An online shopping website (OneStopShop): {webarena_root}:7770
- An e-commerce management website (Magento): {webarena_root}:7780/admin
- A Reddit website (Postmill): {webarena_root}:9999
- A GitLab website: {webarena_root}:8023
- A map website (OpenStreetMap): `http://ec2-3-131-244-37.us-east-2.compute.amazonaws.com:3000`
- A Wikipedia website: `http://ec2-3-131-244-37.us-east-2.compute.amazonaws.com:8888/wikipedia_en_all_maxi_2022-05/A/User:The_other_Kiwix_guy/Landing`

Notes:

1. If you want to use the search function, you don't need to click on the search bar. You can directly use "type [element_id] [things_to_type]", and generally afterwards, you don't need to click the search button (by default, the command contains an ENTER at the end).

2. You can assume that you have signed in to your account (we have set up the cookies, so login is not needed).

The website that you will be working with is:
{WEBSITE INTRO}
Please follow these specific instructions to solve tasks:
{INSTRUCTION}
Here is a more detailed description of the starting screen:
{STARTING SCREEN DESCRIPTION}
Now, based on the information above, what should be the steps to achieve the following goal (please give me a list of textual description of playwright actions, starting with 'List'):
{TASK}
For your reference, here are some experiences from previous failed trials (please consider the following information to generate a better plan):
{FAILED PLAN}
Past experience:
{HISTORY}
To be successful in generating a new plan, you need to provide a list (1, 2, 3, ...), in which each item is a natural language description of one playwright action that is necessary to complete the task (e.g., click on the 'Account' button; scroll down; use the search bar to search for iPhone 13). You should use the information from the past experiences to save unnecessary steps!

Prompt for Action Generation (G_{action})

I am in a sandbox and only have access to the following websites (i.e., no access to external website like www.reddit.com):

- An online shopping website (OneStopShop): {webarena_root}:7770

- An e-commerce management website (Magento): {webarena_root}:7780/admin
- A Reddit website (Postmill): {webarena_root}:9999
- A GitLab website: {webarena_root}:8023
- A map website (OpenStreetMap): `http://ec2-3-131-244-37.us-east-2.compute.amazonaws.com:3000`
- A Wikipedia website: `http://ec2-3-131-244-37.us-east-2.compute.amazonaws.com:8888/wikipedia_en_all_maxi_2022-05/A/User:The_other_Kiwix_guy/Landing`

Now I'm trying to complete a task on a website.
The task is:
{TASK}
The plan to complete this task is:
{PLAN}
I have executed the following actions:
{HISTORY}
And now I'm at this step: {STEP}
Here is the screen I am looking at:
{OBS}
I have taken down the following notes:
{NOTES}
What should be the next action to complete this step in my plan (only give one action)?
Note:

- If the next action is to click, please indicate the element id in [] (format: click [element_id]).
- If the next action is to scroll, please indicate the direction in [] (format: scroll [up or down]).
- If you need to navigate to a URL, please indicate the URL in [] (format: goto [url]).
- If you need to go back to the previous page, please use go_back.
- If the next action is to type, please indicate both element id and the things to type in [] (format: type [element_id] [things to type]).
- If you want to note down something, use this format: note_down [things to note down].

The next action is:

Prompt for Objective Alignment (G_{align})

Imagine that you are imitating humans doing a task on a website step by step.
You are currently working on this step:
{STEP}.
The step above is one of the steps in the following plan:
{PLAN}.
From Screen 1, you executed an action and then arrived at Screen 2.
The action you executed was:
{ACTION}.
Screen 1:
{OBS1}.
Screen 2:
{OBS2}.
Now describe what this action is about in one sentence, starting with 'The action is to'.
Does this action align with the goal of the following step (i.e., are we moving towards the right direction; Answer YES or NO)?
{STEP}

Prompt for Task / Subtask Completion Evaluation ($G_{\text{completed}}$)

Imagine that you are imitating humans doing a task on a website step by step.
You are asked to solve the following task:
{TASK}
You made the following plan to solve it:
{PLAN}
To reach the current screen, you have previously executed the following actions:
{HISTORY}
You have taken down a few notes after each action as follows:
{NOTES}
And here is the accessibility tree of the current screen you are looking at:
{OBS}
Look at the screen, the task, and the actions you executed, and think thoroughly, is the task completed now?
If the task is completed, answer YES.
If the task is not yet completed (meaning further actions are yet to be executed), answer NO.

Prompt for Answer Delivery (G_{answer})

Imagine that you are imitating humans doing a task on a website step by step.
You are asked to solve the following task:

{TASK}

To reach the current screen, you have previously executed the following actions:

{HISTORY}

You have taken down the following notes (to help you answer the question eventually) after each action:

{NOTES}

And here is the accessibility tree of the current screen you are looking at:

{OBS}

Based on the above information, answer the question in the task (starting with ###Answer).

Prompt for Element Mapping (G_{map})

I want to interact with an element with element id: {element_id} in the following screen:

{OBS1}

Now if I want to click on the same element in the following screen, what should be the element id now?

{OBS2}

New element id is: