IMPROVING AUTONOMOUS AI AGENTS WITH REFLECTIVE TREE SEARCH AND SELF-LEARNING

Xiao Yu[1], **Baolin Peng**[2†], **Vineeth Vajipey**[1], **Hao Cheng**[2], **Michel Galley**[2]
Jianfeng Gao[2*] **& Zhou Yu**[1*]
[1]Columbia University [2]Microsoft Research, Redmond
`{xy2437, zy2461}@columbia.edu`
`{baolinpeng, jfgao}@microsoft.com`

ABSTRACT

Autonomous agents have demonstrated significant potential in automating complex multistep decision-making tasks. However, even state-of-the-art vision-language models (VLMs), such as GPT-4o, still fall short of human-level performance, particularly in intricate web environments and long-horizon planning tasks. To address these limitations, we introduce Reflective Monte Carlo Tree Search (R-MCTS), a novel test-time algorithm designed to enhance the ability of AI agents, e.g., powered by GPT-4o, to explore decision space on the fly. R-MCTS extends traditional MCTS by 1) incorporating contrastive reflection, allowing agents to learn from past interactions and dynamically improve their search efficiency; and 2) using multi-agent debate to provide reliable state evaluation. Moreover, we improve the agent's performance by fine-tuning GPT-4o through self-learning, using R-MCTS generated tree traversals without any human-provided labels. On the challenging VisualWebArena benchmark, our GPT-4o-based R-MCTS agent achieves a 6% to 30% relative improvement across various tasks compared to the previous state-of-the-art. Additionally, we show that the knowledge gained from test-time search can be effectively transferred back to GPT-4o via fine-tuning. The fine-tuned GPT-4o matches 97% of R-MCTS's performance while reducing compute usage by a factor of four at test time. Furthermore, qualitative results reveal that the fine-tuned GPT-4o model demonstrates the ability to explore the environment, evaluate a state, and backtrack to viable ones when it detects that the current state cannot lead to success. Moreover, our work demonstrates the compute scaling properties in both training - data collection with R-MCTS - and testing time. These results suggest a promising research direction to enhance VLMs' reasoning and planning capabilities for agentic applications via test-time search and self-learning.[1]

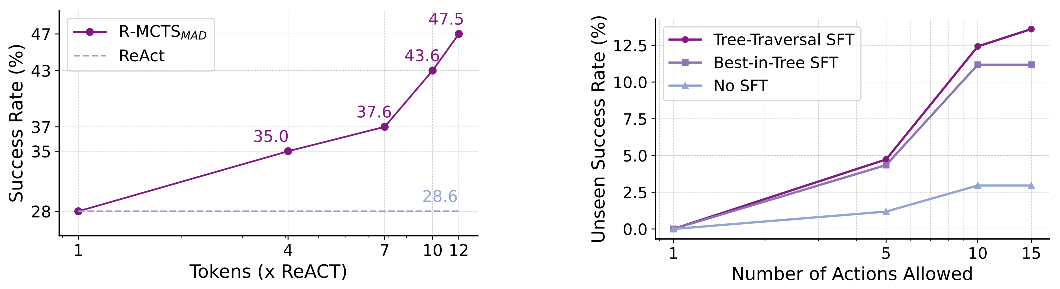


Figure 1: Our work yields compute scaling of GPT-4o with R-MCTS (left) and fine-tuned GPT-4o (right), for both training and testing, respectively. Left is evaluated on all 234 tasks from Classifieds in VisualWebArena, and right is evaluated on 169 unseen tasks from Classifieds.

*Equal Advisory Contribution; † Project Lead
[1]Code and data can be found at `https://aka.ms/RMCTS-Self-Learning`.

1 Introduction

Visual-Language Models (VLMs) have seen significant advancements, becoming increasingly powerful at processing and generating multimodal content. High-capacity models such as GPT-4o (OpenAI, 2024a), Gemini (Gemini, 2024), Phi-3 (Abdin et al., 2024), and Opus (Anthropic, 2024) have demonstrated promising understanding and reasoning abilities, paving the way for building VLM-based agents that can automate computer tasks such as software engineering (Yang et al., 2024; Wang et al., 2024), web navigation (Zhou et al., 2024b; Koh et al., 2024a) and mobile device control (Rawles et al., 2023; 2024). As many computer tasks require agents to interact with complex environments for long-term objectives, one critical challenge is to reason over extended periods and dynamically explore a large decision space. Current models often struggle at accessing a wide range of potential actions in these environments, and balancing exploration and exploitation in long-horizon tasks. Even GPT-4o-based agents significantly underperform humans, achieving success rates less than 20% compared to human performance of 89% (Koh et al., 2024a).

As demonstrated by OpenAI's o1 (OpenAI, 2024b) and Snell et al. (2024), increasing the number of thinking tokens as chain of thoughts at test time improves accuracy on tasks that require long-horizon planning and reasoning, such as math and coding. This naturally raises the question: Can we scale test-time computation to improve VLMs' decision-making capabilities in multistep planning and reasoning for *agentic* tasks? An intuitive strategy for agentic tasks, such as web navigation, is search. By exploring different actions and evaluating their consequences, models can gain a better understanding of the environment, reason about correct actions, and plan more effectively for long-term objectives. Recent work in this direction includes Tree of Thought (Yao et al., 2023a) and Search Agent (Koh et al., 2024b), which rely on best-first search in the form of BFS/DFS and A* search, respectively, to augment the VLM's decision-making process. However, these methods suffer from 1) a lack of balance between exploration and exploitation, which is crucial for handling complex tasks, and 2) an inability to learn from past interactions to improve future searches, which humans naturally do. More importantly, these works focus primarily on prompt-based methods and do not address how to *transfer* knowledge from search back into the model. This results in agents with high inference costs for complex tasks, as it requires the use of VLMs with search algorithms.

In this work, we explore how to effectively scale test-time compute to improve VLM agents, and efficiently transfer knowledge acquired from search back to the model to enhance its reasoning and planning abilities. First, we introduce our Reflective Monte Carlo Tree Search (R-MCTS) agent. Our method extends the classic MCTS with two innovations: *contrastive reflection* to improve search quality in real time using high-quality reflections obtained from past experiences, inspired by conceptual models of human learning (Marton, 2014); and a *multi-agent debate* technique for reliable state evaluation, inspired by collaborative and iterative human evaluation approaches (van der Lee et al., 2019). Second, to transfer new knowledge acquired from our R-MCTS agents with increased test-time compute, we propose to update the base VLM with two supervised fine-tuning (SFT) methods, 1) Best-in-Tree SFT, which involves utilizing the *optimal, final actions* returned from the search process; and 2) Tree-Traversal SFT, which leverages search tree *traversals* to enable the model to learn how to explore the environment, evaluate states, and backtrack to viable states.

Experiments on the challenging VisualWebArena (Koh et al., 2024a) show that our GPT-4o-based R-MCTS agent achieves a new state-of-the-art performance, with a 6% to 30% relative improvement across various environments compared to the previous state-of-the-art. Additionally, when trained with Best-in-Tree SFT on only 65 trajectories, GPT-4o can recover 97% of the search performance, while reducing the inference cost by 4x. Furthermore, GPT-4o trained with Tree-Traversal SFT demonstrates the ability to 1) explore, evaluate, and backtrack without any search augmentation. Moreover, our work yields the compute scaling properties in both training - where data collection with R-MCTS occurs - and testing time, as shown in Figure 1. This represents an important step towards improving VLM's planning and reasoning capabilities for agentic applications by leveraging test-time search and self-learning.

2 Background

2.1 Navigating the Web

Decision making in a complex environment is typically formulated as a Partially Observable Markov Decision Process (POMDP) of $(\mathcal{S}, \mathcal{A}, \Omega, \mathcal{T})$, which consists of sets of states, actions, observations, and a transition function. In the context of web navigation, a *state* $s \in \mathcal{S}$ represents the entire environment's state including the current webpage and database states; an *action* $a \in \mathcal{A}$ is an action the agent can express in natural language and execute in the environment (see Table 1); an *observation* is a textual or visual representation of the current webpage; and the *transition function* $\mathcal{T}(s, a) \rightarrow (s', o')$ executes an action a in the environment and returns the next state and observation. For more details, we refer the reader to Zhou et al. (2024b); Koh et al. (2024a).

2.2 Monte Carlo Tree Search

In long-horizon tasks with a large action space, Monte Carlo Tree Search (MCTS) (Silver et al., 2017; Świechowski et al., 2022) is a popular method to improve decision making. MCTS explores multiple actions, evaluates their consequences, and returns the best action for execution after conducting a large number of simulations. We briefly describe the MCTS algorithm here and refer to Appendix A.1 for more details. Given a state to search from, MCTS iteratively constructs a tree by: 1) *selecting* an action a to explore/exploit using Upper Confidence Tree Bound (UCT) (Kocsis & Szepesvári, 2006; Rosin, 2011); 2) *expanding and evaluating* the selected action by simulating the next state $\mathcal{T}(s, a) \rightarrow (s', o')$ and evaluating current progress; 3) *backpropagating* the evaluation to update the tree's value estimates of quality of the executed action. This search process is repeated until the search budget is exhausted, and the most-visited (Silver et al., 2017) or the highest-value action (Kocsis & Szepesvári, 2006) is returned.

Action Type a	**Description**
click `[elem]`	Click on `elem`.
hover `[elem]`	Hover over `elem`.
type `[elem]` `[text]`	Type `text` into `elem`.
press `[key_comb]`	Press a key combo.
new_tab	Open a new tab.
tab_focus `[index]`	Focus on the i-th tab.
tab_close	Close current tab.
goto `[url]`	Go to `url`.
go_back	Click back.
go_forward	Click forward.
scroll `[up/down]`	Scroll up or down.
stop `[answer]`	End with an output.

Table 1: Action space for web navigation defined in VisualWebArena (Koh et al., 2024a).

2.3 Setup

We now present the setup commonly used by VLMs for agentic tasks, e.g., that of VisualWebArena (Koh et al., 2024a). A web agent begins by receiving a task g in natural language (and images) and starting webpage o_0. Then, at each time step t: 1) the agent generates an action $a_t \in \mathcal{A}$ based on the last state s_t, which encodes information from past actions and observations; and 2) a_t is executed, transiting to a new state $\mathcal{T}(s_t, a_t) \rightarrow (s_{t+1}, o_{t+1})$, and observation o_{t+1} is returned to the agent. This agent-environment interaction continues until either the maximum number of steps T is reached, or the agent issues a `STOP` action to terminate the episode. Finally, a terminal reward $r_T \in \{0, 1\}$ is returned based on the final answer or the last environment state s_T.

In practice, visual agents are often implemented using a large vision-language model (VLM) such as GPT-4o (OpenAI, 2024a). Common approaches include directly prompting the VLM as a policy function to generate an action $\pi(\cdot) \rightarrow a$ given a task and previous actions and observations (Yao et al., 2023b; Koh et al., 2024a); or to additionally construct value functions $V(\cdot)$ and conduct simulation-based searches before returning an action (Yao et al., 2023a; Zhou et al., 2024a; Koh et al., 2024b). We detail these methods below.

VLM as a Policy Function Given a task g and an trajectory $\tau = \{o_0, a_0, \ldots, ..., a_{t-1}, o_t\}$, a policy $\pi(g, \tau) \rightarrow a_t$ returns the next action a_t to execute. Popular methods such as ReACT (Yao et al., 2023b) treat use a VLM model as policy by directly prompting it for an action, after optionally generating a reasoning step. Then, the environment executes action a_t and returns a new observation o_{t+1}. This process repeats until a `STOP` action is issued or the maximum number of steps is reached.

VLM as a Value Function To augment an agent's decision-making process, many recent works implement a value function $V(\cdot)$ to guide the VLM to predict better actions. Given a trajectory τ, a value function estimates the expected success rate given the current state s_t. However, since state s_t of the environment (e.g., including database information) may not always be accessible to the agent, value functions are often implemented using the current *observed* trajectory τ and the task description g instead: $V(g, \tau) \rightarrow [0, 1]$. In this work, we consider two value functions. A *single-*

agent value function (Yao et al., 2023a; Koh et al., 2024b) directly prompts a VLM with all inputs (task g, trajectory τ) to generate a value estimate $v_{\texttt{SA}}$:

$$v_{\texttt{SA}} = \text{VLM}\,(g, \tau) \in [0, 1],$$

and a value function based on multi-agent debate, which we describe in the next section.

3 Method

3.1 Reflective MCTS

Web tasks are highly diverse as different websites have different layouts and functionalities (e.g., Amazon versus Reddit), and almost every webpage is unique. Although search-augmented agents have shown promising results in web navigation tasks, their performance is limited by the complexity of web environments and their inability to quickly adapt to unseen environments.

We propose Reflective Monte Carlo Tree Search (R-MCTS), an extension of classic MCTS that improves the agent's decision making process on the fly by incorporating reflection over its past task executions, and state estimations using multi-agent-debate (Section 2.3). We present a high-level overview of an R-MCTS agent in Figure 2, and a pseudo-code description in Appendix A.2. Similarly to classic MCTS, an R-MCTS agent during **inference** predicts an action a_t by iteratively building a search tree that explores the current decision space (i.e., looping over selection, expansion, simulation, and back-propagation steps). Unlike classic MCTS, R-MCTS also iteratively **improves** the search process by 1) using contrastive reflection to identifying past mistakes/successes; and 2) updating the agent's policy and value functions (in-context) to improve its future task executions during inference. This improvement step helps R-MCTS to avoid repeating the same mistakes and to explore more promising actions when faced with similar situations in the future. For clarity, we describe this improvement process in the context of *policy* reflection, since value reflection is highly similar[2].

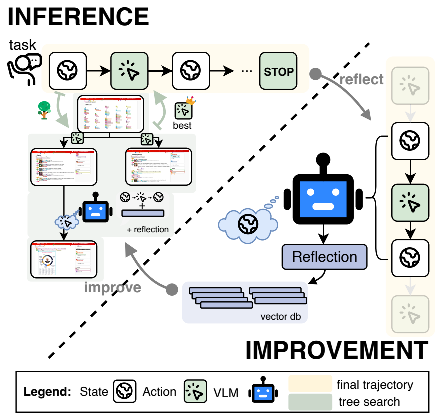


Figure 2: Overview of an R-MCTS Agent. We omit value function reflection for brevity.

Contrastive Reflection Policy (or value) function consists of three steps: 1) error attribution; 2) constrastive learning; and 3) memorization of reflection. Given a (long-horizon) trajectory τ, this process intuitively corresponds to first identifying the most 'erroneous' action, then analyzing the error by comparing what the agent *expected* to achieve with what *actually happens*, and finally storing this knowledge by remembering the task and state where the error occurred. This process is inspired by human cognitive learning (Marton, 2014), where contrasting our mental model of the world with reality serves as an effective way to gather new knowledge.

Formally, given a task g and trajectory τ, we first identify n_π most erroneous actions (and n_V value estimates for value reflection) for reflection, based on the difference between the VLM's *predicted* future success V and the search-tree's *simulated* future success Q:

$$\text{error}_\pi(a_t|\tau) = |V(o_{t+1}) - Q(o_t, a_t)|, \quad \text{error}_V(o_t|\tau) = |V(o_t) - Q(o_{t-1}, a_{t-1})|.$$

Note that this form of comparing value function V and action-value function Q is similar to Temporal Difference Error used in reinforcement learning (Sutton & Barto, 2018). For each of the erroneous actions $\{\tilde{a}_1, ..., \tilde{a}_{n_\pi}\}$, we then prompt the VLM to identify reasoning mistakes and gain understanding of navigating in the specific environment. We achieve this by contrastive learning[3]:

[2]For value reflection, we reflect on states for erroneous state estimations instead of actions. Then, we prompt the VLM to predict the expected future actions, and generate a reflection contrasting the agent's actual actions.

[3]For simplicity, we use the accessibility tree representation of o in text modality throughout this process.

we first 1) prompt the VLM to predict the expected transition $\hat{o}_{t+1}$ after executing $\tilde{a}_t$, and 2) prompt the VLM to generate a reflection by contrasting the current $o_t, \tilde{a}_t$, the expected $\hat{o}_{t+1}$ and real o_{t+1}:

$$\hat{o}_{t+1} = \text{VLM}_{\text{simulate}}(g, \{o_0, a_0, ..., o_t, \tilde{a}_t\}), \quad \text{reflect}(\tilde{a}_t | g, \tau) = \text{VLM}_{\text{reflect}}(g, o_t, \tilde{a}_t, \{o_{t+1}, \hat{o}_{t+1}\}).$$

Finally, to help the agent memorize this reflection at test-time, we embed the reflection using the current task and state (g, o_t) and store it in a vector database.

Improvement Then, given a new task and a new observation $(g^{\text{new}}, o_t^{\text{new}})$ to perform inference, we improve the policy (and value) function by 1) retrieving the most relevant reflections using cosine similarity of their embeddings[4], and 2) appending them into the agent's current context. For clarity, we also illustrate this reflection and improvement process in Figure 6 (Appendix A.2).

Multi-Agent Value Function In addition to the reflection-improvement loop, we also experiment with using multi-agent value function to provide more reliable state estimates. Intuitively, this method offers 1) a more holistic view of the current state to mitigate mistakes caused by oversights; and 2) stronger reasoning elicited by collaborative/adversarial incentives (Bowman et al., 2022).

Formally, a *multi-agent* value function prompts multiple VLMs to each generate a value estimate v^i, and then aggregate all estimates $\{v^1, v^2, ...\}$ to produce a final estimate $v_{\texttt{MA}}$:

$$v^i = \text{VLM}_i\,(g, \tau) \in [0, 1] \quad \text{and} \quad v_{\texttt{MA}} = \text{aggregate}(g, \tau, \{v^1, v^2, ...\}) \in [0, 1].$$

In this study, we implement the multi-agent value function using *multi-agent-debate* (`MAD`). Given a task g and a trajectory τ, `MAD` prompts two VLMs to generate two opposing arguments for the current value estimate (i.e., why the current state is promising/not promising), and then aggregates the two arguments using another VLM to obtain a final judgement:

$$\text{aggregate}(g, \tau, \{v^1, v^2, ...\}) = \text{VLM}_{\text{judge}}\left(g, \tau, \{v^1, v^2, ...\}\right).$$

For the prompts used by contrastive reflection, improvement, and `MAD`, please refer to Appendix D.2.

3.2 Self-Learning

Search-augmented agents such as R-MCTS improve performance at the expense of increased test-time computation. Using GPT-4o powered R-MCTS as an example, we explore how to effectively transfer the knowledge gained from tree search back to GPT-4o. We propose two methods: **Best-in-Tree SFT** (BiT SFT) which directly fine-tunes GPT-4o on the actions returned by the search algorithm, and **Tree-Traversal SFT** (TrT SFT) which finetunes GPT-4o on all tree traversals that happened during search. We illustrate these two methods in Figure 3.

Given a task g, a trajectory $\tau = \{o_0, a_0, ..., a_T\}$, and all search trees $\text{Tree}(o_0), .., \text{Tree}(o_T)$ obtained during R-MCTS, Best-in-Tree SFT trains GPT-4o to learn an improved policy by directly fine-tuning on the actions executed in τ (i.e., best actions selected by search). This method is simple, but relies on the model itself to learn the implicit reasoning process. Tree-Traversal SFT improves GPT-4o's decision making ability by using both τ and tree traversals $\text{Tree}(o_0), .., \text{Tree}(o_T)$. Specifically, given a tree $\text{Tree}(o_i)$, we 1) replay the tree search process[5] to obtain observations o, actions explored or backtracked a, and the estimated values v; 2) combine value estimation and action into a single action $a \leftarrow (v, a)$, and 3) append them to form a single trajectory τ_i'. We then repeat this for all trees, and obtain a single trajectory $\tau' = \{\tau_0', ..., \tau_T'\}$ used to train GPT-4o.

4 Experiments

We experiment on VisualWebArena (VWA), a benchmark designed to evaluate multimodal agents performance on across a wide range of web navigation tasks and environments. We describe the experimental setup, baselines, and other relevant implementation details below.

[4]For simplicity, we directly concatenate $(g^{\text{new}}, o_t^{\text{new}})$ into a single string and feed into an embedding model (text-ada-003-small, OpenAI (2024c)).

[5]This replay stops at the first time when the search process reached the best action.

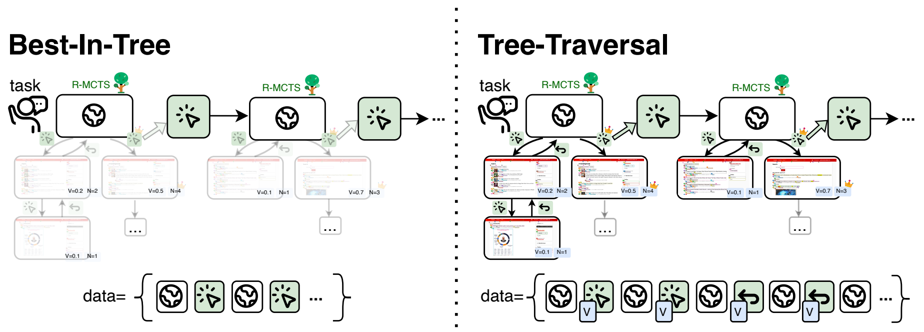


Figure 3: Given a task and a trajectory from R-MCTS, *Best-in-Tree SFT* removes intermediate search trees and directly trains GPT-4o to learn from the final executed actions; *Tree-Traversal SFT* flattens tree traversals into a single trajectory and trains GPT-4o to explore, evaluate, and backtrack.

Benchmarks VisualWebArena (Koh et al., 2024a) is a large-scale benchmark consisting of 910 tasks across three different web environments: Classifieds, Reddit, and Shopping. All tasks in VWA require visual understanding of webpage content to solve effectively, and 25.2% of the tasks also include images as input (e.g., help me make a post selling this item + an image of a phone). We host all web environments locally and evaluate on all 910 tasks, unless otherwise specified. We follow (Koh et al., 2024a;b) and use Set-of-Mark (Yang et al., 2023b) augmented web screenshots as the agent's input. An example is presented in Appendix D.1.

Baselines We compare R-MCTS against direct prompting methods as well as search-augmented agents backed by various algorithms. For direct prompting methods, we consider REACT (Yao et al., 2023b), which prompts a VLM to generate the reasoning step before generating the action. For search, we consider Tree of Thought (TOT, Yao et al. (2023a), which uses BFS or DFS to explore the decision space and returns the best action found according to V; SEARCH AGENT (Koh et al., 2024b), which is a best-first method inspired by A* search; and MCTS (Silver et al., 2017; Zhou et al., 2024a; Yu et al., 2023), which uses MCTS to determine the best action to take.

Policy and Value Implementation Search methods require a policy that can generate up to b actions for exploration, and a value function that returns a value between 0 and 1. We follow Koh et al. (2024b) to implement such a policy by 1) sampling up to 20 responses from the VLM with a temperature of 1.0 and a top-p of 0.95, using nucleus sampling (Holtzman et al., 2020); 2) aggregating the counts of each unique action; and 3) returning the top-b actions with the highest counts. We implement the single-agent value function V_{SA} by 1) prompting the VLM with a multiple-choice based prompt; 2) sampling 20 responses with temperature of 1.0 and top-p of 0.95; 3) converting the selected choices into a numeric value; and 4) returning the average value of all responses. We implement the multi-agent value function V_{MA} by 1) prompting the VLM twice to generate reasons why the current state is promising/unpromising; 2) prompt the VLM again to generate a final judgement with a multiple-choice-based prompt; and 3) converting the selected choice into a numeric value. We use GPT-4o (2024-05-13) (OpenAI, 2024a) as the VLM for all methods as it has been widely used in the development of state-of-the-art agents (Koh et al., 2024b; Wang et al., 2024)

Search Parameters We follow Koh et al. (2024b) to compare all search methods with a breadth b and depth d limit of 5 per tree, and to stop task execution after a maximum of 5 actions. Since different algorithms behave differently *during* search, we also set a maximum search time of 5 minutes per state. For MCTS-based algorithms, we use a UCT bound with $c_p = 1.0$ for action selection, and use the most-visited child for selecting the best action (Silver et al., 2017).

4.1 R-MCTS RESULTS

We summarize the performance of R-MCTS and other search-augmented agents in Table 2. We observe that all search-based methods deliver significant improvements across the VWA environments, albeit with increased test-time compute measured by the number of tokens. Among these methods, MCTS consistently outperforms its counterparts. This is likely due to MCTS's ability to balance

Method	Search	Value	Classifieds		Reddit		Shopping	
			Tokens	Success	Tokens	Success	Tokens	Success
ReAct	-	-	1x	28.6%	1x	13.8%	1x	23.2%
ToT	BFS	SA	3.2x	30.7%	4.7x	19.5%	4.3x	29.2%
ToT	DFS	SA	3.1x	30.3%	4.2x	16.7%	4.0x	28.3%
Search Agent	Best-First	SA	4.2x	33.8%	5.4x	21.9%	5.1x	30.3%
MCTS	MCTS	SA	7.2x	37.6%	9.5x	23.8%	8.9x	29.4%
R-MCTS	MCTS	SA	7.3x	40.2%	7.3x	25.2%	7.6x	31.9%
R-MCTS	MCTS	MAD	7.4x	**41.0**%	9.7x	**28.7**%	10.1x	**32.3**%

Table 2: Comparing different agent's token consumption and performance on VisualWebArena. We show average token used per task using ReAct as a baseline.

Method	Model	Max Steps	Training	Tokens	Success		
					Seen	Unseen	Overall
ReAct	GPT-4o-mini	20	✗	1.0x	-	-	20.9%
ReAct	o1-mini	20	✗	1.0x[†]	-	-	23.9%
ReAct	GPT-4o	5	✗	1.0x	-	-	22.2%
ReAct	GPT-4o	20	✗	2.4x	-	-	22.2%
R-MCTS$_{\texttt{MAD}}$	GPT-4o	20	✗	9.6x	-	-	32.1%
ReAct	BiT SFT	20	65	2.5x	80.0%	12.4%	31.2%
ReAct	TrT SFT	20	35	3.6x	80.0%	18.6%	27.8%

Table 3: Comparing different model's performance evaluated in the Classifieds environment. Since GPT-4o finetuning does not support images, we run all experiments in a text-only modality provided by VisualWebArena. [†]While total token consumption is similar, o1-mini used 20x more output tokens (reasoning + output) than GPT-4o-mini's output tokens (output).

exploration and exploitation through UCT, whereas other methods primarily rely on V to guide the search. Additionally, our R-MCTS agent sets a new state-of-the-art performance benchmark across all VWA environments, with relative improvements ranging from 6% to 30% over the previous best-performing method, Search Agent. When using a single-agent value function, R-MCTS improves upon MCTS by an average of 2.2 points. Furthermore, with the integration of multi-agent debate, R-MCTS achieves an additional 1.6-point improvement on average. These findings highlight a promising approach to scaling test-time compute in agentic applications by integrating search, reflection, and multi-agent debate strategies.

4.2 Self-Learning Results

Next, we evaluate whether knowledge from R-MCTS can be transferred back to GPT-4o to improve its performance. Since GPT-4o fine-tuning does not support images, we evaluate all methods with a text-only modality from VWA. We mainly consider training and evaluating on 234 tasks from the Classifieds environment, as GPT-4o finetuning is costly. For simplicity, we refer to R-MCTS with a multi-agent-debate value function $V_{\texttt{MAD}}$ as R-MCTS$_{\texttt{MAD}}$.

Training Data We first run R-MCTS$_{\texttt{MAD}}$ on Classifieds, and then sampled 65 trajectories for Best-in-Tree SFT according to the estimated values of final success. However, since search tree traversals are typically long, we further remove trajectory that results in more than 20 actions, producing 35 trajectories. We format these data into multi-turn chats, and use OpenAI finetuning API for training.

Quantitative Results We present the results in Table 3. First, we find that both Best-in-Tree SFT and Tree-Traversal SFT outperform the untrained GPT-4o by a large margin without relying on any search algorithm. When compared to R-MCTS$_{\texttt{MAD}}$, we find that *both* methods recovered more than 85% of the performance while reducing compute usage by up to 4x. We believe this result indicates even strong models like GPT-4o can still significantly benefit from training with agentic data.

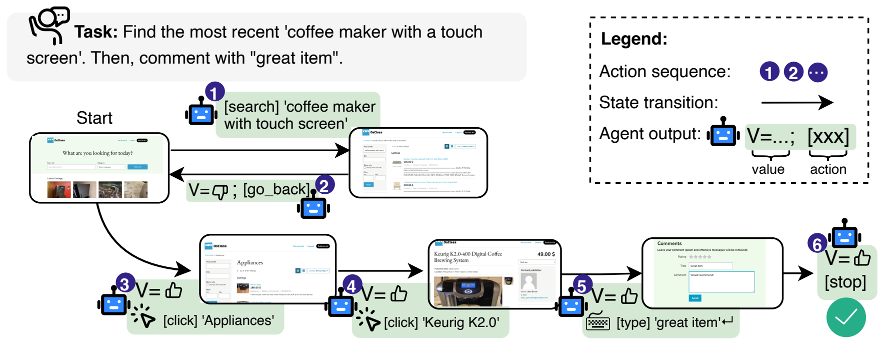

Task: Find the most recent 'coffee maker with a touch screen'. Then, comment with "great item".
Legend:
Action sequence: 1 2 ...
State transition:
Agent output: V=...; [xxx]
value
action
Start
1 [search] 'coffee maker with touch screen'
V=👎; [go_back] 2
3 V=👍 [click] 'Appliances'
4 V=👍 [click] 'Keurig K2.0'
5 V=👍 [type] 'great item'↵
6 V=👍 [stop]

Figure 4: After Tree-Traversal SFT, GPT-4o demonstrates the ability to explore, evaluate, and backtrack without augmenting with search algorithms.

Qualitative Results With Tree-Traversal SFT, we find that GPT-4o is able to perform better on unseen tasks by performing exploration, state evaluation, and backtracking without relying on any search algorithm. We illustrate an example in Figure 4: after attempting to directly search for "coffee maker with a touch screen", the TrT SFT-trained GPT-4o finds that the search function in Classifieds may be unreliable (top search result is a sofa) and backtracked to use category filters instead.

4.3 Compute Scaling Results

We explore whether R-MCTS$_{\text{MAD}}$ *and* the fine-tuned model after the self-learning stage has the property of test-time computing scaling, i.e., whether performance can increase when more test-time computes, such search tokens, are allocated. To study this, we follow the same experiment setup in our main results in Section 4.1, but vary the search budget to allow R-MCTS$_{\text{MAD}}$ for $\{2, 5, 10, 15\}$ nodes per search tree. We then compare performance (*Success Rate*) and token consumption per task compared to REACT (*Tokens*) in Figure 1a. Since searching with more nodes significantly increases time and cost, we report results by testing on all 234 tasks in the Classifieds environment.

As shown in Figure 1, our proposed R-MCTS and Tree-Traversal SFT exhibit strong scaling properties for both train-time and test-time compute. Particularly, in Figure 1 left, we find that R-MCTS$_{\text{MAD}}$ significantly improves performance when allocated with more test-time compute: using 15 nodes per tree, R-MCTS$_{\text{MAD}}$ achieves a relative improvement of 66% compared to REACT. Furthermore, Figure 1 right also illustrates that *without* search, both Best-in-Tree SFT and Tree-Traversal SFT substantially boost the performance of an untrained GPT-4o when allowed more actions per task. Notably, Tree-Traversal SFT demonstrates a more favorable scaling trend in test-time compute compared to Best-in-Tree SFT, as it is trained to learn to search. These results highlight the potential for self-improvement via reflective search and self-learning in agentic applications.

5 Analysis

5.1 Success Rate Breakdown

In addition to having tasks in different environments, VWA labels these task for *action difficulty*. These are labeled by estimating the number of steps an average human[6] would take to complete the task: easy (1-3 steps), medium (4-9 steps), and hard (10 steps or more). We evaluate all methods using the same setup as in our main results in Section 4.1. We present the results in Table 4.

R-MCTS improves performance across difficulties Compared to prior best, we find R-MCTS agent improved performance by an avearge of 2.5% absolute across all difficulties. This shows that the search and improvement loop in the R-MCTS agent not only benefits planning in long-horizon tasks, but also enhances robustness in solving easier tasks.

[6] Annotations guides are approximate and devised by human annotators of VWA (Koh et al., 2024a).

Multi-Agent-Debate benefits difficult tasks When using a multi-agent value function (R-MCTS$_{\text{MAD}}$), performance is further improved in medium and hard tasks. We believe this is because multi-agent debate encourages a more critical view of the current state, and making it less error-prone in long horizon tasks. However, in easier tasks, we find such method can suffer from "over-thinking" by trying to avoid non-existent issues, which can slightly reduce performance.

Method	Easy	Medium	Hard
REACT	42.4%	20.9%	11.6%
ToT$_{\text{BFS}}$	46.4%	28.0%	14.7%
ToT$_{\text{DFS}}$	46.8%	26.5%	12.8%
Search Agent	45.9%	29.6%	18.4%
MCTS	47.8%	32.5%	16.5%
R-MCTS	**50.7%**	33.6%	19.9%
R-MCTS$_{\text{MAD}}$	49.3%	**34.1%**	**23.6%**

Table 4: VWA performance by difficulty.

5.2 Ablation Studies

We perform ablation studies for each of component in R-MCTS$_{\text{MAD}}$ to understand their contributions to the overall performance. In Table 5, we report the overall performance across all 910 tasks in VWA when removing 1) reflection from value function; 2) reflection from policy; and 3) the search algorithm. We find that reflections for policy improvement are more crucial than reflections for value function, and that search is essential for competitive performance.

5.3 Qualitative Analysis

To better understand the search process in R-MCTS$_{\text{MAD}}$, we manually inspect 80 randomly sampled trajectories and their corresponding search trees. We compare against trajectories generated by SEARCH AGENT and analyze errors made by our method.

Method	Overall	
	Token	Success
R-MCTS$_{\text{MAD}}$	9.3x	33.7%
- *Value refl.*	9.0x	32.9%
- *Policy refl.*	8.6x	30.2%
- *Search*	1.0x	21.9%

Table 5: Ablation studies on VWA.

Exploration and Reflection improves action quality Amongst the 80 samples, we find that R-MCTS$_{\text{MAD}}$ outperforms SEARCH AGENT in 12.5% of the tasks. We find that this is primarily due to R-MCTS$_{\text{MAD}}$'s ability to improve its policy (60%) using the reflection-improvement process (Section 3.1); and its ability to balance exploration-exploitation (30%) using UCT instead of best-first.

GPT-4o still lacks finegrained image and web understanding In Figure 5, we categorize all errors made by R-MCTS$_{\text{MAD}}$ during its search process by 1. failing to correctly format the final answer (*answer formatting error*); 2. errors caused by the agent-environment interaction (*environmental error*); 3. errors caused by GPT-4o's inability to perform fine-grained image understanding (*model image understanding*); and 4. errors caused by GPT-4o's limited web understanding/reasoning ability (*model web understanding*). For a more detailed categorization, we refer the reader to Appendix C.1. Our analysis reveals that most of the errors are caused by the backbone VLM's inability to understand the provided webpage screenshot/product images. Examples include failing to correctly identify the item on *a-th row and b-th column* on a shopping page; and misunderstanding that the cheapest product shown on the *current page* (sorted by most recently listed) is the cheapest product available *on the entire site*. We believe these model issues are fundamental to many current agents that rely on prompting and suggest that model training (alike Section 3.2) is essential to further improve agent's performance.

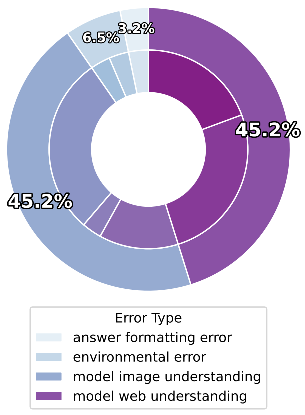


Figure 5: R-MCTS$_{\text{MAD}}$ error analysis. Sub-error types are omitted for brevity (see Appendix C.1).

6 Related Work

Language-Guided Agents Advances in large language models (LLMs) have motivated many recent works to re-purpose LLMs for automating agentic tasks (Nakano et al., 2022; Kim et al., 2023; Xi et al., 2023). These methods include prompting LLM directly as a policy (Hong et al., 2023; Yao et al., 2023b; Sridhar et al., 2023; Yang et al., 2023a), as a component of a manually designed agentic framework (Park et al., 2023; Sumers et al., 2024), combined with manually crafted heuristics/prompt designs to improve performance (Fu et al., 2024; Sodhi et al., 2024), or combined with tool uses for question answering/code generation (Yang et al., 2024; Xia et al., 2024). For tasks that requires visual understanding, many recent work have thus swapped the prompting LLMs to prompting visual language models (VLMs) such as GPT-4o (Zheng et al., 2024; Koh et al., 2024a). However, these methods typically suffer in long-horizon tasks that requires planning and error recovery. Our work thus augments VLMs agents with search algorithms such as MCTS (Silver et al., 2017) to complete tasks in complex and realistic web environments.

Augmenting Agent with Search/Reflection Using LLMs or VLMs to solve long-horizon tasks such as web navigation is a highly challenging task (Liu et al., 2023; Zhou et al., 2024b; Koh et al., 2024a). Many recent work thus also explored various strategies to improve agent's decision making process, such as: iteratively prompting the model to improve its own output (Madaan et al., 2023; Yu et al., 2024; Shinn et al., 2023); and augmenting agent's decision process using search algorithms such as BFS/DFS (Yao et al., 2023a), best-first search (Koh et al., 2024b), and MCTS (Yu et al., 2023; Zhou et al., 2024a). However, many of these methods primarily focus on text-based environments such as question answering and programming, and do not involve substantial interaction with a complex environment. In contrast, our work focuses on 1) conducting MCTS in long-horizon, agentic tasks in a realistic web environment; 2) incorporating multi-agent-debate value function to improve state evaluation; and 3) using contrastive reflection to learn from past successes and failures in long execution trajectories.

Training Agents Besides improving agent's performance at test-time, many works also explored methods to train the backbone LLM/VLM. Examples include Mind2Web (Deng et al., 2023), FireACT Chen et al. (2023), AgentInstruct (Zeng et al., 2023), WebGum (Furuta et al., 2024), AutoWebGLM (Lai et al., 2024), and more (Zhang et al., 2024b; Liu et al., 2024; Zhang et al., 2024a). These methods rely on collecting human or machine generated trajectories based on direct prompting, and perform supervised training to improve model's performance. To our knowledge, we are the first to train GPT-4o using trajectories produced by MCTS, and to teach GPT-4o to explore, evaluate, and backtrack in complex web environments through training on MCTS tree traversal.

7 Conclusion

In this work, we first introduced R-MCTS agent, a search-augmented agent powered by 1) MCTS to explore, evaluate, and backtrack in complex and realistic web environments; 2) a multi-agent-debate value function for reliable state evaluation; and 3) contrastive reflection to learn from past successes and failures in long execution trajectories. Then, we proposed methods to transfer knowledge acquired from search back to GPT-4o by 1) training on best actions returned by tree search (Best-in-Tree SFT); and 2) training on the entire MCTS tree traversal (Tree-Traversal SFT). Experiments on the challenging VisualWebArena benchmark show that R-MCTS achieves a new state of the art, and that our finetuned GPT-4o agent begins to display test-time compute scaling properties similar to search algorithms. We believe our work presents a promising direction to combine test-time search and model self-learning to develop the next-generation autonomous agents.

8 Limitations

High cost of scaling test-time compute Although MCTS-based agents achieve strong performance on benchmarks such as VWA, they consume nearly 10x more tokens compared to REACT. This presents a clear trade-off between performance and cost/time, and may limit the practicality of deploying such agents in real-world applications. However, these search-augmented agents can be used to *curate training data* for VLMs without human labels, which can in turn help develop stronger VLMs. Our work presents a promising step in this direction using Best-in-Tree SFT and Tree-Traversal SFT, and we believe such a self-learning loop is crucial to build stronger agents.

Long trajectories from tree traversal In long-horizon tasks, R-MCTS agents require a large amount of simulation to identify optimal actions, resulting in long tree traversals. Training on these long traversals can be costly, and may be difficult for models to learn from. Nevertheless, we believe this issue can be iteratively mitigated by using self-learning to improve model capability, and by conducting search using the improved policy/value functions. We leave explorations for future work.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Anthropic. The claude 3 model family: Opus, sonnet, haiku. `https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf`, 2024.

Samuel R. Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Christopher Olah, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Jackson Kernion, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Liane Lovitt, Nelson Elhage, Nicholas Schiefer, Nicholas Joseph, Noemí Mercado, Nova DasSarma, Robin Larson, Sam McCandlish, Sandipan Kundu, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Ben Mann, and Jared Kaplan. Measuring progress on scalable oversight for large language models, 2022. URL `https://arxiv.org/abs/2211.03540`.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023. URL `https://arxiv.org/abs/2310.05915`.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL `https://arxiv.org/abs/2306.06070`.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents, 2024. URL `https://arxiv.org/abs/2403.08978`.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models, 2024. URL `https://arxiv.org/abs/2305.11854`.

Gemini. Gemini: A family of highly capable multimodal models, 2024. URL `https://arxiv.org/abs/2312.11805`.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020. URL `https://arxiv.org/abs/1904.09751`.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2023. URL `https://arxiv.org/abs/2308.00352`.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks, 2023. URL `https://arxiv.org/abs/2303.17491`.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pp. 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 354045375X. doi: 10.1007/11871842_29. URL https://doi.org/10.1007/11871842_29.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks, 2024a. URL https://arxiv.org/abs/2401.13649.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents, 2024b. URL https://arxiv.org/abs/2407.01476.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: Bootstrap and reinforce a large language model-based web navigating agent, 2024. URL https://arxiv.org/abs/2404.03648.

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. Agentbench: Evaluating llms as agents, 2023. URL https://arxiv.org/abs/2308.03688.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Ference Marton. *Necessary Conditions of Learning*. Routledge, 1st edition, 2014. doi: 10.4324/9781315816876. URL https://doi.org/10.4324/9781315816876.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022. URL https://arxiv.org/abs/2112.09332.

OpenAI. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/, 2024a. Accessed: 2024-09-28.

OpenAI. Introducing OpenAI o1. https://openai.com/o1/, 2024b. Accessed: 2024-09-29.

OpenAI. New embedding models and api updates. https://openai.com/index/new-embedding-models-and-api-updates/, 2024c. Accessed: 2024-09-28.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023. URL https://arxiv.org/abs/2307.10088.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2024. URL https://arxiv.org/abs/2405.14573.

Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, March 2011. ISSN 1012-2443. doi: 10.1007/s10472-011-9258-6. URL `https://doi.org/10.1007/s10472-011-9258-6`.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL `https://arxiv.org/abs/2303.11366`.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL `https://arxiv.org/abs/1712.01815`.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL `https://arxiv.org/abs/2408.03314`.

Paloma Sodhi, S. R. K. Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions, 2024. URL `https://arxiv.org/abs/2310.03720`.

Abishek Sridhar, Robert Lo, Frank F. Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation, 2023. URL `https://arxiv.org/abs/2305.14257`.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents, 2024. URL `https://arxiv.org/abs/2309.02427`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Chris van der Lee, Albert Gatt, Emiel van Miltenburg, Sander Wubben, and Emiel Krahmer. Best practices for the human evaluation of automatically generated text. In Kees van Deemter, Chenghua Lin, and Hiroya Takamura (eds.), *Proceedings of the 12th International Conference on Natural Language Generation*, pp. 355–368, Tokyo, Japan, October–November 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-8643. URL `https://aclanthology.org/W19-8643`.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Opendevin: An open platform for ai software developers as generalist agents, 2024. URL `https://arxiv.org/abs/2407.16741`.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023. URL `https://arxiv.org/abs/2309.07864`.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents, 2024. URL `https://arxiv.org/abs/2407.01489`.

Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions, 2023a. URL `https://arxiv.org/abs/2306.02224`.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023b. URL `https://arxiv.org/abs/2310.11441`.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL `https://arxiv.org/abs/2405.15793`.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

Xiao Yu, Maximillian Chen, and Zhou Yu. Prompt-based monte-carlo tree search for goal-oriented dialogue policy planning, 2023. URL https://arxiv.org/abs/2305.13660.

Xiao Yu, Baolin Peng, Michel Galley, Jianfeng Gao, and Zhou Yu. Teaching language models to self-improve through interactive demonstrations, 2024. URL https://arxiv.org/abs/2310.13522.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 2023. URL https://arxiv.org/abs/2310.12823.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*, 2024a.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems. *arXiv*, 2024b. URL https://arxiv.org/abs/2409.03215.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded, 2024. URL https://arxiv.org/abs/2401.01614.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024a. URL https://arxiv.org/abs/2310.04406.

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. Webarena: A realistic web environment for building autonomous agents, 2024b. URL https://arxiv.org/abs/2307.13854.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3): 2497–2562, July 2022. ISSN 1573-7462. doi: 10.1007/s10462-022-10228-y. URL http://dx.doi.org/10.1007/s10462-022-10228-y.

A Additional Implementation Details

A.1 Monte Carlo Tree Search

We describe the Monte Carlo Tree Search (MCTS) algorithm used in our work in Algorithm 1. We use the same MCTS search implementation for all relevant methods (MCTS, R-MCTS, and R-MCTS$_{\texttt{MAD}}$ agent). For clarity, we abuse the notation s to represent *observations* in this section, in order to comply with notations used in other works (Świechowski et al., 2022).

Algorithm 1 MCTS

```
Require: VLM policy $\pi(g, \tau)$
Require: VLM value $V(g, \tau)$
Require: environment $\mathcal{T}$
Require: goal $g$ and current observation $s$
Require: actual trajectory so far $\tau$ (before $s$)
 1: while search budget is not exhausted do
 2:     $s_{\text{curr}} \leftarrow s$
 3:     $\tau_{\text{curr}} \leftarrow \tau \cup s$
 4:     // selection
 5:     while $s_{\text{curr}}$ is not a leaf node do
 6:         $a \leftarrow \arg\max_a Q(s_{\text{curr}}, a) + U(s_{\text{curr}}, a)$
 7:         $s_{\text{curr}} \leftarrow \mathcal{T}(s_{\text{curr}}, a)$
 8:         $\tau_{\text{curr}} \leftarrow \tau_{\text{curr}} \cup \{a, s_{\text{curr}}\}$
 9:     end while
10:     // expansion
11:     $\{a^1, a^2, ..., a^N\} \leftarrow \pi(g, \tau_{\text{curr}})$
12:     // evaluation
13:     $v \leftarrow V(g, \tau_{\text{curr}})$
14:     // back-propagation
15:     while $s_{\text{curr}} \neq s$ do
16:         $s_{\text{curr}} \leftarrow$ parent($s_{\text{curr}}, a_{\text{curr}}$)
17:         update $Q, N$ using $v$ with Equation (2)
18:     end while
19: end while
20: // prediction
21: $a^* \leftarrow \arg\max_a N(s, a)$
22: return $a^*$
```

During selection, we follow Silver et al. (2017) and use a variant of PUCT (Rosin, 2011) to balance exploration and exploitation:

$$U(s, a) = c_p \cdot P(a|s) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{1}$$

where c_p is a hyperparameter controlling the exploration rate, $P(a|s)$ is the VLM's prior probability of generating action a in state s, and $N(s, a)$ is the state visit count. We use c_p = 1.0 in our experiments.

During evaluation, we prompt the VLM as a value function using the process described in Section 2.3. During back-propagation, we update the search tree's action-value estimate Q and visitation count N using running average:

$$Q(s, a) \leftarrow Q(s, a) + \frac{v - Q(s, a)}{N(s, a)}; \quad N(s, a) \leftarrow N(s, a) + 1. \tag{2}$$

Algorithm 2 *R*-MCTS Agent

```
Require: VLM policy $\pi(g, \tau)$
Require: VLM value $V(g, \tau)$
Require: environment $\mathcal{T}$
Require: list of tasks to complete $G$
 1: for task $g$ in $G$ do
 2:     // inference loop
 3:     $\tau \leftarrow \{s_0\}$
 4:     tree $\leftarrow \{s_0\}$
 5:     while task not terminated do
 6:         while search budget not exhausted do
 7:             // selection
 8:             $s_t \leftarrow$ selection(tree)
 9:             // expansion
10:             reflections$_\pi \leftarrow$ retrieve(db$_\pi$, $g, s_t$)
11:             $a_t^1, ..., a_t^k \leftarrow \pi(g, \tau,$ reflections$_\pi)$
12:             // simulation
13:             $s_{t+1} \leftarrow \mathcal{T}(s_t, a_t)$
14:             reflections$_V \leftarrow$ retrieve(db$_V$, $g, s_{t+1}$)
15:             $v \leftarrow V(g, \{s_0, a_0, ..., s_{t+1}\},$ reflections$_V)$
16:             // back-propagation
17:             tree $\leftarrow$ update(tree, $s_{t+1}, v$)
18:         end while
19:         $a_t \leftarrow$ best_action(tree)
20:         $s_{t+1} \leftarrow \mathcal{T}(s_t, a_t)$
21:         $\tau \leftarrow \tau \cup \{a_t, s_{t+1}\}$
22:         tree $\leftarrow \{s_{t+1}\}$
23:     end while
24:     $r \leftarrow$ task success/failure
25:     // improvement loop
26:     reflections$_\pi$, reflections$_V \leftarrow$ reflect(VLM, $g, \tau$)
27:     db$_\pi \leftarrow$ update(reflections$_\pi$)
28:     db$_V \leftarrow$ update(reflections$_V$)
29: end for
```

Algorithm 3 Policy Reflection

```
Require: VLM used during search
Require: Terminated task $g$ and trajectory $\tau$
Require: Search tree statistics $Q, V$
Require: Vector database db
1: // error attribution
2: $\tilde{a}_t \leftarrow \arg\max_a$ error$(a|g, \tau)$
3: // contrastive reflection
4: $\hat{o}_{t+1} \leftarrow$ VLM$(g, \{o_0, a_0, ..., o_t, \tilde{a}_t\})$
5: reflection $\leftarrow$ VLM$(g, o_t, \tilde{a}_t, \{o_{t+1}, \hat{o}_{t+1}\})$
6: // memorization
7: key $\leftarrow$ embedding$(g, o_t)$
8: db $\leftarrow$ add(key, reflection)
9: return
```

Algorithm 4 Value Reflection

```
Require: VLM used during search
Require: Terminated task $g$ and trajectory $\tau$
Require: Search tree statistics $Q, V$
Require: Vector database db
1: // error attribution
2: $\tilde{o}_t \leftarrow \arg\max_o$ error$(o|g, \tau)$
3: // contrastive reflection
4: $\hat{a}_t \leftarrow$ VLM$(g, \{o_0, a_0, ..., \tilde{o}_t\})$
5: reflection $\leftarrow$ VLM$(g, a_{t-1}, \tilde{o}_t, \{a_t, \hat{a}_t\})$
6: // memorization
7: key $\leftarrow$ embedding$(g, o_t)$
8: db $\leftarrow$ add(key, reflection)
9: return
```

A.2 R-MCTS PSEUDO CODE

Overall Algorithm We describe our R-MCTS agent in Algorithm 2. For clarity, we also abuse the notation s to represent *observations* in this section, in order to comply with notations used in other works (Świechowski et al., 2022). We omitted details about action selection, expansion, simulation, and back-propagation as they are described in Appendix A.1. We present the main loop in R-MCTS

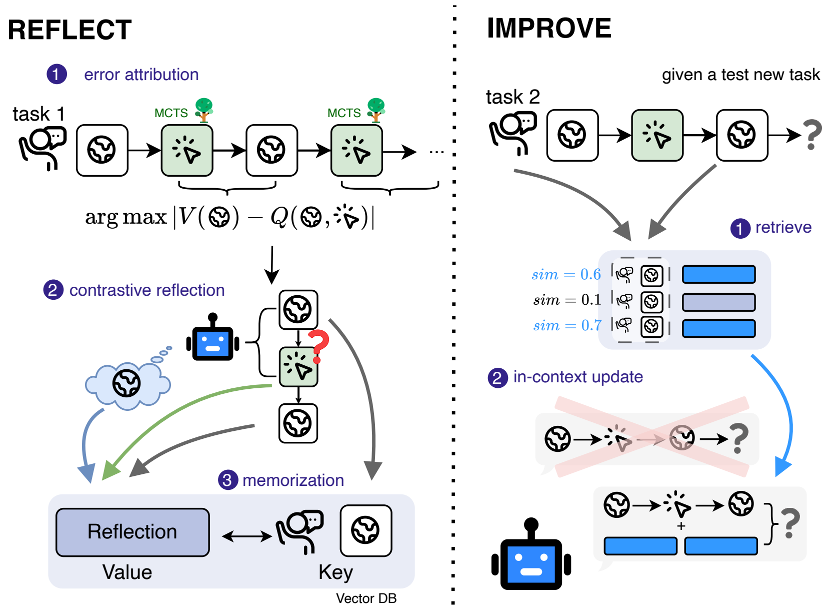


Figure 6: In-context reflection-improvement loop for an *R*-MCTS agent. For brevity, we only present policy reflection, as value reflection is similar. **Left**: after a complete episode, the agent 1) uses search tree statistics to select the most erroneous action $\tilde{a}_t$ from τ to reflect on; 2) prompts the VLM to generate a reflection by contrasting what it *expects* to achieve $\hat{o}_{t+1}$ and what *actually* happens o_{t+1}; 3) embeds the reflection in a vector database. **Right**: to generate an action in a new task, the agent 1) retrieves m most relevant reflections from the database; 2) improves its policy (and value function) using in-context learning.

which consists of 1) an **inference loop** where the agent conducts tree search to find the best action for a given state; and 2) an **improvement loop** where the agent reflects on its past actions, and updates its policy and value function in-context using retrieval for future tasks. We detail the reflection and update process below.

Contrastive Reflection We present the pseudo code for both policy contrastive reflection and value contrastive reflection in Algorithm 3 and Algorithm 4, respectively, and illustrate the reflection-update process in Figure 6. This reflection process is repeated for n_π times per trajectory for the top-n_π erroneous actions during policy reflection, and n_V times for value function reflection. We use $n_\pi = 3$ and $n_V = 1$ in our experiments. We use text-ada-003-small (OpenAI, 2024c) as the embedding model. For clarity, we repeat the error attribution equations in Section 3.1 below:

$$\text{error}_\pi(a_t|\tau) = |V(o_{t+1}) - Q(o_t, a_t)|, \quad \text{error}_V(o_t|\tau) = |V(o_t) - Q(o_{t-1}, a_{t-1})|. \tag{3}$$

Improvement Given a new task g and trajectory $\tau = \{o_0, a_0, ..., o_t\}$, we retrieve the $m = 2$ most relevant reflections from the database, and use them to improve the policy and value function by appending them into the existing context (see Appendix D.2 for the prompt template). The retrieval process is done by computing the cosine similarity between the current task and observation embedding(g, o_t) and the keys stored in the vector database. Since many tasks/webpages are unique, we also set a minimum similarity threshold of 0.25 to only use relevant reflections.

Method	Task Affected by ID Error	Avg. Time per Task	Success Rate
Koh et al. (2024b)	40.00%	33.47 min	45.00%
+Asyncio	40.00%	14.65 min	35.00%
+Caching	35.00%	10.03 min	40.00%
+Action Re-mapping	10.00%	10.10 min	45.00%

Table 6: Ablation studies of our modified browser implementation. Results are evaluated using SEARCH AGENT over 20 randomly sampled task in the Classifieds environment.

B ADDITIONAL DETAILS ABOUT VWA

B.1 WEB BROWSER

Search-augmented agents require frequent backtracking to explore new actions. As a result, we find prior implementations of the *web browser* is inefficient and error prone: 1) all processing is done in a single thread sequentially; 2) every (backtracked) page is re-processed from scratch; and 3) can frequently cause action *element id mismatch* errors since backtracked page may not always be the same as before (e.g., *search history* is stored in the website's database and cannot be reset).

We thus modified the existing web browser used in Koh et al. (2024b), and used the same modified browser for *all methods* in our experiments. Our modifications include:

1. use `asyncio` to parallelize webpage processing (*Asyncio*);
2. implement a caching mechanism to store and retrieve SoM processing output when the same webpage is encountered (*Caching*);
3. when SoM element ids changed after backtracking, we implement a simple heuristic (word overlap of some metadata associated with the element) to re-map the VLM generated action id back to the correct element id (*Action Re-mapping*).

We present a quantitative study of these changes in Table 6.

B.2 VWA EVALUATION

One common error we find with GPT-4o based agents (e.g., using REACT) is to directly return the (correct) item's name and description from (e.g.,) the search result page, while VWA evaluates these tasks by checking if the *final state's URL* is the desired item's URL. However, we believe this is due to ambiguously phrased task intent, such as "Find me the cheapest blue kayak on this site", and "Find the most expensive car from Virginia that is neon green", which does not explicitly mention/require navigating to the item's page to complete.

To this end, we updated these task instructions by appending the sentence: "Finish the task by navigating to that item's page and returning me the url link(s)" to these tasks for clarity. This change affected 93 instances in the Classifieds environment, 0 instance in the Reddit environment, and 34 instances in the Shopping environment. We present a quantitative study of this change in Table 7, and find that all methods/VLMs significantly improved by simply aligning the task intent with the evaluation metric. Unless otherwise specified, we use these updated task intents for all methods and experiments in this work.

C ADDITIONAL ANALYSIS DETAILS

C.1 ERROR ANNOTATION SCHEME

We label errors made by our R-MCTS agent into 4 primary categories (answer formatting error, environmental error, model text understanding, and model image understanding). We find these categories covered most of the error cases. We further sub-divided these categories into 8 sub-error types for a more detailed analysis, and present the result in Figure 7. We detail the error types below.

Method	Model	Success Rate Before	Success Rate After
REACT	GPT-4o-mini	10.67%	14.67%
SEARCH AGENT	GPT-4o-mini	18.67%	22.67%
REACT	GPT-4o	14.67%	27.27%
SEARCH AGENT	GPT-4o	26.67%	38.57%

Table 7: Performance comparison after updating task intents in VWA. We measure the task success before and after the updates for 75 randomly sampled tasks in the Classifieds environment.

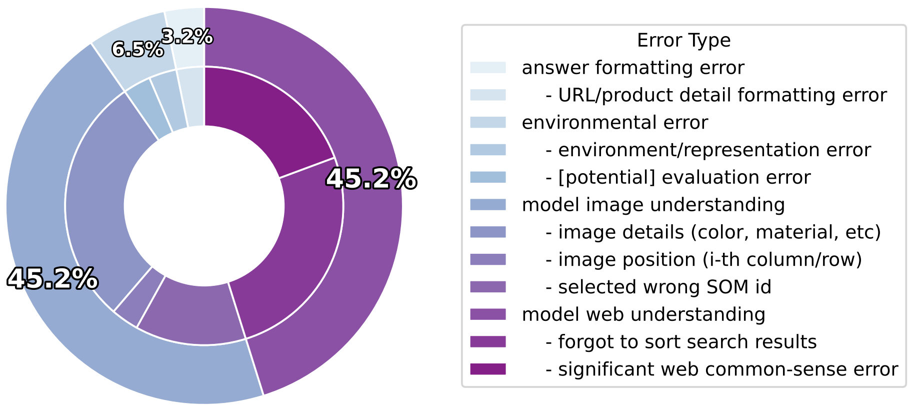


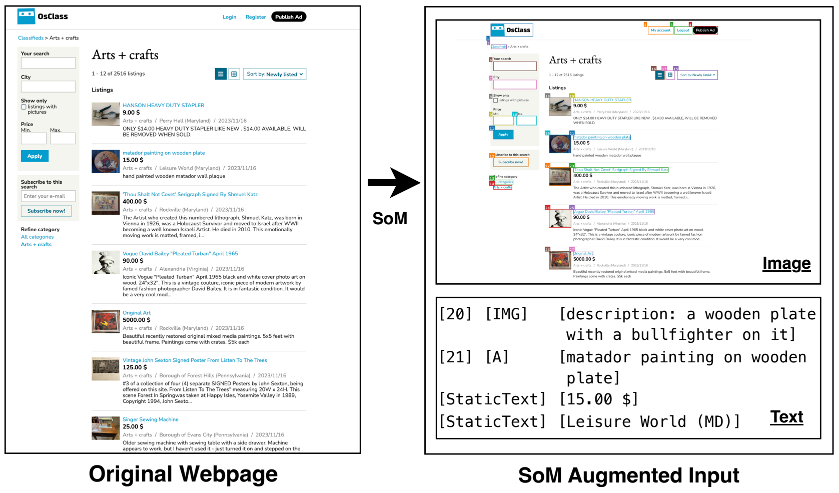
Figure 7: Error analysis for R-MCTS$_{\text{MAD}}$ after manually inspecting the search trees for 80 randomly sampled tasks from the Classfieds environment.

Answer Formatting Answer formatting errors include: 1) returning the item's name/description instead of its URL used for evaluation, or failing to format a range of prices into the instructed format (*URL/product detail formatting error*).

Environmental Errors Environmental errors include: 1) incompatibility issues with certain website functionality (e.g., dropdown menu) with SoM augmentation (*environment/representation error*); and 2) ambiguous task intents leading to more than one potentially correct answer, one of which the model returned (*potential evaluation error*).

Model Image Understanding Model image understanding errors include: 1) failing to correctly identify certain properties (e.g., color, pattern, material, etc.) from an image (*image details*); 2) failing to identify product on the i-th row and/or j-th column (*image position*); and 3) generating an action with element ID not present on the current observation (*selected wrong SoM id*).

Model Web Understanding Model web understanding errors include: 1) forgetting to sort search results and misunderstands the most expensive/cheapest item on the current page is also the most expensive/cheapest item on the entire website (*forgot to sort search results*); and 2) other errors potentially caused by a lack web common-sense knowledge, such as trying to filter for price ranges by typing in the *search bar* instead of using price filters, and returning related items on other pages when specifically instructed to find it on the *current* page (*significant web common-sense error*).



```
[20] [IMG]        [description: a wooden plate
                   with a bullfighter on it]
[21] [A]          [matador painting on wooden
                   plate]
[StaticText] [15.00 $]
[StaticText] [Leisure World (MD)]
```

Figure 8: Example Set-of-Mark augmented webpage used as inputs for VLM agents.

D Additional Prompting Details

D.1 Set-of-Mark Augmentation

We follow Koh et al. (2024a;b) and represent visual webpages using Set-of-Mark (SoM) (Yang et al., 2023b) augmentation. SoM augments a webpage by 1) creating bounding boxes around each interactive element on the page; and 2) adding a unique identifier to each element. We present an example in Figure 8.

D.2 Prompts for R-MCTS Agent

Prompts for Contrastive Reflection We present the prompts to generate an expected observation given a previous state an action in Table 8. We then generate a reflection using use the prompts in Table 9.

Prompts for Improvement We present the prompts used to format the relevant reflections retrieved from the vector database back into model's current context in Table 10.

Prompts for Multi-Agent-Debate We present the prompts to perform multi-agent-debate in Table 11, and Table 12. For simplicity, we present prompts to generate opposing arguments for value estimation Table 11, since prompts to generate supporting arguments is highly similar.

Role	Prompt
System	You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue. Here's the information you'll have: (... some content omitted)
User	OBSERVATION: [] [StaticText] [What are you looking for today?] [5] [INPUT] [] [6] [SELECT] [Select a category] [7] [BUTTON] [Search] [] [StaticText] [Latest Listings] [] [StaticText] [Atlas Powered Audio System w/ Tripod 150.00 $ Music instruments Borough of Red Lion (Pennsylvania) 2023/11/16] (... some content omitted) OBJECTIVE: Find me the cheapest red Toyota. It should be between $3000 to $6000. Finish the task by navigating to that item's page and returning me the url link(s). IMAGES: (1) current page screenshot image_url: data:image/png;base64,iVBORw0KGgoAAAANSUh...
Assistant	Based on the objective, I need to find the cheapest red Toyota priced between $3000 to $6000. (... some content omitted) In summary, the next action I will perform is "'type [5] [red Toyota] [1]"'
User	What do you expect to happen after taking this action? Briefly describe what you think will appear on the webpage after performing the action.
Assistant	After taking this action, I expect the webpage to display search results displaying red Toyotas, (... some content omitted)

Table 8: Prompts used to generate the expected state transition. *image_url* is the base64 encoded SoM augmented web screenshot. Multimodal input in highlighted in orange. Generated response is highlighted in blue.

Role	Prompt
System	You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue. Here's the information you'll have: (... some content omitted)
User	OBSERVATION: [] [StaticText] [What are you looking for today?] [5] [INPUT] [] [6] [SELECT] [Select a category] [7] [BUTTON] [Search] [] [StaticText] [Latest Listings] [] [StaticText] [Atlas Powered Audio System w/ Tripod 150.00 $ Music instruments Borough of Red Lion (Pennsylvania) 2023/11/16] (... some content omitted) OBJECTIVE: Find me the cheapest red Toyota. It should be between $3000 to $6000. Finish the task by navigating to that item's page and returning me the url link(s). IMAGES: (1) current page screenshot image_url: data:image/png;base64,iVBORw0KGgoAAAANSUh...
Assistant	Based on the objective, I need to find the cheapest red Toyota priced between $3000 to $6000. (... some content omitted) In summary, the next action I will perform is ”'type [5] [red Toyota] [1]”' Expectation: After taking this action, I expect the webpage to display search results displaying red Toyotas, (... some content omitted)
User	NEXT OBSERVATION: [596] RootWebArea 'red Toyota - Classifieds' focused: True [630] link 'Classifieds, description: a blue and black logo...' [663] img 'Classifieds, description: a blue and black logo...' [638] link 'My account' [639] link 'Logout' [640] link 'Publish Ad' [647] link 'Classifieds' (... some content omitted) IMAGES: (1) next page screenshot Image URL: data:image/png;base64,iVBORw0KGgoAAA... Is this webpage what you expected? If not, can you conclude anything special about navigating on this website? If you faced the same situation again, what would you do differently at a high level? Do NOT propose any specific actions/answers. Keep your response within 100 words.
Assistant	This webpage does not display the expected car listings but rather includes various (... some content ommitted)

Table 9: Prompt used to generate contrastive reflections given an expected observation highlighted in teal. *image_url* is the base64 encoded SoM augmented web screenshot. Multimodal input in highlighted in orange. Generated response is highlighted in blue.

Role	Prompt
System:	You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue. Here's the information you'll have: (... some content omitted)
User	REFLECTIONS: here are some relevant reflections from other tasks. Note that these reflections may not be directly related to the new task below, but they may provide some useful insights. OBJECTIVE (1): I recall seeing this exact item (... some content omitted) ATTEMPTED ACTION (1): Let's think step-by-step. The objective is to find the most recent post of the item shown in the image, which is a personal watercraft. To find this, I will search for "watercraft" in the search box. (... some content omitted) In summary, the next action I will perform is "'type [5] [watercraft] [1]"' REFLECTION (1): This webpage (... some content omitted). However, it did not directly show the item from the provided image. The search term might have been too broad. (...some content omitted) ##### OBJECTIVE (2): Tell me the name of the lister with (... some content omitted) ATTEMPTED ACTION (2): Let's think step-by-step. The objective is to find the name of the lister with the most expensive green vehicle from West Virginia. (...some content ommitted) In summary, the next action I will perform is "'click [60]"' REFLECTION (2): Yes, the webpage is what I expected, displaying listings specifically from West Virginia. However, it appears that filtering by color (green) is not directly possible. (... some content omitted) !IMPORTANT! Below is the task you need to solve. Please read the user's intent and input images (if any) carefully. OBSERVATION: [1] [IMG] [Classifieds, description: a blue and black logo with the words ohm, url: http://coffee.cs.columbia.edu:57981...] [2] [A] [My account] [3] [A] [Logout] [4] [A] [Publish Ad] [] [StaticText] [What are you looking for today?] (... some content omitted) URL: http://classifieds.com/ OBJECTIVE: How many yellow or blue motorcycles in total were posted on 25th October 2023? PREVIOUS ACTION: None NOTE: Remember that you should consider user's intent and reflections (if applicable) to better plan the next action. IMAGES: (1) current page screenshot Image URL: data:image/png;base64,iVBORw0KGgoAAAAN...
Assistant	Let us think step by step. The objective is to find the total number of yellow or blue motorcycles posted on 25th October 2023. From the reflections (... some content omitted)

Table 10: Prompt used to generate an action given retrieved reflections, which is highlighted in teal. *image_url* is the base64 encoded SoM augmented web screenshot. Multimodal input in highlighted in orange. Generated response is highlighted in blue. For brevity, we present prompts for trajectories with only one action executed so far.

Role	Prompt
System	You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue. Here's the information you'll have: (... some content omitted)
User	User Intent: Find me the cheapest red Toyota. It should be between $3000 to $6000. Finish the task by navigating to that item's page and returning me the url link(s). IMAGES: (1) start page screenshot Image URL: data:image/png;base64,iVBORw0KGgoAAAAN...
Assistant	type [5] [red Toyota] where [5] is [INPUT] element with content []
User	Last page URL=http://coffee.cs.columbia.edu:57981/index.php?page=... IMAGES: (1) last page screenshot. Image URL: data:image/png;base64,iVBORw0KGgoAAA... Now, it is your turn to evaluate the success/failure of the agent's execution so far. (... some content omitted) And your goal is to find the most convincing evidence why the agent is NOT on the right track to complete the task. (... some content omitted) Keep your response within 100 words.
Assistant	The agent has not successfully completed the task. The user's intent was to find the cheapest red Toyota priced between $3000 and $6000. The search results include vehicles that are outside the price range and have the wrong color (... some content omitted)

Table 11: Prompts used to generate opposing arguments for value estimation. *image_url* is the base64 encoded SoM augmented web screenshot. Multimodal input in highlighted in orange. Generated response is highlighted in blue. For brevity, we present prompts for trajectories with only one action executed so far.

Role	Prompt
System	You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue. Here's the information you'll have: (... some content omitted)
User	User Intent: Find me the cheapest red Toyota. It should be between \$3000 to \$6000. Finish the task by navigating to that item's page and returning me the url link(s). IMAGES: (1) start page screenshot Image URL: data:image/png;base64,iVBORw0KGgoAAAAN...
Assistant	type [5] [red Toyota] where [5] is [INPUT] element with content []
User	Last page URL=http://coffee.cs.columbia.edu:57981/index.php?page=... IMAGES: (1) last page screenshot. Image URL: data:image/png;base64,iVBORw0KGgoAAA... Now, it is your turn to evaluate whether the agent's actions so far are successfully aligned with the user's intent. (... some content omitted) To better verify if user's intent is fulfilled correctly, you may find the following opinions helpful: Reasons why the agent is NOT on the right track: The agent has not successfully completed the task. The user's intent was to find the cheapest red Toyota priced between \$3000 and \$6000. The search results include vehicles that are outside the price range and have the wrong color (... some content omitted) Reasons why the agent is on the right track: The agent has successfully navigated to the search results page for "red Toyota". Specifically, the "2007 Toyota Yaris" at \$3000 and the "2007 Toyota Corolla" at \$6500 are relevant. (...some content omitted) Note that these opinions may or may NOT be correct. You should make your own judgment based on the user's intent, the observations, and the agent's executed actions so far. To make a final decision, choose one of the following status codes and provide your thought process. STATUS CODES: A. The agent's last action is "'stop"' and it contains the correct answer (...some content omitted) B. The agent is very close to finishing the task (...some content omitted) C. The agent still needs a few more actions (...some content omitted) D. (...some content omitted) E. (...some content omitted)
Assistant	Thoughts: The agent has successfully navigated to the results page for "red Toyota". However, it hasn't identified the cheapest (...some content omitted) STATUS CODE: C

Table 12: Prompts used to generate the final judgement for value estimation, given opposing arguments highlighted in teal. *image_url* is the base64 encoded SoM augmented web screenshot. Multimodal input in highlighted in orange. Generated response is highlighted in blue. For brevity, we present prompts for trajectories with only one action executed so far.