
Let’s Think in Two Steps: Mitigating Agreement Bias in MLLMs with Self-Grounded Verification

Anonymous Author(s)

Affiliation

Address

email

Abstract

Verifiers — functions assigning rewards to agent behavior — have been key for AI progress in domains such as math and board games. However, extending these gains to domains without clear-cut success criteria (e.g., computer use) remains a challenge: while humans can recognize suitable outcomes, translating this intuition into scalable rules is nontrivial. Multimodal Large Language Models (MLLMs) emerge as a promising solution, given their world knowledge, human-preference alignment, and reasoning skills. We evaluate MLLMs as verifiers of agent trajectories across web navigation, computer use, and robotic manipulation, and identify a critical limitation: agreement bias, a strong tendency for MLLMs to favor information in their context window, often generating chains of thought to rationalize flawed behavior. This bias is pervasive across models, resilient to test-time scaling, and can impact existing methods relying on MLLMs as evaluators (e.g., data filtering). Notably, it occurs despite MLLMs showing strong, human-aligned priors on desired behavior. To address this, we propose Self-Grounded Verification (SGV), a lightweight method that enables more effective use of MLLMs’ knowledge and reasoning by harnessing their own sampling mechanisms via unconditional and conditional generation. SGV operates in two steps: first, the MLLM is elicited to retrieve broad priors about task completion, independent of the data under evaluation. Then, conditioned on self-generated priors, it reasons over and evaluates a candidate trajectory. Enhanced with SGV, MLLM verifiers show gains of up to 20 percentage points in accuracy and failure detection rates, and can perform real-time supervision of heterogeneous agents, boosting task completion of a GUI specialist in OSWorld, a diffusion policy in robomimic, and a ReAct agent in VisualWebArena — setting a new state of the art on the benchmark, surpassing the previous best by 48%. Finally, we release an updated version of the (Visual)WebArena benchmark featuring more human-aligned evaluators, improved environment parallelization with higher execution fidelity, and runtime speedup of over 10x. Our code, data, and additional visualizations are available at <https://self-grounded-verification.github.io>.

1 Introduction

Several breakthroughs in artificial intelligence can be viewed from the lens of search guided by verifiers—functions assigning rewards to agent behavior aligned with desired criteria. Notable examples include seminal work in games such as Go [40] and Chess [5], where search and learning are guided by 0/1 rewards tied to the game’s final outcome, and recent advancements in large reasoning models (LRMs), leveraging formal verifiers in code and math [37, 58]. Reinforcing this perspective, recent studies highlight the importance of verifiers in enhancing the efficiency and robustness of training pipelines, particularly in reasoning [35, 8] and preference alignment [44].

However, while domains such as mathematics, programming, and board games benefit from relatively well-defined criteria to evaluate agent behavior, such clarity diminishes in open-ended settings.

Preprint.

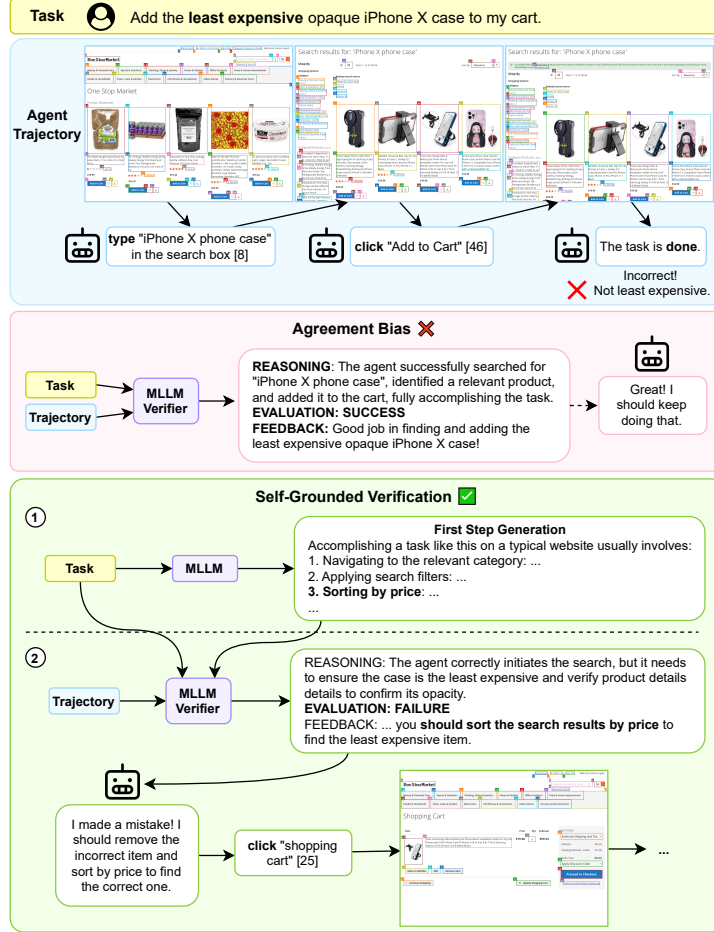


Figure 1: **Top**: example of a task in VisualWebArena and a corresponding agent trajectory. **Middle**: an MLLM verifier validates and reinforce flawed agent behavior, generating reasoning to rationalize incorrect judgments. **Bottom**: Self-Grounded Verification leads to more accurate verification and enables the agent to backtrack.

Evaluation in such scenarios often requires nuanced criteria and reasoning over possibly long sequences of multimodal inputs. Consider, for example, evaluating correctness for the trajectory produced by a digital agent tasked with adding “*the least expensive opaque case*” for a mobile phone to a shopping cart, as illustrated in Figure 1 (top). Should the agent sort products by prices, or perform an advanced search? Should it inspect product details to confirm specific attributes? What is “opaque enough”? Although humans can intuitively recognize satisfactory outcomes in such scenarios, formalizing this intuition into precise and scalable rules remains a challenge.

Multimodal Large Language Models (MLLMs) emerge as a promising solution to bridge this gap. With vast world knowledge, human-preference alignment, and large context windows, MLLMs hold the potential to serve as general-purpose verifiers, capable of handling inputs and producing rewards in multiple modalities. In this work, we investigate this potential by evaluating MLLMs as verifiers of agent behavior in a diverse range of open-ended tasks. We consider two digital agent environments — VisualWebArena [16] and OSWorld [51] — and the long-horizon robot manipulation suite robomimic [25]. Together, they span over 1,200 tasks across varied domains, where evaluating agent behavior demands nuanced criteria and multimodal reasoning, offering a compelling testbed for studying MLLM verifiers.

We identify a critical limitation that undermines the reliability of MLLMs in this role: a strong tendency to favorably evaluate agent behavior, a phenomenon we call agreement bias. As shown in Figure 1 (middle), an MLLM verifier validates flawed agent behavior and even produces chains-of-thought (CoT) to rationalize its incorrect judgments. We show that this bias is pervasive across model families, and persists despite the use of established test-time scaling techniques, training for

reasoning, and instructions with task-specific evaluation criteria. Quantitatively, it manifests as: high ratio of false positives, positive bias and skewness, answers tilted toward positive judgments, and low ($<50\%$) probability of detecting failures. We argue that agreement bias can negatively impact several pipelines where MLLMs play evaluative roles. Particularly, it can limit the effectiveness of MLLMs to serve as curators of data for finetuning and inference-time refinement; as providers of feedback and rewards to guide search; and as judges and monitors of agent behavior, where failure detection is essential for balanced assessments and to prevent harmful outcomes.

We observe that such failures occur despite MLLMs possessing strong, human-aligned priors on desired agent behavior, suggesting a retrieval bottleneck [1, 2] not resolved with major test-time scaling techniques, nor training for reasoning. To address this issue, we introduce Self-Grounded Verification (SGV), a simple yet effective method that significantly improves the reliability of MLLM-based verification (Figure 1, bottom). At its core, SGV harnesses MLLMs’ own sampling mechanisms by modulating unconditional and conditional generation to effectively leverage their world knowledge, alignment, and reasoning capabilities. SGV operates in two steps: first, the model is elicited to retrieve a broad set of priors associated with successful task completion, conditioned only on minimal context to frame the task. In the second step, conditioned on its self-generated priors, the model then evaluates a candidate trajectory. SGV leads to more balanced and accurate verification across environments and policies, while introducing minimal token overhead and being easy to integrate into pipelines relying on MLLM verifiers.

To demonstrate SGV’s effectiveness and versatility, we evaluate MLLM verifier performance in key settings: automatic evaluation of agent trajectories with fine-grained performance analysis of the verifications themselves, self-refinement pipelines, and online supervision to guide agents toward task completion. On over 1,200 agent trajectories from VisualWebArena and OSWorld, SGV improves verification performance of several MLLMs and LRMs, leading to gains of up to 17 percentage points (pp) in accuracy and up to 20 pp in true-negative identification, outperforming baselines leveraging established test-time scaling techniques and even instructions with access to privileged information. We then show that SGV can be paired with ReAct agents [56] to guide their decision-making, achieving a 6pp gain ($\approx 13\%$ relative) on VisualWebArena, establishing a new state of the art by outperforming the previous best by 17 pp ($\approx 50\%$ relative), with mild increase in token usage. Similarly, our verifiers guide the GUI-specialist UI-TARS-1.5 [32] to improve by 5 pp ($\approx 22\%$ relative) in OSWorld, and a diffusion policy to outperform oracle supervision by 8 pp ($\approx 33\%$ relative) on robomimic’s tool-hang task. Finally, as a byproduct of our work, we release an updated version of the (Visual)WebArena benchmark featuring more human-aligned evaluators, improved environment parallelization with higher execution fidelity, runtime speedup of over 10x, and strong Agent baselines, thereby facilitating the development of digital agents.

2 Related Work

MLLMs as Evaluators. MLLMs have been employed as evaluators of model outputs in various scenarios. This work focuses on their application in multimodal and environment interaction settings. MLLMs have been used to score and filter agent trajectories for subsequent use in both finetuning [45, 29, 24] and inference-time refinements such as prompts and reflections [48, 60, 39, 34]. They have also served as a source of online feedback, in the form of natural language “critiques” [30], real-valued scores to rank action proposals in search [18, 54], and rewards for training [43]. These applications appear under various names — MLLMs as judges, critics, reward models, and value estimators. We propose a unified perspective based on the use of multimodal rewards derived from MLLM verifiers and identify agreement bias as a key limitation that can undermine MLLMs’ effectiveness across these applications. Our SGV method improves the reliability of MLLM verifiers, and can be easily integrated into applications relying on their evaluations.

Scaling Test-Time Compute. Test-time scaling (TTS) has emerged as a major paradigm to improve model performance without increasing the number of parameters [41, 50]. Early work [49, 19] demonstrates that prompting LLMs to generate intermediate “chains-of-thought” leads to substantial improvements in reasoning-oriented tasks. Orthogonal approaches scale test-time compute through sampling [4] and search [62, 55, 12], where multiple generations are selected through heuristics [4] or reward models [36, 22]. Recent work leverages sampling, reinforcement learning, and formal verifiers to train (M)LLMs that autonomously generate reasoning traces [9, 27, 10, 11], achieving impressive results in challenging benchmarks [3].

AI Agents. There is growing interest in building AI agents to act in various environments, including the web [16], mobile phones [21], and computer systems [23]. In this field, the work most related to ours is [29], where a GPT-4V-based evaluator is prompted with benchmark-specific rubrics and used to identify successful agent trajectories in WebArena [63], which are subsequently utilized for Reflexion [39] and behavior cloning. Following works employ a similar evaluator to build value functions to guide tree search [59, 17], and filter trajectories to generate prompts and reflections to improve agent performance on subsequent executions [48, 59].

3 Problem Setup

3.1 Preliminaries and Definitions

Agents and Multimodal Verifiers. We study functions that approximate human judgment of agent behavior in interactive environments. Specifically, an agent is tasked to complete a task $q \in \mathcal{Q}$ through a series of actions a_t . Given a history of environment states $s_{r:t} \equiv [s_r, \dots, s_t]$, the agent generates and executes an action sampled from a policy π : $a_t = \pi(s_{r:t}, q)$, leading to a new state s_{t+1} . The repetition of this process yields a **trajectory** $\tau_{r:t} \equiv (s_r, a_r, \dots, s_t, a_t)$ that, along with the task q , serves as the basis for evaluating agent performance.

To evaluate trajectory-task pairs $(q, \tau_{r:t})$, humans can process and produce information in multiple modalities motivating our definition of a verifier and the scope of tasks considered. We define a **multimodal verifier** as a function $r : \mathcal{T} \times \mathcal{Q} \rightarrow \mathbb{R} \times (\mathcal{V} \cup \{\emptyset\})$, $r \in \mathcal{R}$, that maps trajectory-task pairs to rewards consisting of a real-valued score and optional outputs in other modalities from $\mathcal{V}$. These multimodal aspects are reflected in our settings. A task q can be represented as images, text, or both, states s_t as screenshots or DOM trees, actions a_t as text (e.g. <type opaque phone>) or images from URLs, and verifier outputs as 0/1 scalars or natural language feedback flagging errors in off-track trajectories.

Oracle Verifiers. In the benchmarks we study, instantiations of r are obtained via human-written scripts using privileged information about the task and environment states, and produce 0/1 rewards indicating task completion. We treat these as closely aligned to human judgments, and refer to them as **oracle** verifiers. We note, however, that these functions are not perfect, and discuss limitations in Section 3.2 and Section 4.

3.2 MLLM Verifiers, Agreement Bias, and Self-Grounded Verification

Scripted evaluation for open-ended tasks is imperfect and hard to scale, motivating the use of MLLMs as an alternative. The conventional approach to build r with an MLLM is to prompt it with the task q , trajectory $\tau_{r:t}$, and additional context C that can include rubrics for evaluation and instructions for reasoning steps:

$$r_{MLLM}(q, \tau_t, C) = h \left(\prod_{i=1}^n P(y_i \mid y_{<i}, q, \tau_t, C) \right)$$

where $y = y_1, \dots, y_n$ are MLLM outputs, and h is a function mapping them to rewards — e.g., a regex that maps model completions to 0/1 scores.

We observe, however, that this approach is subject to a failure mode we call **agreement bias**. Specifically, MLLMs display a tendency to overly validate agent behavior, judging flawed trajectories as aligned with the task q despite the use of carefully crafted instructions C , established test-time scaling techniques, and choice of decoding algorithms. This bias directly degrades the quality of r_{MLLM} with potential negative effects on several of the existing applications that make direct and indirect uses of r_{MLLM} .

Notably, agreement bias occurs despite models exhibiting strong, human-aligned priors on desired characteristics for agent behavior (e.g., first-step generations in Figure 1 and Figure 5). This suggests a bottleneck in knowledge extraction that is not resolved with major techniques to elicit reasoning, nor in models explicitly trained to produce intermediate reasoning steps. This observation aligns with findings in LLM literature [2], [38] which point to root causes in pretraining data composition and inherent limitations stemming from RLHF. Addressing these issues through changes to pretraining corpora or retraining large models remains unclear and prohibitively costly, highlighting the need for alternative solutions.

Motivated by these observations, we propose **Self-Grounded Verification (SGV)** (Figure 1 (middle)), a simple yet effective method that substantially improves the evaluation performance of MLLMs. We hypothesize that MLLMs can generally generate priors that are well-aligned with human preferences, and can serve as impartial references to ground the verification. SGV operates in two steps: first, the MLLM is elicited to retrieve a broad set of priors $\hat{k}_q$ associated with the successful completion of the task q , conditioned only on the data needed to frame the task:

$$\hat{k}_q = g \left(\prod_{i=1}^n P(y_i \mid y_{<i}, s_{0:t}, C, q) \right),$$

where $s_{0:t}$ are states needed to define the task (e.g., an initial screenshot) and g is a function to select a completion. In the second step, the MLLM evaluates the trajectory conditioned on the first-step generation:

$$r_{\text{SGV}}(\tau_t, C, q) = h \left(\prod_{i=1}^n P(y_i \mid y_{<i}, \hat{k}_q, \tau_t, C, q) \right).$$

Intuitively, by modulating (un)conditional generation, SGV harnesses MLLMs’ own sampling mechanisms to enable more effective use of their knowledge and reasoning capabilities. Conditioning only on essential information in the first step encourages the model to fully explore its probability distribution, extracting impartial information pertinent to the task at hand and independent of the data under evaluation. In the second step, the MLLM evaluates a candidate trajectory by sampling from a conditional distribution induced by its own priors, for which we expect a higher likelihood of accurate verification.

Practical Implementation. SGV introduces low compute overhead and generally improves performance without complex prompt engineering, making it simple to integrate into pipelines where MLLMs play evaluative roles. For instance, adding the first step with a single-line instruction to produce domain knowledge generally helps, with little to no change to existing prompts. Nonetheless, we observe the following patterns that may inform general use: **1)** generating broader, more encompassing priors in the first step tends to be superior; **2)** producing priors in a single, monolithic step degrades performance, as models tend to anchor them to the data under evaluation; **3)** some models benefit when the second step includes an instruction to compare against their priors — something SGV inherently enables — although a standard CoT instruction typically suffices. See Sections B and B.6 for prompts and ablations, and our codebase for more examples.

3.3 Evaluating MLLM Verifiers

To properly assess verifiers and ensure our findings are robust and generalizable, we consider several factors in our experimental setup.

Environment Diversity and Multimodality. We consider benchmarks that require nuanced verification criteria, involve multimodal reasoning, and collectively span a diverse range of settings. VisualWebArena (VWA) [16] emulates a web browser and spans 910 tasks, many of which combine text and image instructions (e.g., “Find a product like this” + *<image>*). OSWorld [51] emulates a computer system, with 369 tasks involving widely-used web and desktop applications in both single- and multi-app workflows. Finally, robomimic [25] provides long-horizon robot manipulation tasks, and we focus on *tool hang*, the most challenging among them, which consists of two subtasks (1) inserting an L-shaped pencil into a base and (2) hanging a wrench on it. Section H illustrates representative tasks and trajectories.

Annotations and Scalability. Human annotation, while ideal, is prohibitively expensive at the scale of our experiments. Similar to previous work [29], we therefore rely on oracle verifiers provided by the benchmarks as a proxy for human judgment. However, we observed several issues in the oracles included in the VisualWebArena benchmark—an observation also noted in concurrent work [26], which compares multiple evaluators against human annotations. To establish a more robust and trustworthy reference, we addressed non-ambiguous issues in the VisualWebArena oracle verifiers, including bugs in string parsing, mismatches between task intent and oracle requirements, and incorrect annotations. To validate the effectiveness and impartiality of our changes, we evaluate the updated oracles against external human annotations obtained from [26]. As shown in Table 7, the revised oracles exhibit near-perfect agreement with human preferences, confirming the impartiality and effectiveness of the proposed updates. Details are documented in our codebase and Section G.5.

Agents and Trajectory Data Quality. Weak agents and buggy environments often produce trajectories that are trivial to verify. For instance, some trajectories in [26] exhibit “page not found” errors or infinite action loops, which can artificially inflate verifier performance. Moreover, incorporating trajectories generated by diverse models and methods is crucial for ensuring the generalizability of our findings. Accordingly, we fixed several bugs in the (Visual)WebArena environments (Section G.5), developed agents that yield a balanced mix of successes and failures, and considered agents built from different methods. To generate trajectories in VisualWebArena, we build a prompt-based agent [56] equipped with a text-based memory that prevents looping behavior and enables backtracking when paired with a verifier. For OSWorld, we employ the GUI-Specialist Agent UI-TARS-1.5-7B [32] which, at the time of this work, is the best-performing open-source model on the benchmark. For robomimic, we train a diffusion policy with 200 expert rollouts from human demonstrations collected by [61].

Applications of Verifiers and Choice of Experiments. To characterize the potential benefits and risks of MLLM verifiers, we present a unified perspective of several applications based on how they utilize r and are therefore affected by its quality. We categorize them as falling into *offline* and *online* settings. The *offline* setting covers cases where r is applied to trajectories post hoc. A canonical example is the evaluation of agent performance, where a set of trajectories is mapped to real-valued scores reflecting task completion. Composite applications include pipelines that use r to filter trajectories to improve subsequent performance. This includes finetuning (a.k.a behavior cloning), as well as variants of Reflexion [39] that builds memory for subsequent executions—e.g., reflections on past mistakes [39], synthesizing tools [48], or abstractions [34]. The *online* setting covers cases where r is used to influence action distribution during task execution. An example is online supervision, where r maps trajectories to scalars or text rewards to steer agents toward task completion. Composite applications include using rewards to update policy parameters, or to rank samples of trajectory-action pairs to guide (tree) search.

MLLMs have been used to approximate r in several of these applications. While exploring this full range is infeasible, we focus on three representative cases: offline evaluation of agent trajectories, Reflexion, and online supervision. Our choice for these applications is motivated by the following considerations: (1) they are of direct practical interest and are often components for other pipelines; (2) they exhibit fewer confounding factors than alternatives such as behavior cloning or RL; and (3) performance on them can yield cleaner and informative signals for broader applicability. For instance, if r_{MLLM} produces a high number of false positives in trajectory evaluation, subsequent finetuning or self-refinement based on those trajectories is likely to be negatively affected.

Quantitative Metrics. Verifiers should negatively (positively) evaluate trajectories marked as failures (success) by humans or proxies to them. Evaluations that are consistently biased toward one side are likely undesirable. Accordingly, we consider several metrics that capture the degree of alignment—or misalignment—in MLLM responses. Following [52], we consider **(1) Signed bias:** $\text{Bias}(\hat{\theta}) = \frac{1}{n} \sum_{i=1}^n (\mathbb{E}[\hat{\theta}_i] - \theta_i)$, and **(2) distance skewness** $dSkew_n(X) = 1 - \frac{\sum_{i,j} \|x_i - x_j\|}{\sum_{i,j} \|x_i + x_j - 2\gamma\|}$. These two are summary statistics reflecting the distribution of MLLM responses, where positive value means the Verifier consistently evaluate individual trajectories more positively than oracles.

(3) True Positive Rate (TPR) and True Negative Rate (TNR):

$$\text{T?R}(c) = \frac{\sum_{i=1}^n \mathbf{1}(r_{i,\text{MLLM}} = c \wedge r_{i,\text{oracle}} = c)}{\sum_{i=1}^n \mathbf{1}(r_{i,\text{oracle}} = c)} \approx \hat{P}(r_{i,\text{MLLM}} = c | r_{\text{oracle}} = c) \quad c \in \{0, 1\}.$$

as well as $\text{Accuracy} = (1 - \text{SuccessRate}) \times \text{TNR} + \text{SuccessRate} \times \text{TPR}$, which serves as an *auxiliary* metric reflecting the TPR and TNR tradeoff to support interpretation and comparison.

While we consider multiple metrics for robustness, we highlight the practical importance of statistics that directly relates to a verifier’s core role—identifying flawed behavior and provide feedback to improve agent performance. TNR and TPR are empirical estimates of the conditional probabilities that directly relates to this purpose and are therefore important indicators for a verifier’s effectiveness. Verifiers are typically called multiple times during an application, and low values for such probabilities can be highly detrimental—especially a low probability of flagging failures as failures (low TNR), as we further discuss in Section 5.

Ultimately, the value of a verifier should also be measured by its effectiveness in downstream applications. If a verifier provides effective feedback, this should manifest as improved agent performance. Therefore, for downstream applications we compare task completion rates of base Agents with and without MLLM Verifiers.

Finally, we note that: (i) TNR satisfies $TNR = 1 - \text{False Positive Rate}$, and we use a “high number of false positives” interchangeably with a “low TNR.”; (ii) Section F reports the explicit distribution of MLLM responses under various ablations, which explicitly illustrates tilts towards positive/negative directions.

Baselines and Ablations. To probe limitations of MLLM verifiers and establish strong baselines, we consider established methods, including chain-of-thought (CoT) [49], majority voting [47], set-of-marks prompting [53], and reasoning models. For VWA, we also consider the approach proposed in Pan et al. [29], where we additionally provide benchmark-specific rubrics on how to evaluate the tasks. In all cases, MLLMs are given complete trajectories represented as sequences of screenshot-action pairs and asked to label them on a ternary Likert scale: SUCCESS, PARTIAL SUCCESS, or FAILURE, mapped to [1, 0, 0] to align with oracle scores. Sections B.6, F and G.2) ablates on all such choices, including model size, family, SGV design choices, trajectory length, and prompt/scoring templates. Additional experimental details and all prompts are provided in Sections A and B.

4 Offline Evaluation of Agent Trajectories

Table 1 and Table 2 show the performance of verifiers built with a range of models and test-time scaling techniques. Trajectories are based Gemini-2.5-Flash on VisualWebArena and with UI-Tars-1.5 in OSWorld. The average success rate is 43%. Other variations are provided in the appendix. The following provides a discussion of the main findings.

Table 1: Performance of MLLM verifiers on evaluation of digital agent trajectories. (a) Agreement bias: MLLMs display strong tendency to overly validate agent behavior. (b) SGV improves performance across all models and metrics.

	CoT-based Verifier					Thinking		Weaker Models	
	Gemini 2.5	GPT 4.1	GPT4o	Llama4	Qwen2.5	GPT-o1	Gemini Thinking	Gemini 2.0	GPT4o-mini
Acc (No SGV)	65	66	61	62	61	67	67	52	50
TNR (No SGV)	47	50	52	47	54	62	51	41	38
TPR (No SGV)	89	86	85	78	79	79	87	80	79
Skew (No SGV)	20	16	25	10	22	16	16	32	33
Bias (No SGV)	25	22	30	14	28	22	22	37	39
Acc (SGV)	74	75	74	68	66	75	72	64	64
TNR (SGV)	70	73	75	63	63	77	66	65	70
TPR (SGV)	81	78	74	73	74	72	80	63	56
Skew (SGV)	5	2	6	1	13	3	4	7	8
Bias (SGV)	9	6	11	5	20	8	10	15	17
Delta Acc	9	9	13	6	5	8	5	12	14
Delta Skew	-15	-14	-19	-9	-9	-13	-12	-25	-25
Delta Bias	-16	-16	-19	-9	-8	-14	-12	-22	-22

Pervasiveness and Resilience of Agreement Bias. MLLMs display a strong tendency to overly validate agent behavior. This manifests as a high ratio of false positives (low 49–54% TNR), responses tilted towards positive evaluations (strictly positive bias and skewness), a low probability of flagging failures (in some cases, below 50%), and reduced aggregate metrics such as accuracy and precision. As shown in Table 2 (row 3), adding a CoT step yields only modest improvements in such patterns. This limitation persists even when majority voting is applied (row 4) and task-specific evaluation criteria are provided (row 5). A key reason behind these results is illustrated in Figure 5 and Figure 6: MLLMs often generate reasoning that rationalizes flawed trajectories and their wrong judgments. Sampling fails to mitigate this issue and can even exacerbate it, as higher temperatures can increase the likelihood of hallucinations and spurious correlations. Finally, agreement bias occurs when evaluating trajectories of both strong and weak agents, as illustrated by the high number of false positives across scenarios of high and low success rates.

Agreement and Self-Bias. Notably, agreement bias occurs despite inputs including multimodal, external, information produced by the environment, and generations from models of distinct families and sizes. This makes it a separate phenomenon from what is known in the LLM literature as *self-bias*,

Table 2: Impact of test-time scaling on performance of MLLM-based verifiers in trajectory evaluation. Oracle success rates in VisualWebArena and OSWorld are 46% and 22%, respectively.

#	Method	VisualWebArena			OSWorld		
		TPR (%)	TNR (%)	Acc. (%)	TPR (%)	TNR (%)	Acc. (%)
1	No CoT, Binary	80	49	62	97	57	64
2	No CoT	84	52	65	97	57	66
3	CoT	89	54	69	97	59	65
4	CoT + Majority	90	52	69	94	61	69
5	Pan et al. [29]	78	59	67	–	–	–
6	Thinking	84	65	73	94	74	79
7	Thinking + Majority	88	61	73	97	71	79
8	SGV	72	71	73	96	70	76
9	SGV + CoT	75	77	76	89	80	82
10	SGV + Thinking	75	83	80	90	83	84

where LLMs tend to favor text generated by themselves or by stronger models [52, 42, 31], and for which injection of external information is a typical remedy. In our settings, inputs are externally produced by the environment, including screenshots augmented with Set-of-Mark prompting [53], model generations parsed into actions by the environment, semantic information about objects that agents interact with. Moreover, agreement bias occurs for trajectories produced by policies based on models quite distinct from the ones used for verification, such as the GUI-specialist UI-TARS, ReAct agents built from distinct model families and sizes, and a diffusion policy (Section 5).

Adverse Impacts of Agreement Bias. The effects of agreement bias can extend beyond imprecise evaluation of agent performance. A quantitative demonstration is deferred to Section 5, with a qualitative example shown in Figure 5. For the task “*Buy the cheapest <product> from <category> with <attribute>*,” a trajectory where the agent searches for “<product>,” clicks the first result, and adds it to the cart, is deemed successful by MLLM verifiers, despite omitting essential steps for generalization, such as filtering, sorting, and finishing the purchase. Selecting such trajectories for finetuning or test-time refinements is likely to be suboptimal, and risks reinforcing brittle behavior [52].

Effectiveness of SGV. Applying SGV substantially improves verification performance across models and benchmarks, increasing TNR by up to 20 percentage points, overall accuracy by 7–11 points, and yielding responses more closely aligned with human evaluations (i.e., reduced bias and skewness). Notably, the inclusion of a CoT step proves more effective when coupled with SGV. As illustrated in Figures 5 and 6, SGV leads to more grounded and unbiased reasoning steps, improving its overall effect. Moreover, SGV exhibits superior performance to handwritten instructions with task-specific evaluation criteria (Table 2, row 5). This indicates that SGV enables models to generate completions for conditioning themselves that are on par with, or even surpass, human-written rubrics, offering a more scalable alternative.

Training for Reasoning and SGV. LRMs tend to display lower agreement bias (row 6 of Table 2), aligning with findings in concurrent work suggesting long-CoT as a form of mitigating self-bias in LLMs [7]. More remarkably, we observe that (i) SGV enables non-reasoning models to match the performance of their reasoning counterparts, and (ii) SGV boosts performance of reasoning models by up to 7 pp — a perhaps surprising result, given that these models are explicitly trained to produce intermediate traces and such interventions can degrade performance [28]. These results align with our intuitions of retrieval bottlenecks and the findings and conjectures in [1], raising interesting questions on the potential benefits of including an SGV step and pretraining data rewriting [2] into training pipelines aimed at reasoning.

Need for Fine-Grained Classification Metrics. These results highlight the importance of reporting direct metrics of both false positives and false negatives in works proposing MLLMs in evaluative roles and artifacts produced by them. Reporting only single metrics like accuracy and recall can be misleading. For example, the 97% TPR observed in OSWorld masks the fact that the verifier produces a high number of false positives, which can contaminate filtered datasets or undermine the verifier’s effectiveness in online supervision.

Challenges in Open-Ended Verification and Stricter Verifiers. SGV-enhanced verifiers exhibit lower TPR, which stems from two main causes: disagreements with overly permissive evaluation

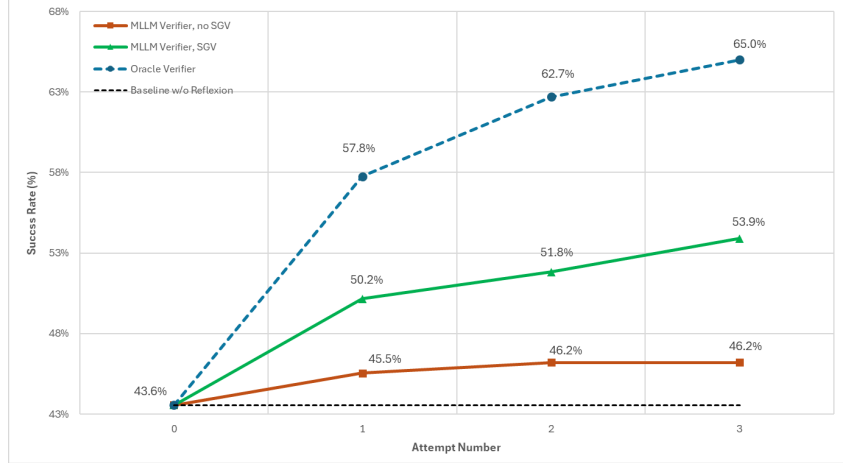


Figure 2: Results of using an MLLM as a verifier within a Reflexion pipeline in VisualWebArena. A conventional MLLM verifier fails to deliver improvements due to its inability to reliably identify flawed behavior, whereas SVG improves success rates.

scripts on simplistic tasks, and overly demanding judgments. Specifically, evaluation in these digital benchmarks comes from hard-coded rules based on a subset of states. This means that trajectories such as in Figure 6 for a task “Buy the cheapest <product>,” where an agent searches for “<product>,” clicks on the first item, and completes the purchase, are deemed successful by oracle scripts. Influenced by agreement bias, MLLMs tend to validate such trajectories, but when SVG is applied, behaviors like these are rejected for lacking steps that confirm the item is the cheapest. On the other hand, as shown in Figure 11, an SVG-enhanced verifier initially rejects an otherwise correct execution for lacking actions to double-check item attributes. Nevertheless, as shown in this example and discussed next, such interventions are typically non-disruptive and often promote more generalizable behavior, ultimately yielding a positive effect on task completion.

5 Downstream Applications

In this section, we evaluate MLLM verifiers on their ability to guide agents toward task completion both in Reflexion and online supervision. These are applications of direct interest that so far, have shown limited benefits and occasional performance degradation [30, 39, 13, 15, 42, 46]. Additionally, this setting allows us to assess the net impact of SVG, given the tradeoffs discussed in Section 4.

Results - Reflexion. In this setting, after each episode: (i) the verifier evaluates the trajectory; (ii) the agent reflects on its past execution conditioned on the verifier’s evaluation; and (iii) these reflections are incorporated into subsequent attempts to enable self-correction over time. As shown in Figure 2, this approach yields an improvement of up to 21pps in completion rates after three iterations under the supervision of an oracle verifier, which sets an upper bound for performance. However, the base agent’s performance quickly plateaus when supervised by an MLLM verifier, achieving only 2.6pps. The reason is straightforward: influenced by agreement bias, the MLLM verifier fails to provide corrective feedback precisely when it is most needed—when the agent’s behavior is flawed and requires improvement. Quantitatively, this is reflected in the verifier’s TNR of 52% (Table 2): on average, corrective feedback is offered for only half of the flawed executions, substantially limiting the agent’s ability to revise its strategy and improve. In contrast, by enhancing failure identification, SVG enables consistent progress across runs, boosting task completion rates by up to 10.4pps.

Results - VisualWebArena and OSWorld Table 3 presents the performance of digital agents with and without feedback from MLLM verifiers. Compared to a no-verifier baseline, SVG results in a 5 percentage point (pp) increase in success rate on VisualWebArena ($\approx 13\%$ relative improvement) and a 5 pp gain in OSWorld ($\approx 22\%$ relative). Notably, our ReAct agent + SVG Verifier achieves a new **state of the art on VisualWebArena**, surpassing the previous best by 16 pp ($\approx 48\%$ relative). Unlike search methods, these gains come with a modest increase in token usage, and backtracking occurs through native web actions, making them more applicable to real-world scenarios. In contrast, and in line with previous work, we observe no improvements for a CoT-based MLLM verifier.

Table 3: Task success rate (SR) and token usage across VisualWebArena domains and OSWorld. Token usage (for VisualWebArena) is relative to the ReAct agent with no verifier.

Method	All VWA		Classifieds		Reddit		Shopping		OSWorld
	SR	Tokens	SR	Tokens	SR	Tokens	SR	Tokens	SR
Search Agent [17]*	29	4.9x	34	4.2x	22	5.4x	30	5.1x	–
R-MCTS [59]*	34	9.3x	41	7.4x	29	9.7x	32	10.1x	–
Base Agent	46	1x	49	1x	29	1x	52	1x	21.6
+ Verifier, no SGV	46	1.5x	48	1.5x	29	1x	53	1x	–
+ Verifier, SGV	52	1.9x	54	1.9x	38	1.7x	57	1.9x	26.3

*Results from prior work [17, 59].

A few factors explain these results. First, as shown in Figure 9, the SGV-enhanced verifier identifies suboptimal behavior and provides feedback that enables agents to backtrack and complete the task. When SGV is omitted, agreement bias prevents MLLM verifiers from providing feedback precisely when agent behavior is flawed, limiting their effectiveness (e.g., Section H.2). Second, while SGV leads to stricter verifiers (Section 4), their interventions are often non-disruptive. For example, in Figure 11 the verifier rejects a greedy strategy that technically satisfies the benchmark, prompting the agent to search for and confirm user-requested attributes, ultimately leading to task completion via a more generalizable strategy. Therefore, although performance declines in cases where agents fail to backtrack (e.g., Figure 12), the net effect of SGV remains positive. Table 11 quantitatively illustrates this pattern: of the 11% false negatives, only 0.9% transition to failures, while 14% of tasks improve through successful failure identification—yielding a net gain of 13%. In stark contrast, when SGV is not applied, 35% of tasks remain failures due to false positives, leading to no improvements. Further discussion and a categorization of factors influencing success rates are provided in Section C.2.

Results - robomimic Table 4 demonstrates that the inclusion of an SGV-enhanced verifier improves SR on the tool-hang task, surpassing the oracle-only counterpart by about 8 pp. In contrast, when SGV is not applied, MLLM-based verification leads to a reduction in SR. These findings not only align with, but also complement results observed in digital domains: this environment features closer-to-perfect oracles, and is therefore less prone to the evaluation challenges discussed in ?? . To provide intuition for these results, we report the partial success rates (PSR) and number of replans.

In the no-SGV case, the MLLM verifier is successful in guiding the policy up until states of partial completion. However, likely influenced by agreement bias, it struggles to further guide the policy to states of full task completion, as reflected in the low number of replans. In contrast, the SGV-enhanced verifier triggers replans more often, and while some of these interventions lead to failures, their overall impact is largely beneficial.

Table 4: Results of a diffusion policy on robomimic’s tool-hang task under different supervision methods.

Method	FR (%)	PSR (%)	SR (%)	# Replans
Oracle	32	68	24	88
Verifier, no SGV	22	78	16	89
Verifier, SGV	28	72	32	96

*FR, PSR and SR are the proportions of roll-outs with 0, 1 and 2 subtasks (out of 2) completed.

6 Conclusion

We identify agreement bias, a key limitation that hinders MLLMs from reliably serving as verifiers of agent behavior, adverse effects for applications that rely on them in this role. We introduce Self-Grounded Verification (SGV) — a simple yet effective method that significantly improves verification across a range of domains, in both online and offline settings. SGV has low token usage overhead, and can be easily integrated into pipelines leveraging MLLM verifiers.

Limitations and Future Work. While SGV mitigates agreement bias, it does not eliminate it entirely. Our qualitative analysis (Section C.1) shows that remaining failure cases often stem from limitations in the base models’ vision-language reasoning capabilities. Future work may refine SGV through other strategies to enhance multimodal reasoning and perception, and extend applications that could further expose its strengths and limitations. In parallel, compelling directions for open-ended verification include modular approaches combining MLLMs with symbolic methods [14], and the adaptation of recent reinforcement learning pipelines used to train generators for the verification problem.

References

- [1] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction, 2024. URL <https://arxiv.org/abs/2309.14316>.
- [2] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation, 2024. URL <https://arxiv.org/abs/2309.14402>.
- [3] ARC Prize Foundation. OpenAI o1 Results on ARC-AGI-Pub. <https://arcprize.org/blog/openai-o1-results-arc-prize>, September 2024. Accessed: 2025-07-09.
- [4] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL <https://arxiv.org/abs/2407.21787>.
- [5] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [6] Meilin Chen, Jian Tian, Liang Ma, Di Xie, Weijie Chen, and Jiang Zhu. Unbiased evaluation of large language models from a causal perspective, 2025. URL <https://arxiv.org/abs/2502.06655>.
- [7] Wei-Lin Chen, Zhepei Wei, Xinyu Zhu, Shi Feng, and Yu Meng. Do llm evaluators prefer themselves for a reason?, 2025. URL <https://arxiv.org/abs/2504.03846>.
- [8] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training, 2025. URL <https://arxiv.org/abs/2501.17161>.
- [9] DeepSeek-AI et al. Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL <https://arxiv.org/abs/2501.12948>.
- [10] Kimi Team et al. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.
- [11] Gemini Team, Google DeepMind. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. Technical Report —, Google DeepMind, Mountain View, CA, June 2025. Accessed 9 July 2025.
- [12] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL <https://arxiv.org/abs/2305.14992>.
- [13] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.
- [14] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Llms can’t plan, but can help planning in llm-modulo frameworks, 2024. URL <https://arxiv.org/abs/2402.01817>.
- [15] Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1234–1256, 2024.
- [16] 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. *arXiv preprint arXiv:2401.13649*, 2024.
- [17] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

- [18] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents, 2024. URL <https://arxiv.org/abs/2407.01476>.
- [19] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.
- [20] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023. URL <https://arxiv.org/abs/2301.12597>.
- [21] Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*, 2024.
- [22] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step, 2023. URL <https://arxiv.org/abs/2305.20050>.
- [23] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.
- [24] Yecheng Jason Ma, Joey Hejna, Ayzaan Wahid, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Jonathan Tompson, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision language models are in-context value learners, 2024. URL <https://arxiv.org/abs/2411.04549>.
- [25] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.
- [26] Tianyi Men, Zhuoran Jin, Pengfei Cao, Yubo Chen, Kang Liu, and Jun Zhao. Agent-rewardbench: Towards a unified benchmark for reward modeling across perception, planning, and safety in real-world multimodal agents, 2025. URL <https://arxiv.org/abs/2506.21252>.
- [27] OpenAI. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>, September 2024. Accessed: 2025-07-09.
- [28] OpenAI. Reasoning best practices. <https://platform.openai.com/docs/guides/reasoning-best-practices>, 2025. Last accessed July 9, 2025.
- [29] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024.
- [30] Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.
- [31] Arjun Panickssery, Samuel Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations. *Advances in Neural Information Processing Systems*, 37:68772–68802, 2024.
- [32] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.
- [33] Pooyan Rahmazadehgervi, Logan Bolton, Mohammad Reza Taesiri, and Anh Totti Nguyen. Vision language models are blind: Failing to translate detailed visual features into words, 2025. URL <https://arxiv.org/abs/2407.06581>.
- [34] Gabriel Sarch, Lawrence Jang, Michael J. Tarr, William W. Cohen, Kenneth Marino, and Katerina Fragkiadaki. Vlm agents generate their own memories: Distilling experience into embodied programs of thought, 2025. URL <https://arxiv.org/abs/2406.14596>.

- [35] Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. Scaling test-time compute without verification or rl is suboptimal. *arXiv preprint arXiv:2502.12118*, 2025.
- [36] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- [37] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- [38] Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models, 2025. URL <https://arxiv.org/abs/2310.13548>.
- [39] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.
- [40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- [41] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.
- [42] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024.
- [43] Linzhuang Sun, Hao Liang, Jingxuan Wei, Bihui Yu, Tianpeng Li, Fan Yang, Zenan Zhou, and Wentao Zhang. Mm-verify: Enhancing multimodal reasoning with chain-of-thought verification, 2025. URL <https://arxiv.org/abs/2502.13383>.
- [44] Gokul Swamy, Sanjiban Choudhury, Wen Sun, Zhiwei Steven Wu, and J Andrew Bagnell. All roads lead to likelihood: The value of reinforcement learning in fine-tuning. *arXiv preprint arXiv:2503.01067*, 2025.
- [45] Brandon Trabucco, Gunnar Sigurdsson, Robinson Piramuthu, and Ruslan Salakhutdinov. Towards internet-scale training for agents. *arXiv preprint arXiv:2502.06776*, 2025.
- [46] Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can large language models really improve by self-critiquing their own plans?, 2023. URL <https://arxiv.org/abs/2310.08118>.
- [47] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [48] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL <https://arxiv.org/abs/2409.07429>.
- [49] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [50] Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilya Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models, 2024. URL <https://arxiv.org/abs/2406.16838>.

- [51] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.
- [52] Wenda Xu, Guanglei Zhu, Xuandong Zhao, Liangming Pan, Lei Li, and William Wang. Pride and prejudice: Llm amplifies self-bias in self-refinement. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15474–15492, 2024.
- [53] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v, 2023. URL <https://arxiv.org/abs/2310.11441>.
- [54] Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents, 2025. URL <https://arxiv.org/abs/2410.13825>.
- [55] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.
- [56] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [57] Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. Justice or prejudice? quantifying biases in llm-as-a-judge, 2024. URL <https://arxiv.org/abs/2410.02736>.
- [58] Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- [59] Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. *arXiv preprint arXiv:2410.02052*, 2024.
- [60] Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. EXACT: Teaching AI agents to explore with reflective-MCTS and exploratory learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=GBIUbwW9D8>.
- [61] Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024.
- [62] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.
- [63] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [Yes]

Justification: The main claims in the abstract and introduction are supported by the results in sections 4 and 5.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Limitations are discussed in Sections C.2 and 6 and throughout the main text.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not contain theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.

- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: The experiments can be reproduced using the settings described in sections 3, and appendix sections A and B, and the code included in the supplementary material.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [\[Yes\]](#)

Justification: We release the code as part of the supplementary material and will public release it after the review process.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We describe experiment settings in sections 3, A and B.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Due to resource and time constraints, we are unable to perform multiple runs of each experiment.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The resources used are described in section A.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: The research fully conforms with the Code of Ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss potential impacts in the supplementary material.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The methods described in the paper do not pose a high risk for misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the papers of code implementations that we used.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset’s creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: The code and documentation are released as supplementary material.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [\[NA\]](#)

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [\[NA\]](#)

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.

A Experimental Details

A.1 Offline Verifier Experiments

For experiments in Table 1, we use 0 temperature for all models, except in the majority voting majority voting baselines, where we select the most frequent response among 8 completions generated with a temperature of 1.5, top-p of 0.95, and top-k set to 64.

For experiments in table Table 2, we employ `gemini-2.5-flash` with a temperature of 0. When thinking is enabled, we set `thinking` budget to 8192 tokens, the model’s default value, and to 24576 in the “high” budget case.

We omit CoT instructions for LRMs, as these models inherently generate reasoning traces, and adding them can hurt performance [28].

The model is given interleaved images and text, in the form of screenshots produced by the environment and actions generated by the agent. Figure 5 displays an example of a trajectory in VisualWebArena, and Section B contains details on prompt templates.

A.2 Online Setting Experiments

Verifier Agent Implementation Details. For VisualWebArena, the MLLM verifies task completion when the agent issues a stop action, and if the task is deemed incomplete, it generates natural language feedback that is appended to the agent’s context window. For OSWorld, the verifier is additionally called every 5 steps, regardless of task completion. In robomimic, the MLLM verifies task progression at predetermined steps and triggers the diffusion policy to regenerate an action sequence (or “replan”) if the trajectory is deemed off-track; in the baseline scenario, the policy receives oracle-only feedback when subtasks are complete. The numbers reported are averages across three runs in the digital benchmarks, and 50 rollouts with a horizon of 700 time steps in robomimic.

Web Navigation and Computer Use. For VisualWebArena experiments in Section 5, we use `gemini-2.5-flash-preview-04-17` for both the ReAct agent and the verifier. For the no-SGV case, the model is given a CoT instruction as in Section B. In the SGV case, in the first step we adopt the model’s default parameters: `temperature=1`, `top_p=0.95`, and `top_k=64`. In the verification step, we use 0 temperature. The agent and environment settings follow those in Koh et al. [16], using a temperature of 1.0, top-p of 0.95, top-k of 40, and the same 768 token limit. We adopt Set-of-Mark prompting [53] for the screenshots, with a viewport size of 1280×2048 pixels. We use BLIP-2 [20] to provide text captions for images included with the task objectives. We set the maximum number of steps to 30, allowing the agent some room to backtrack while imposing a search budget comparable to — though still more constrained — than that of prior work employing tree search methods [17, 59].

For the OSWorld experiments, we follow the same setup and implementation as in Qin et al. [32]. The verifier configuration remains the same, using `gemini-2.5-flash-preview-04-17` with a temperature of 0 and a token limit of 768. We allow up to 50 steps for the agent. We run the UI-TARS-1.5-7B model on a single RTX 3090 GPU. We find that 28 out of the 369 tasks in OSWorld consistently produce unresolvable errors that prevent evaluation on those tasks; we report our results on the remaining 341 tasks.

In both environments, the verifier functions as an outcome-based reward, evaluating the trajectory when the agent signals task completion, and every 5 steps in the case of OSWorld. The prompt templates and inputs given to the verifier are the same as the ones in the offline scenario.

If the verifier deems the trajectory unsuccessful, it provides natural language feedback, which is then incorporated into the agent’s context window to inform subsequent generations. Section B.3 shows how the feedback is incorporated, and figures in Section H provide examples of evaluations and generated feedback.

Robotic Manipulation For robomimic, the verifier receives a natural language description of the task, along with the last screenshot, and is prompted to assess task completion every 20 time steps. Additional details about the prompts are provided in Section B.5.

B Prompts

For all prompts, content enclosed in curly braces (`{...}`) is dynamically replaced at runtime with the corresponding information. Text enclosed in double angle brackets (`«...»`) provides explanatory notes for the reader and is not included in the prompts at runtime. The labels `USER:` and `ASSISTANT:` indicate role fields passed via the OpenAI API format and are not literal strings in the prompt. Text separated by a pipe (`|`) inside curly braces (`{...}`) indicates phrases that change depending on the environment.

B.1 Verifier Prompts

To maximize consistency and minimize sensitivity due to prompt variations, we construct all verifier prompts from the following parts:

- The system prompt, which describes the role of the agent and environment details.
- The trajectory prompt, which is created in an interleaved user-assistant conversation format, and contains information on the task to be accomplished, screenshots of the environment states, and the actions taken by the agent being evaluated.
- The evaluation prompt, which includes the criteria for the evaluation, as well as instructions for step-by-step reasoning.

We aim to keep the prompts as consistent as possible across experiments, applying slight adjustments to accommodate environment-specific details and prompting variations. For instance, the NoCoT variant in section 4 is nearly identical to the CoT variant, differing only in the removal of the step-by-step reasoning instructions.

Verifier System Prompt - VisualWebArena

You are an intelligent agent tasked with supervising an assistant navigating a web browser to accomplish a web-based task. Your job is to evaluate the assistant's work, and provide feedback so it can progress towards the objective.

Here's the information you'll have:

The objective: This is the task the assistant is trying to complete.

Webpage screenshots: These are screenshots of the webpage, with each interactable element assigned a unique numerical id. Each bounding box and its respective id shares the same color.

The execution trace: This is a sequence of webpage screenshots paired with the assistant's actions, detailing the web navigation so far.

General web knowledge: This is a general description of how tasks like this are typically accomplished on the web. <<Only included for SGV variants>>

Assistant's capabilities: To effectively analyze the assistant's work, consider the actions it can perform. These actions fall into the following categories:

Page Operation Actions:

``click [id]``: Click on an element with a specific id on the webpage.

``type [id] [content] [enter\_after]``: Type the content into the field with id. If `enter\_after` is 0, the "Enter" key is not pressed after typing; otherwise, the "Enter" key is automatically pressed.

``hover [id]``: Hover over an element with id.

``press [key\_comb]``: Press a keyboard key or key combination (e.g., delete, ctrl+a).

``scroll [down]`` or ``scroll [up]``: Scroll the webpage up or down.

```

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

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

### Completion Action:
stop [answer]: This action is issued when the task is believed to be complete. If the task requires a text-based answer, the answer is provided within the brackets. If the task is deemed impossible, this action is issued optionally with a reason why.

## To be successful, it is very important to follow the following rules:
1. You must not assume the assistant's work is correct or incorrect beforehand. You should come up with your own opinion based on the information provided.
2. You must connect the dots between the objective and the information provided.
3. Your evaluation should be based on task completion; don't worry about efficiency for now.
4. Give utmost importance to the visual information provided, especially the screenshots in the execution trace.
5. Use the web knowledge as a rule, but also consider the context of the task given to you. <<Only included for SGV variants>>

```

Verifier System Prompt - OSWorld

You are an intelligent agent tasked with supervising an assistant utilizing a computer to accomplish computer-based tasks. Your job is to evaluate the assistant's work, and provide feedback so it can progress towards the objective.

```

## Here's the information you'll have:
### The objective: This is the task the assistant is trying to complete.
### Screenshots: These are visual captures of the computer screen taken at specific states of the navigation process. The red markers, if present, indicates the destination of mouse actions taken at the corresponding state.
### The execution trace: This is a sequence of screenshots of the computer screen paired with the assistant's responses, detailing the computer navigation so far.
### General Computer Knowledge: This is a general description of how tasks like this are typically accomplished on a computer. <<Only included for SGV variants>>

## Assistant's capabilities: To effectively analyze the assistant's work, consider the actions it can perform. These actions fall into the following categories:
### Mouse Actions:
- click(start_box='<|box_start|>(x1,y1)<|box_end|>'): Left clicks at the (x1,y1) coordinates.
- left_double(start_box='<|box_start|>(x1,y1)<|box_end|>'): Left double click at the (x1,y1) coordinates.
- right_single(start_box='<|box_start|>(x1,y1)<|box_end|>'): Right click at the (x1,y1) coordinates.
- drag(start_box='<|box_start|>(x1,y1)<|box_end|>', end_box='<|box_start|>(x3,y3)<|box_end|>'): Drags the element at the (x1,y1) coordinates to the (x3,y3) coordinates.
- scroll(start_box='<|box_start|>(x1,y1)<|box_end|>', direction='down or up or right or left'): Scrolls the mouse wheel in the specified direction.

### Keyboard Actions:
- hotkey(key=''): Press a specific key.
- type(content=''): Types the string following `content`. If followed by "\n", it means the input is submitted.
- wait(): Pauses for 5s and takes a new screenshot to check for any changes.

### Completion Action:
- finished(content='<|content|>'): This action is issued when the task is believed to be complete. `content` is optional, and is used to provide a reason why the task is complete.

## To be successful, it is very important to follow the following rules:
1. You must not assume the assistant's work is correct or incorrect beforehand. You should come up with your own opinion based on the information provided.
2. You must connect the dots between the objective and the information provided.
3. Give utmost importance to the visual information provided, especially the screenshots in the execution trace.
4. Your evaluation should be based on task completion; don't worry about efficiency for now.
5. Use the General Computer Knowledge as a guide, but also consider the context of the specific task given to you. <<Only included for SGV variants>>

```

Verifier Trajectory Prompt - VisualWebArena and OSWorld

```
USER: ## OBJECTIVE: {task objective text} {task objective images}

USER: ### STATE `t-j` SCREENSHOT: {screenshot}
ASSISTANT: {generated action}
...
USER: ### STATE `t-1` SCREENSHOT: {screenshot}
ASSISTANT: {generated action}
```

Full Verifier Prompt

```
{system prompt}
{trajectory prompt}

USER:
## General web knowledge: {output from first step} <<Only included for SGV variants>>

Now please provide your response.

## Here is the evaluation criteria:
SUCCESS: The assistant executed all of what's necessary to complete the objective. The task is
fully accomplished.
PARTIAL SUCCESS: The assistant executed most of what's necessary to complete the objective.
The task is partially accomplished.
FAILURE: The assistant executed mostly incorrect steps. The task is not accomplished, and
major revisions are needed.

## Provide your response as follows:
REASONING: <Comprehensive step-by-step reasoning to come up with your evaluation and feedback> <<
Only included for CoT variants>>
EVALUATION: <Your evaluation following the evaluation criteria>
FEEDBACK: <Feedback so the assistant can progress towards the objective>
```

Pan et al. [29] Verifier System Prompt

You are an expert in evaluating the performance of a web navigation agent. The agent is designed to help a human user navigate a website to complete a task. Given the user's intent, the agent's action history, the final state of the webpage, and the agent's response to the user, your goal is to decide whether the agent's execution is successful or not.

There are three types of tasks:

1. Information seeking: The user wants to obtain certain information from the webpage, such as the information of a product, reviews, map info, comparison of map routes, etc. The bot's response must contain the information the user wants, or explicitly state that the information is not available. Otherwise, e.g. the bot encounters an exception and respond with the error content, the task is considered a failure. Besides, be careful about the sufficiency of the agent's actions. For example, when asked to list the top-searched items in a shop, the agent should order the items by the number of searches, and then return the top items. If the ordering action is missing, the task is likely to fail.
2. Site navigation: The user wants to navigate to a specific page. Carefully examine the bot's action history and the final state of the webpage to determine whether the bot successfully completes the task. No need to consider the bot's response.
3. Content modification: The user wants to modify the content of a webpage or configuration. Carefully examine the bot's action history and the final state of the webpage to determine whether the bot successfully completes the task. No need to consider the bot's response.

IMPORTANT

Format your response into two lines as shown below:

Thoughts: <your thoughts and reasoning process>
Status: "success", "partial success", or "failure"

Full Pan et al. [29] Verifier Prompt

```
{system prompt}

USER:
## User Intent: {task objective text}

## Action History:
1: {generated action}
...
n: {generated action}

## Last snapshot of the webpage: {screenshot}
```

B.2 SGV Verifier - First Step Prompts

For the main experiments, we use separate prompts for the retrieval step and the evaluation step. Below, we present the prompts for the first step. As shown in Section B.6, similar performance can be achieved by reusing the prompts from Section B.1, as long as the verification process is explicitly divided into retrieval and evaluation phases.

SGV - First Step System Prompt

```
You are a helpful assistant with deep knowledge in {web navigation | computer-based workflows}.
Your job is to provide a description of how tasks like the ones provided to you are typically
accomplished on {the web | computers}.

## Here's the information you'll have:
### Objective: This is an english description of the task, possibly accompanied by images that
provide more context on what must be accomplished.
### Screenshots: These are screenshots of the {webpage | computer screen}, giving you context to
further understand what must be accomplished.
```

SGV - Full First Step Prompt

```
{system prompt first-step}
## OBJECTIVE: {task objective text}

## IMAGES:
Image 0: {task objective image 1}
...
Image n: {task objective image n}
Image n+1: initial {webpage | computer} screenshot: {screenshot}

Now please provide your response:
<Description of how tasks such as this are typically accomplished on {the web | computers}>.
```

B.3 VisualWebArena Agent Prompt

The prompt for VisualWebArena is built upon the authors' original implementation [16], with the following changes:

- We provide the history of thoughts and actions generated by the agent in previous interactions, which serve as a text-based memory to prevent loops and allow backtracking.
- We refine parts of the prompt to use Markdown formatting and to make the instructions more precise.
- For one of the examples, we include previous thoughts and actions to match the new inputs.

VisualWebArena Agent Prompt

```
{system prompt}
{examples}

## RATIONALE AND ACTION HISTORY:
### STATE `t-j`:
- **RATIONALE**: {agent generations}
- **ACTION**: {action parsed by the environment}
...
### STATE `t-1`:
- **RATIONALE**: {agent generations}
- **ACTION**: {action parsed by the environment}

Here is the current state `t` for you to determine what to do next:
## TEXT OBSERVATION `t`: {text observation}

## URL: {webpage URL}

## OBJECTIVE: {task objective text}

## FEEDBACK: Here is your previous response at the current state `t` and **feedback** about it.
Use this to revise your response if you deem appropriate.
### Previous generated response: {last generated action}
### Feedback: {verifier feedback} <<Only included in rounds where feedback is provided by the
verifier>>

IMAGES:
Image 0: {task objective image 0}
...
Image n: {task objective image n}
Image n+1: current webpage screenshot at state `t`: {screenshot}
```

B.4 OSWorld Agent Prompt

OSWorld Agent Prompt

```
You are a GUI agent. You are given a task and your action history, with screenshots. You need to
perform the next action to complete the task.

## Output Format
...
Thought: ...
Action: ...
...

## Action Space

click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>', end_box='<|box_start|>(x3,y3)<|box_end|>')
hotkey(key='')
type(content='') #If you want to submit your input, use "\n" at the end of `content`.
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>', direction='down or up or right or left')
wait() #Sleep for 5s and take a screenshot to check for any changes.
finished(content='xxx') # Use escape characters '\', '\"', and '\n' in content part to ensure we can
parse the content in normal python string format.

## Note
- Use English in `Thought` part.
- Write a small plan and finally summarize your next action (with its target element) in one
sentence in `Thought` part.

## User Instruction
{task objective text}
```


B.5 robomimic Prompts

robomimic SGV Prompt - First Step

You are highly skilled in robotic tasks.
Based on the task description and the image of the initial state, predict how the task must look like at the current time stamp.

Task Description:
Pick up L-shaped pencil and insert L-shaped pencil into the hole
Pick up tool and hang tool on L-shaped pencil

Current time stamp: {n/N}
{image}

robomimic SGV Prompt - Second Step

Compare the prediction with the actual state of the system in the current time step depicted in the image.

If the current state shows a situation that depicts failure, label it as failure.

If both tasks being completed or nearing completion was predicted and both tasks have been completed, label it as success.

If the prediction is that the task will be in progress and the task is in progress, label it as in progress.

{first step generation}
Current time stamp: {n/N}
{image}

robomimic - Verifier Prompt, no SGV

You are highly skilled in robotic task verification.

Based on the task description, the current time step and the the actual state of the system in the current time step (depicted in the image), label it as 'success', 'failure' or 'in progress'.

Given the time step, if the current state is not as per expectation (with dropped tools), label it as failure.

If the time step is towards completion, and the task has been completed, label it as success.

Given the time step, if the task is expected to be in progress and the task is in progress, label it as in progress.

Current time stamp: {n/N}
{image}

B.6 Ablation Prompts

As we show in Section B.6, similar performance for SGV can be achieved by re-utilizing the verifier prompts from Section B.1, as long as retrieval and verification are divided into two phases. Below is the prompt used in the first step. The second step prompts are identical to those in Section B.1.

SGV Unified - Full First Step Prompt

{verifier system prompt} <<same as in Appendix B.6>>

OBJECTIVE: {task objective text}

IMAGES:
Image 0: {task objective image 0}

```

...
Image n: {task objective image n}
Image n+1: initial {webpage | computer} screenshot: {screenshot}

Please first provide the following
<Description of how tasks such as this are typically accomplished on {the web | computers}>.

```

The following prompts are used when retrieval and verification are performed together in a single, monolithic step. The system prompts are the same as in Section B.1, except for the rules, which are replaced by the following.

Monolithic Retrieve and Verify - System Prompt Rules

```

## To be successful, it is very important to follow the following rules:
1. You must not not assume the assistant's work is correct or incorrect beforehand. You should come up with your own opinion based on the information provided.
2. You must connect the dots between the objective and the information provided.
3. Your evaluation should be based on task completion; don't worry about efficiency for now.
4. Give utmost importance to the visual information provided, especially the screenshots in the execution trace.
5. You should come up with a detailed description of how tasks like this are typically accomplished on {the web | computers}.
6. Use this General {Web Knowledge | Computer Knowledge} as a guide, but also consider the context of the specific task given to you.

```

Monolithic Retrieve and Verify - Full Prompt

```

{system prompt}
{trajectory prompt}

Now please provide your response.

## Here is the evaluation criteria:
SUCCESS: The assistant executed all of what's necessary to complete the objective. The task is fully accomplished.
PARTIAL SUCCESS: The assistant executed most of what's necessary to complete the objective. The task is partially accomplished.
FAILURE: The assistant executed mostly incorrect steps. The task is not accomplished, and major revisions are needed.

## Provide your response as follows:
GENERAL {WEB | COMPUTER} KNOWLEDGE: <Description of how tasks such as this are typically accomplished on {the web | computer}>
REASONING: <Comprehensive step-by-step reasoning to come up with your evaluation and feedback> << Only included for CoT variants>>
EVALUATION: <Your evaluation following the evaluation criteria>
FEEDBACK: <Feedback so the assistant can progress towards the objective>

```

C Qualitative Analysis

C.1 Agreement Bias

Figure 5 illustrates the trajectory for a failed task in VisualWebArena, along with evaluations produced by three verifier variants. Two key flaws are evident in the trajectory: (i) the agent does not perform any price comparisons, and (ii) it stops after adding an item to the cart, without proceeding to checkout. The CoT verifiers validate the execution, providing reasoning traces that justify it as correct despite the evident shortcomings. In contrast, the SGV verifier flags the absence of price comparison, and accurately declares the execution as unsuccessful.

C.2 Online Verification and Feedback

As discussed in section 5, our method enables an MLLM verifier to monitor agent behavior and provide natural language feedback in real time, guiding the agent toward successful task completion. We show there that this mechanism has a positive overall impact on performance, resulting in

a 5 percentage point increase in success rate on VisualWebArena (approximately 10% relative improvement) and a 2 percentage point gain in OSWorld (approximately 9% relative), compared to a no-verifier baseline.

The success of such a mechanism relies on three factors: (i) the accuracy of the verifier’s judgment, (ii) the quality of its feedback, and (iii) the ability of the agent to interpret and act upon that feedback, including the capacity to revise or backtrack when necessary.

In this section, we explore these factors through representative cases that offer insight into both the strengths and failure modes of our method — and of MLLM verifiers more broadly. The cases are as follows:

- **Accurate verification with helpful feedback:** The verifier correctly identifies flaws and provides actionable feedback that improves task execution.
- **Accurate verification with helpful feedback, limited by agent ability:** The verifier offers correct judgments and useful feedback, but the agent fails to act on it effectively.
- **Overly strict verification, harmless:** False negatives from the verifier that do not affect the agent’s final outcome and often elicit more robust behavior.
- **Overly strict verification, harmful:** False negatives that lead to degraded task performance or failed executions.
- **Overly lenient verification:** False positives where the verifier fails to detect clear mistakes, and contributes no useful signal. This stems from agreement bias and is the most prominent failure case for conventional MLLM verifiers.

Accurate Verification with Helpful Feedback Figure 9 shows an example where the verifier successfully steers the ReAct agent to a correct execution on a challenging VisualWebArena task. The agent performs a sequence of reasonable steps and finds a product closely aligned with the user’s request, but not the cheapest one. The verifier detects this oversight and provides feedback that prompts the agent to backtrack, sort the results correctly, and select the appropriate item.

This is a particularly challenging case that is highly susceptible to agreement bias, as the initial trajectory comprises reasonable steps and ends with a product that closely aligns with the user’s query.

Accurate Verification with Helpful Feedback, Limited by Agent Ability Figure 10 presents a failure case where, despite the verifier correctly identifying the issue and providing appropriate feedback, the agent is unable to recover. The verifier advises the agent to calculate shipping costs before leaving a review. However, the agent instead performs a sequence of poor actions, from which it never recovers. This example demonstrates a central limitation of our agent: the performance is ultimately constrained by the capabilities of the generator, regardless of the quality of the verifier.

Overly Strict Verification, Harmless Figure 11 illustrates this dynamic in a borderline case. The agent clicks on the first visible option without searching. Although this behavior is marked as correct by automated scripts, the verifier rejects it, requiring the agent to explicitly search for the product. After performing the search, the agent provides the same (correct) response, but is again rejected for not clicking on the item to inspect its details. In this task, that step is arguably unnecessary, as the required information is already visible. Still, the agent proceeds to click and confirm the details, ultimately converging on the same answer a third time.

This example reveals two insights: (i) Even though the verifier is demanding, its feedback aligns well with the user’s original intent, and does not prevent task success. (ii) In fact, the strict verifier elicits robust behavior from the agent. The benchmark task is relatively simple and susceptible to shortcuts, but the verifier enforces a robust and generalizable strategy.

Overly Strict Verification, Harmful Figure 12 presents an example where a correct initial execution is degraded into a failure. The user’s query accepts either “Ohio” or “Pennsylvania” as valid answers, but the verifier demands the cheapest option across both. The agent, however, is unable to navigate back to Ohio to perform a valid comparison and runs out of steps. This example also illustrates the difficulty of interpreting real-world user intents, which are often vague and under-specified.

Overly Lenient Verification Our verifier remains susceptible to false positives, primarily due to limitations in the underlying MLLM’s vision-language perception and reasoning capabilities.

Figure 7 shows a failure on a relatively simple task in VisualWebArena that exposes this issue: both the agent and the verifier fail to count the number of elements in an image correctly, resulting in a false positive. This is a known weakness of current MLLMs [33] that becomes even more problematic in our setting, where perception errors can compound across multi-step trajectories.

A more complex failure is illustrated in Figure 8, which juxtaposes the verifier’s strengths and weaknesses. The verifier successfully detects that the agent has terminated the task prematurely, failing to submit a required comment. However, it misses two critical issues: (i) that the agent has incorrectly counted the number of red keys in the image, and (ii) that it has posted the comment in the wrong location. Accurately verifying these mistakes would require the detection of fine-grained visual cues, knowing which small button the agent clicked on, for example.

We expect perceptual and reasoning abilities to improve in future MLLMs, and consequently, the effectiveness of our SGV approach to increase as well, as the improvements it provides are largely orthogonal to these capabilities. Alternatively, an interesting direction for future work would be to augment the MLLM with specialist models focused on fine-grained perception tasks, such as object counting or UI element recognition, thereby enhancing the verifier’s accuracy in visually demanding scenarios.

D Ablations

D.1 Agreement Bias in Retrieval and Ablations to SGV

Table 5 presents ablations for results in ???. Specifically, we (1) start with a conventional MLLM verifier, (2) add a retrieval step to it, (3) decouple the retrieval and verification steps without modifying any of the prompts, and (4) add a specific prompt for the retrieval step. In all cases, we include a CoT instruction. Prompt templates are provided in Section B.6.

The conventional MLLM-based verification yields suboptimal performance, with slight improvement when a retrieval step is added (rows 1–2). One reason for this behavior is that when steps are combined, models tend to retrieve priors that align to information in their context window — a manifestation of agreement bias — reducing their effectiveness in grounding the verification. Decoupling the steps improves performance (rows 3–4), with SGV achieving similar results with or without retrieval-specific prompts. This suggests that SGV’s effectiveness does not stem from prompt specialization and that it can be integrated easily with minimal to no modifications to existing codebases.

Table 5: Evaluation of trajectories in VisualWebArena - ablations. TPR = true positive rate, TNR = true negative rate.

#	Method	TPR (%)	TNR (%)	Accuracy (%)
1	Base MLLM Verifier	89	54	69
2	+ Retrieval Step	80	63	70
3	SGV, Unified	73	74	74
4	SGV	75	77	76

D.2 Online Supervision and SGV with Weaker Models

Table 6 shows the performance of online verification with and without SGV on the full VisualWebArena benchmark utilizing an earlier and stable version of Gemini, `gemini-2.0-flash-001`, compared to the one used in Section 5. Consistent with findings in Section 5, the no-SGV verifier fails to improve performance and even results in degradation in certain domains. In contrast, the SGV verifier consistently improves results across all domains, achieving a 3 percentage point gain overall (approximately 9% relative improvement). Moreover, and in line with Section 4, these results also reveal that SGV can benefit both weak and strong models.

E AgentRewardBench Results

Table 7 shows the performance of the updated oracle evaluators in AgentRewardBench. The original oracles in VisualWebArena exhibit relatively low alignment with human annotations, particularly

Table 6: Task success rates (%) on VisualWebArena with and without SGV.

Method	All VWA	Shopping	Reddit	Classifieds
ReAct	35	38	23	41
+ Verifier, no SGV	35	39	23	40
+ Verifier, SGV	38	41	25	46

in terms of TPR. After applying our fixes, both TPR and overall accuracy improve substantially, with the corrected oracles reaching near-perfect agreement. Furthermore, Table 8 shows that our SGV-Verifier achieves state-of-the-art performance on the benchmark for the VisualWebArena environments, despite relying on a weaker non-thinking model. Note that in AgentRewardBench the main performance metric for the leaderboard is precision.

Table 7: Performance of oracle evaluators in AgentRewardBench (Oracle = Rule-based).

	Original	Fixed Oracle
P	85.2	100
TPR	58.2	92
TNR	95.9	100
Acc	85.1	98

Table 8: Comparison of our verifier to best performing judges in AgentRewardBench for the VisualWebArena environment.

Method	Precision
Rule-based	85
WebJudge (o4) (SOTA)	75
SGV (Gemini 2.5, no Thinking)	80
No-SGV	68

F Agent Trajectory Evaluation: Additional Agreement Bias Metrics

In Figure 3, we show the distribution of MLLM Verifier responses across over 20 scoring templates covering both Likert (e.g., SUCCESS, PARTIAL SUCCESS, UNCERTAIN, FAILURE) and score-based templates (e.g., scalar scores reflecting the MLLM confidence on correctness of a given trajectory). We also include randomized option ordering, criteria shuffling and reversal following [6] and [57]. We highlight three observations. First, MLLMs tend to concentrate their evaluations at the high end of the scale, largely independent of how the evaluation is prompted. For example, some templates explicitly tie the highest scores to strong confidence that the assistant executed all required steps to complete the task. Second, when applying SGV, the distribution of scores becomes more balanced, avoiding the sharp skew toward overly positive judgments. Finally, models rarely indicate uncertainty in their responses, even for tasks that are inherently vague. Due to space limitations, please refer to our codebase for all templates.

G Additional Details and Results for VisualWebArena

G.1 Representative Subset

Running the full VisualWebArena benchmark is time-consuming, and this hinders rapid prototyping, experimentation, and ablation studies. To address this issue, we release a representative subset of tasks that preserves the performance trends observed on the full benchmark, along with the algorithm used to construct such subsets.

The full code is available in our codebase. At a high level, the subset is constructed based on the task-level performance of a base agent on the full benchmark. Tasks are iteratively added or removed

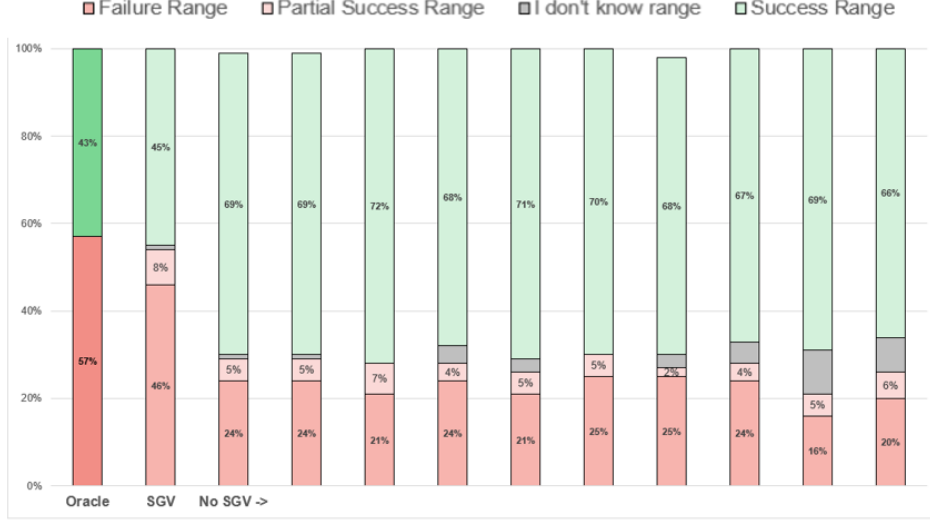


Figure 3: Average distribution of MLLM Verifier responses across scoring templates

to match the distribution of task templates and other characteristics, while ensuring that the aggregate success rate within each domain remains closely aligned with that of the full set.

Our VisualWebArena subset is one third the size of the full benchmark. To avoid introducing bias toward our implementations, we construct the subset based on the performance of Search Agent [17], for which task-level scores are publicly available. We compare task success rates across various agents and model families on both the full benchmark and the subset in Table 9.

The differences in success rate between the full benchmark and the subset range from 0 to 1 percentage point, while execution time and token usage are reduced by approximately 67%, substantially accelerating the development cycle and lowering inference costs. Note that performance gaps are small even when relative improvements between similar agent variants are considered. For instance, the performance gain of the ReAct + Verifier agent over the ReAct agent is approximately 4.5 percentage points on both the full benchmark and the representative subset, indicating that the subset preserves meaningful signals for measuring improvement.

Table 9: Task success rates (%) in VisualWebArena for the full benchmark and our representative subset.

Method	Model	Full Benchmark				Representative Subset				Δ
		All	Shop.	Reddit	Class.	All	Shop.	Reddit	Class.	
Search Agent [17]	GPT-4o	27	29	21	30	28	29	22	30	-0.3
ReAct	Gemini 2.0	35	38	23	41	35	35	26	43	0.4
ReAct	Gemini 2.5	46	52	28	49	45	51	29	46	0.9
ReAct + Verifier	Gemini 2.0	38	41	25	46	38	38	29	47	0.2
ReAct + Verifier	Gemini 2.5	50	57	33	52	49	55	33	52	0.8
Number of Tasks		910	466	210	234	305	156	70	79	-
Token Usage (millions)*		104	54	29	21	34	18	10	6	-

*Token usage measured for Gemini 2.0 (ReAct + Verifier), which at the time of writing is the latest non-experimental version of Gemini.

G.2 Results for GPT-4o

The VisualWebArena success rates reported in section 5 are based on `gemini-2.5-flash-preview-04-17`, which makes comparison to prior work difficult. To enable a direct comparison, Table 10 reports results for `gpt-4o-2024-05-13`, the same model used in previous studies, evaluated on the

representative task subset described in Section G.1 without the environment modifications explained in Section G.4 and Section G.5.

Table 10: Task success rates (SR, %) and relative token usage on the VisualWebArena representative subset (Section G.1) using gpt-4o-2024-05-13 as the base model.

Method	All VWA		Classifieds		Reddit		Shopping	
	SR	Tokens	SR	Tokens	SR	Tokens	SR	Tokens
Search Agent [17]*	29	4.9x	34	4.2x	22	5.4x	30	5.1x
R-MCTS [59]*	34	9.3x	41	7.4x	29	9.7x	32	10.1x
ReAct	31	1x	29	1x	24	1x	36	1x
ReAct + SGV Verifier	36	1.35x	33	1.6x	27	1.2x	42	1.3x

*Results from prior work.

G.3 Online Verification - Breakdown of Performance per confusion metrics

Table 11: VisualWebArena - % Tasks converted to failures, success and preserved, per confusion case.

	SGV				No-SGV			
	0→0	0→1	1→0	1→1	0→0	0→1	1→0	1→1
FN	-	-	0.9	11	-	-	1.1	4
TN	28	14	-	-	18	1	-	-
FP	17	-	-	-	35	-	-	-
TP	-	-	-	33	-	-	-	41

G.4 Task Objective Modifications

Some tasks in VisualWebArena require navigating to a specific web page for successful completion, but this is not stated in the original task objectives. Following [60], we append the following instruction to the objectives of such tasks: *“To finish the task, please make sure to navigate to the page of the corresponding item.”* Examples for each domain are shown below.

Task Objective Modification Examples

Classifieds

Original Objective: Find me the cheapest red Toyota. It should be between \$3000 to \$6000.

Modified Objective: Find me the cheapest red Toyota. It should be between \$3000 to \$6000. To finish the task, please make sure to navigate to the page of the corresponding item.

Shopping

Original Objective: Search for usb wifi and show me the first item in the second row.

Modified Objective: Search for usb wifi and show me the first item in the second row. To finish the task, please make sure to navigate to the page of the corresponding item.

Reddit

Original Objective: Find me a post in /f/dataisbeautiful that illustrates the Nvidia revenue in FY22, and tell me the revenue of the whole company in FY22 (in billions).

Modified Objective: Find me a post in /f/dataisbeautiful that illustrates the Nvidia revenue in FY22, and tell me the revenue of the whole company in FY22 (in billions). To finish the task, please make sure to navigate to the page of the corresponding item.

G.5 Environment Refinements

Bug Fix. Several tasks in VisualWebArena’s shopping domain require sorting products by certain attributes. However, a bug in the environment prevents performing such actions, so we modify the environments to allow them. We provide an ablation to this change in Table 12.

Table 12: Task success rates (%) on the VisualWebArena representative subset (Section G.1) using `gemini-2.5-flash-preview-04-17` as the base model, before and after our bug fix.

Method	All VWA	Classifieds	Reddit	Shopping
ReAct - Before Bug Fix	44	47	30	49
ReAct - After Bug Fix	45	46	29	51
ReAct + SGV Verifier - Before Bug Fix	47	51	33	51
ReAct + SGV Verifier - After Bug Fix	49	52	33	55

We also introduce several other technical improvements to facilitate prototyping in VisualWebArena. They include: a faster method for detecting page load completion, parallelization of docker containers, the incorporation of additional models for fuzzy match evaluation, model hosting for the captioner, and minor refinements to the inputs provided to the agent, such as as Set-of-Mark prompting, tab information, and scroll bars.

In Table Section G.5, we report the speedup gains obtained with the environments we release and our Gemini 2.5-based agents. When considering our representative subsets, these gains can reach up to $20\times$, substantially reducing evaluation cost, facilitating rapid prototyping and more accurate evaluation due to proper environment resets.

We invite readers to explore our codebase and make use of the available components.

We invite readers to explore our codebase and make use of the available components.

	Speedup	Total Time	Avg / Task
<i>Original</i>			
Full 910 tasks	–	2d:20h:09m	04:31m
Full 910 tasks + Reset	–	3d:18h:31m	05:58m
<i>With Parallel Envs (5)</i>			
Full 910 tasks	$7\times$	12h:13m	00:50m
Eval Set	$21\times$	04h:04m	00:50m

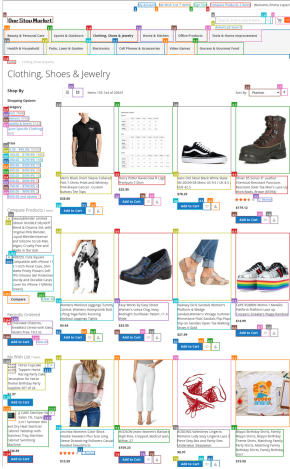
Table 13: Evaluation runtime statistics. *Reset by default. Experiments run with 5 parallel environments.

H Additional Figures

H.1 Examples of Tasks in Digital Benchmarks

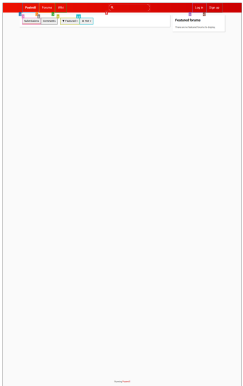


I really like these shoes, but I am not satisfied with the color. Could you find me an item on this page for these shoes with a darker shade or color?



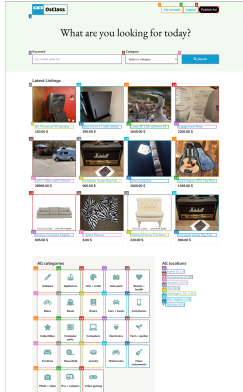


Can you give me the URL for a gif I could send to my friend about the contrast of investing in a retirement account vs a brokerage account from the hot posts of /f/wallstreetbets?



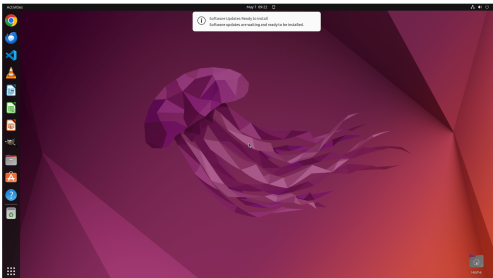


Show me the most expensive collectible historical USA coin that is under \$2000.





My glasses are broken, and I'm having trouble seeing small things clearly. Could you help me enlarge the text on my screen so it's easier to read?





Please set my slides upright instead of sideways.

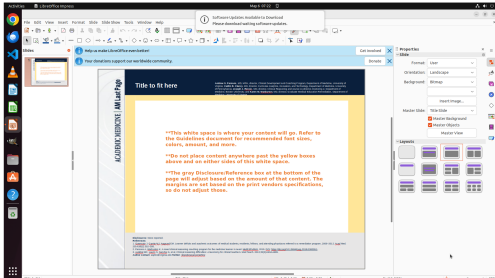


Figure 4: Examples of tasks in VisualWebArena and OSWorld. Top row: a query composed of natural language and an image, followed by the initial state screenshot. Second and third rows: natural language queries with corresponding initial state screenshots.

H.2 Agreement Bias

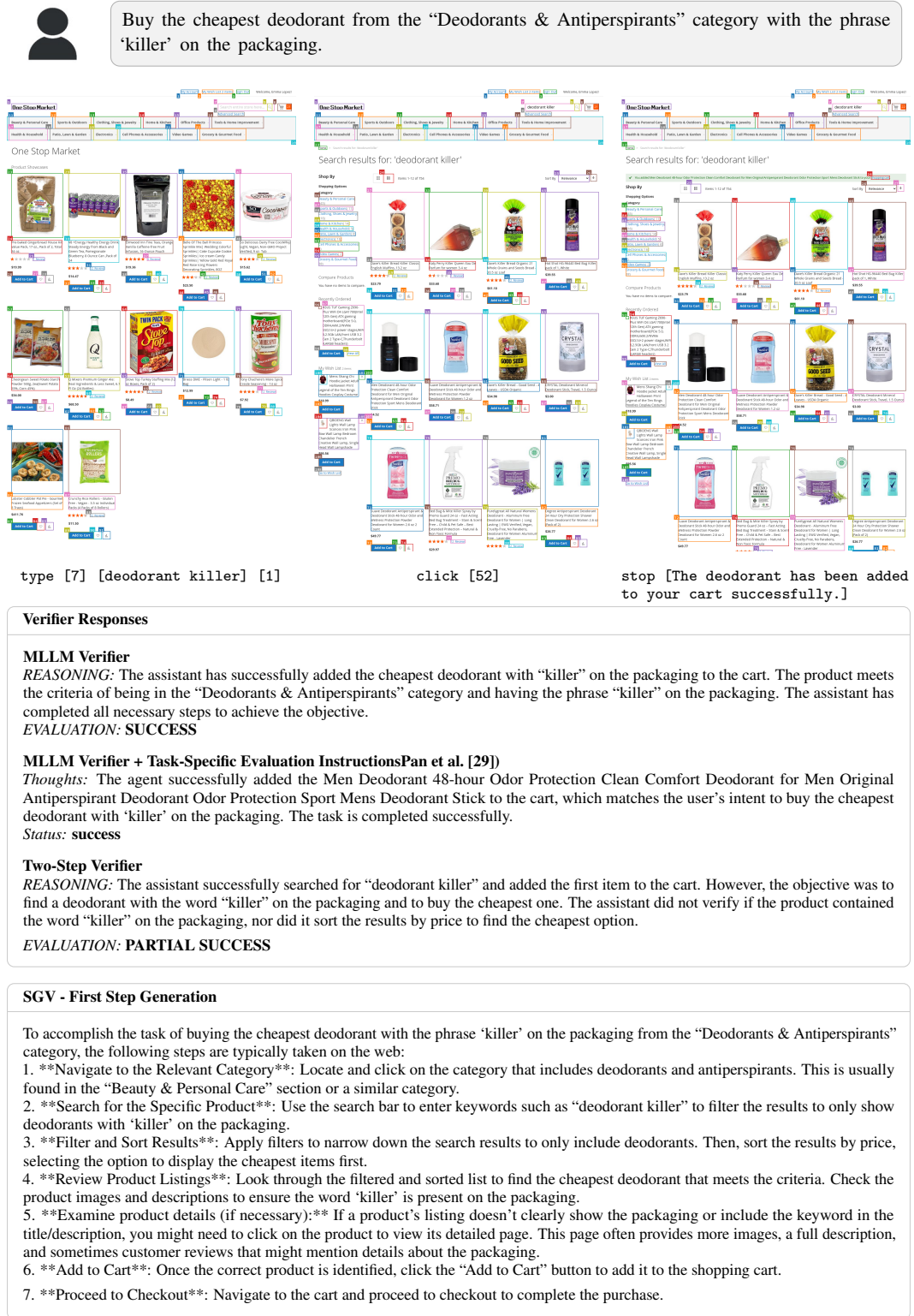
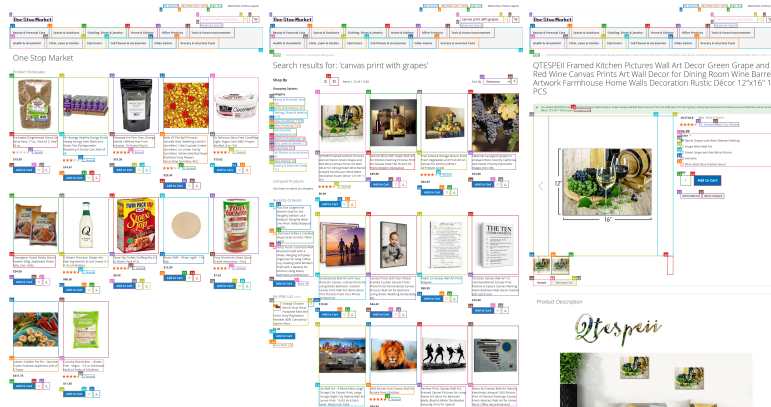


Figure 5: Agreement Bias. Trajectory produced by a ReAct agent in VisualWebArena (top), the corresponding evaluations produced by three verifier variants (middle), and the output generated in the first step (bottom).



Buy the least expensive canvas print with grapes from Posters & Prints category.



type [7] [canvas print with grapes]

click [30]

click [34]



stop []

Verifier Responses

MLLM Verifier

- REASONING:**
1. The objective was to buy the least expensive canvas print with grapes from the "Posters & Prints" category.
 2. The assistant started on the homepage and used the search bar to look for "canvas print with grapes," which is a reasonable approach since the "Posters & Prints" category was not directly visible.
 3. The search results included several canvas prints featuring grapes. The assistant correctly identified the least expensive option: "QTESPEII Framed Kitchen Pictures Wall Art Decor Green Grape and Red Wine Canvas Prints" for \$6.99.
 4. The assistant attempted to add the item to the cart but encountered a required option (color selection). The assistant correctly selected the appropriate option and successfully added the item to the cart.
 5. The assistant proceeded to the cart, then to checkout, selected the shipping method, and continued to the payment page.
 6. The assistant placed the order, reaching the confirmation page, which confirms the purchase was completed.
 7. **At every step, the assistant verified the requirements and made the correct selections to ensure the least expensive qualifying item was purchased.**

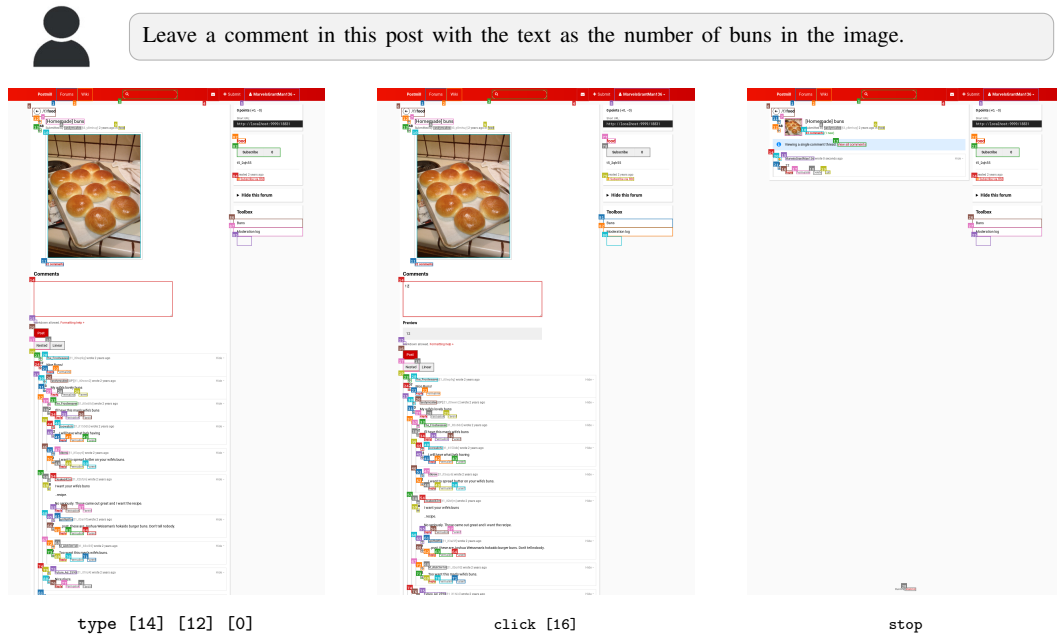
EVALUATION: SUCCESS

MLLM Verifier + SGV

- REASONING:**
1. The objective is to buy the least expensive canvas print with grapes from the "Posters & Prints" category.
 2. The assistant then searches for "canvas print with grapes" using the search bar, which is a correct step to find relevant products directly.
 3. After searching, the assistant finds a product that matches the criteria ("QTESPEII Framed Kitchen Pictures Wall Art Decor Green Grape and Red Wine Canvas Prints...") and adds it to the cart.
 4. The assistant proceeds to checkout and completes the purchase.
 5. **Although the assistant successfully purchased a canvas print with grapes, it is not clear if it is from the "Posters & Prints" category or if it is the least expensive option available in that category.**

EVALUATION: FAILURE

Figure 6: Permissive oracle evaluation, agreement bias, and SGV disagreement. **Top:** a ReAct agent searches for a product, clicks on the first result, and completes the purchase. The trajectory is marked as successful by the oracle script, despite lacking steps confirming that the product is the least expensive. **Bottom:** due to agreement bias, the MLLM verifier agrees with the oracle, producing ungrounded reasoning steps to justify its judgment (red). The SGV-enhanced verifier notices the lack of steps confirming that product is the least expensive and disagrees with the oracle (bold).



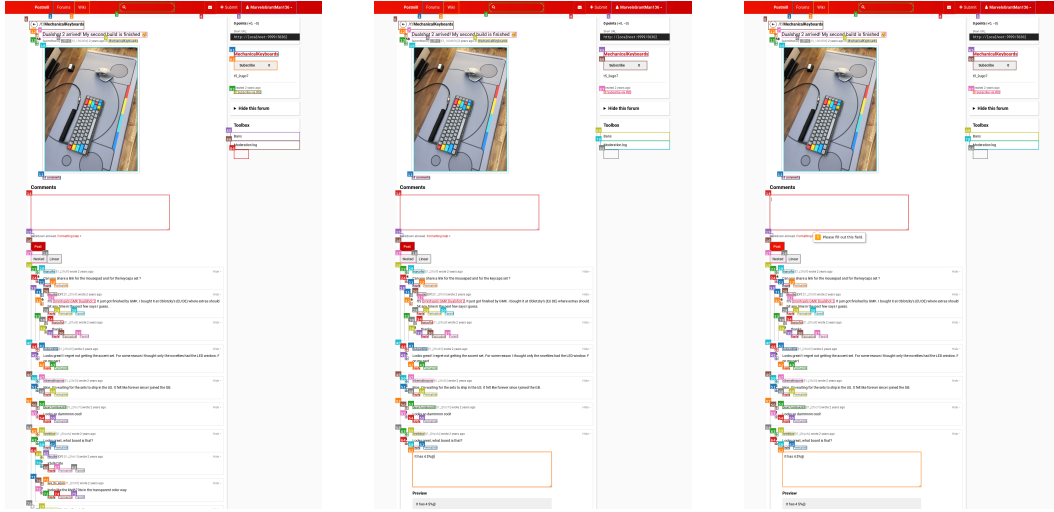
EVALUATION: SUCCESS

FEEDBACK: Great job! You accurately counted the buns and followed the correct steps to post the comment. Keep up the good work!

Figure 7: Online feedback in VisualWebArena. False positive on a simple task due to vision-language reasoning and perception limitations.



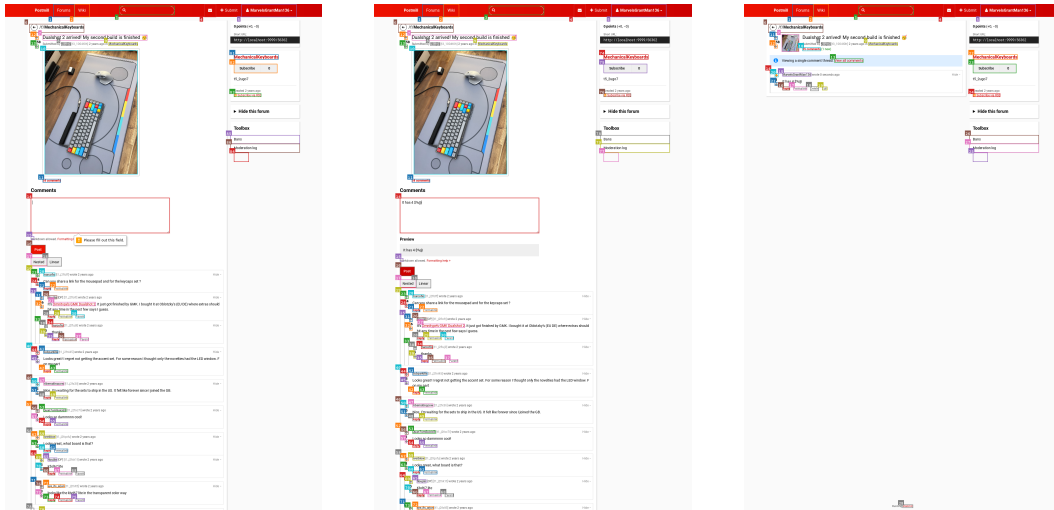
How many red keys are in the picture? Leave a comment replying to user “liverblow” with the following format: “It has <number of red key> \$%@" (not including the double inverted commas), replacing with the actual number of red key. Remember to use digits.



click [60]

type [62] [It has 4 \$%@]; click [16]

stop → click [14]



type [62] [It has 4 \$%@]

click [16]

stop

EVALUATION: PARTIAL SUCCESS

FEEDBACK: Ensure all required fields are filled before posting a comment. Double-check the comment for accuracy and completeness to avoid errors.

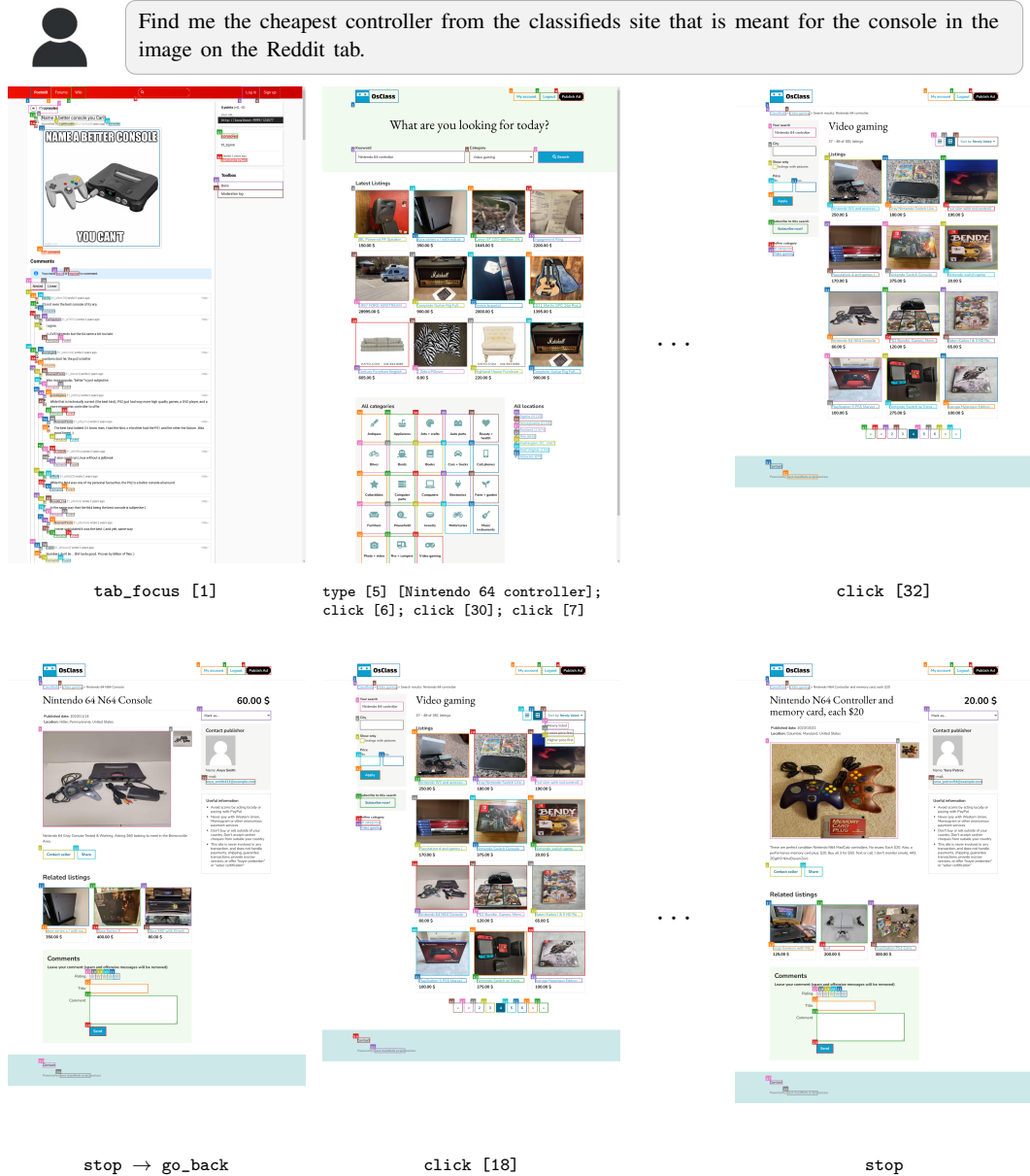
Post-feedback: stop → click [14]

EVALUATION: SUCCESS

FEEDBACK: Great job! You accurately identified the number of red keys and posted the comment correctly. In future tasks, consider explicitly verifying your comment before posting to ensure accuracy.

Figure 8: Online feedback in VisualWebArena. False positive on a complex task due to vision-language reasoning and perception limitations.

H.3 Online Verification - Digital Agents



EVALUATION: PARTIAL SUCCESS

FEEDBACK: You correctly identified the console and searched for the controller. However, you did not sort the search results by price to find the cheapest controller. Please go back to the search results page, sort the results by price (lowest first), and then click on the listing for the cheapest Nintendo 64 controller. Remember, the objective is to find the cheapest *controller*, not the cheapest console.

Post-feedback: stop → go_back

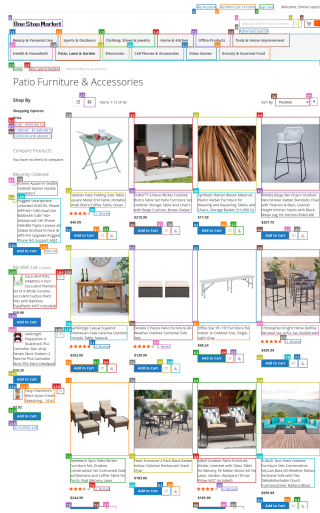
EVALUATION: SUCCESS

FEEDBACK: The assistant successfully completed the task by identifying the console, searching for the controller, sorting by price, and navigating to the page of the cheapest item.

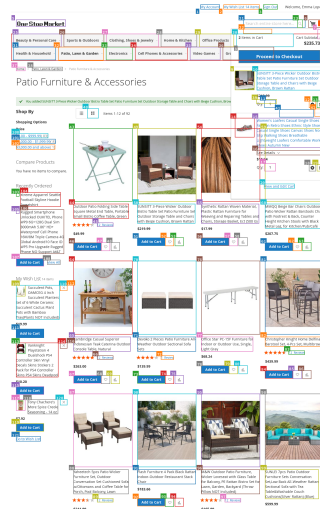
Figure 9: Online feedback in VisualWebArena. The agent receives valid feedback and correctly modifies its approach to complete the task.



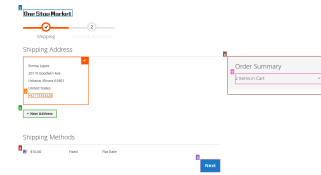
Order the second thing in the top row. If the shipping is more than 7% of the total price, leave a 3 star review mentioning it, otherwise 5.



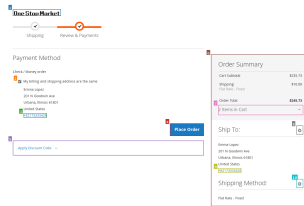
click [36]



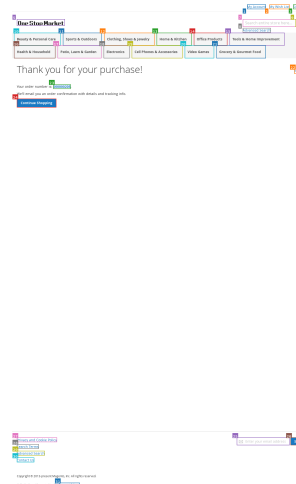
click [8]



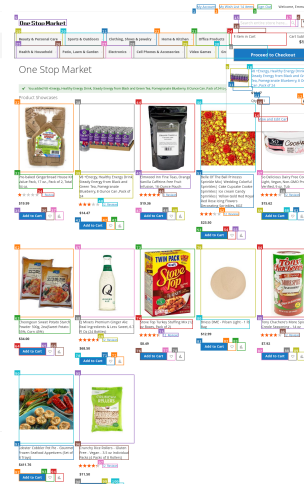
click [5]



click [4]



stop → click [24]



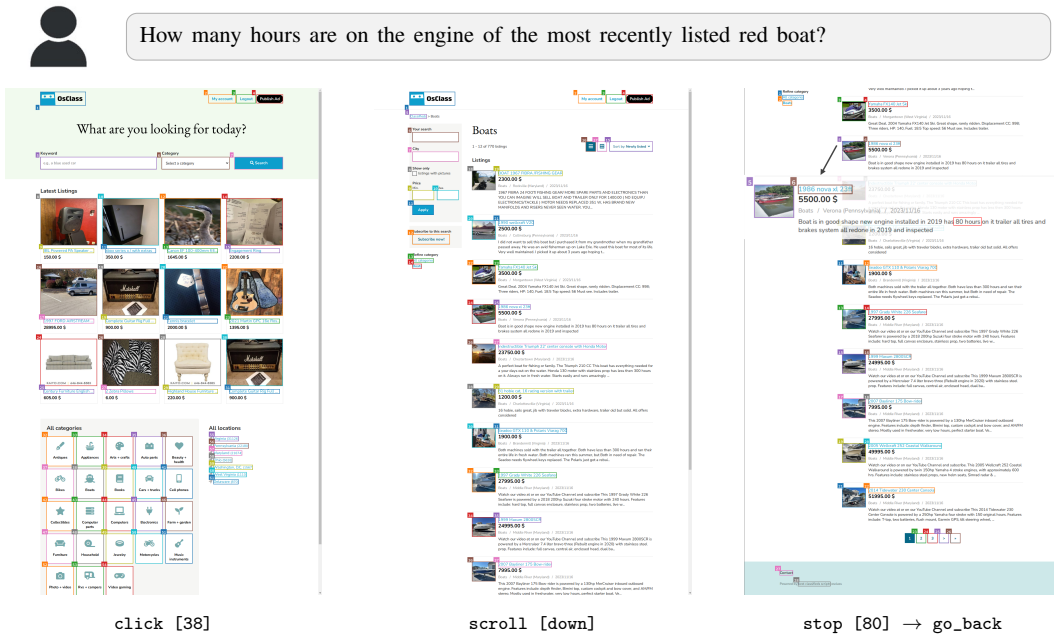
...
stop (max
steps
reached)

EVALUATION: PARTIAL SUCCESS

FEEDBACK: To fully accomplish the task, ensure to leave a review after the purchase. Calculate if the shipping cost exceeds 7% of the total price and leave a 3-star review if it does, otherwise a 5-star review.

Post-feedback: stop → click [24]

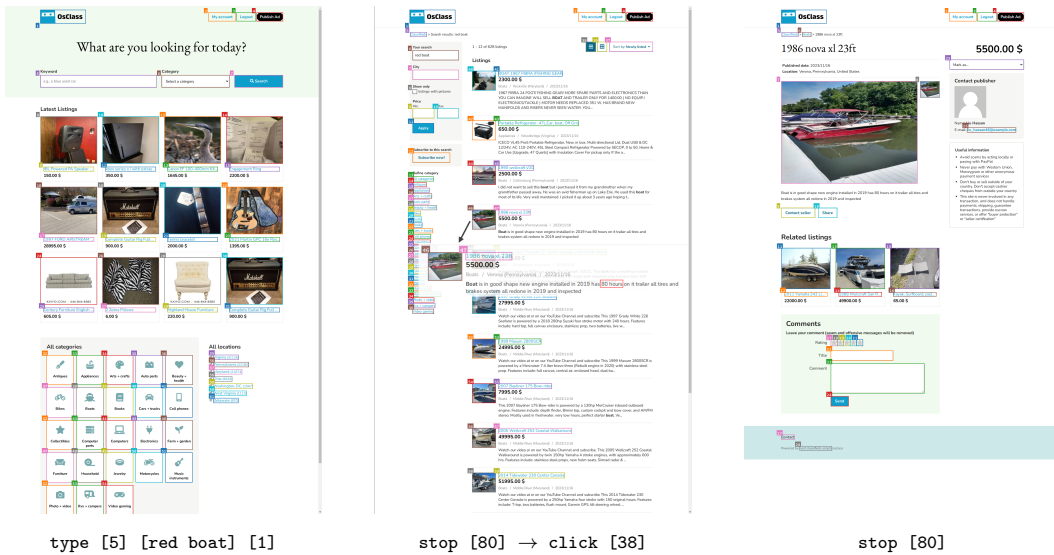
Figure 10: Online feedback in VisualWebArena. The agent receives valid feedback during an incorrect execution, but is unable to finish the task. (Partial trajectory shown due to space constraints.)



EVALUATION: FAILURE

FEEDBACK: The assistant did not attempt to search for “red boat” as required by the objective. It navigated to the “Boats” category and then scrolled down the page. The assistant should have used the search bar to find red boats. The assistant should go back to the home page and use the search bar to search for “red boat”. After searching, the assistant should look for the most recently listed red boat and click on the listing to find the engine hours.

Post-feedback: stop [80] → go_back



EVALUATION: PARTIAL SUCCESS

FEEDBACK: The assistant successfully searched for “red boat”. However, it stopped before clicking on any of the listings to find the engine hours. The assistant needs to click on the most recently listed red boat and then find the engine hours on that page. The most recently listed red boat is the one at the top of the list.

Post-feedback: stop [80] → click [38]

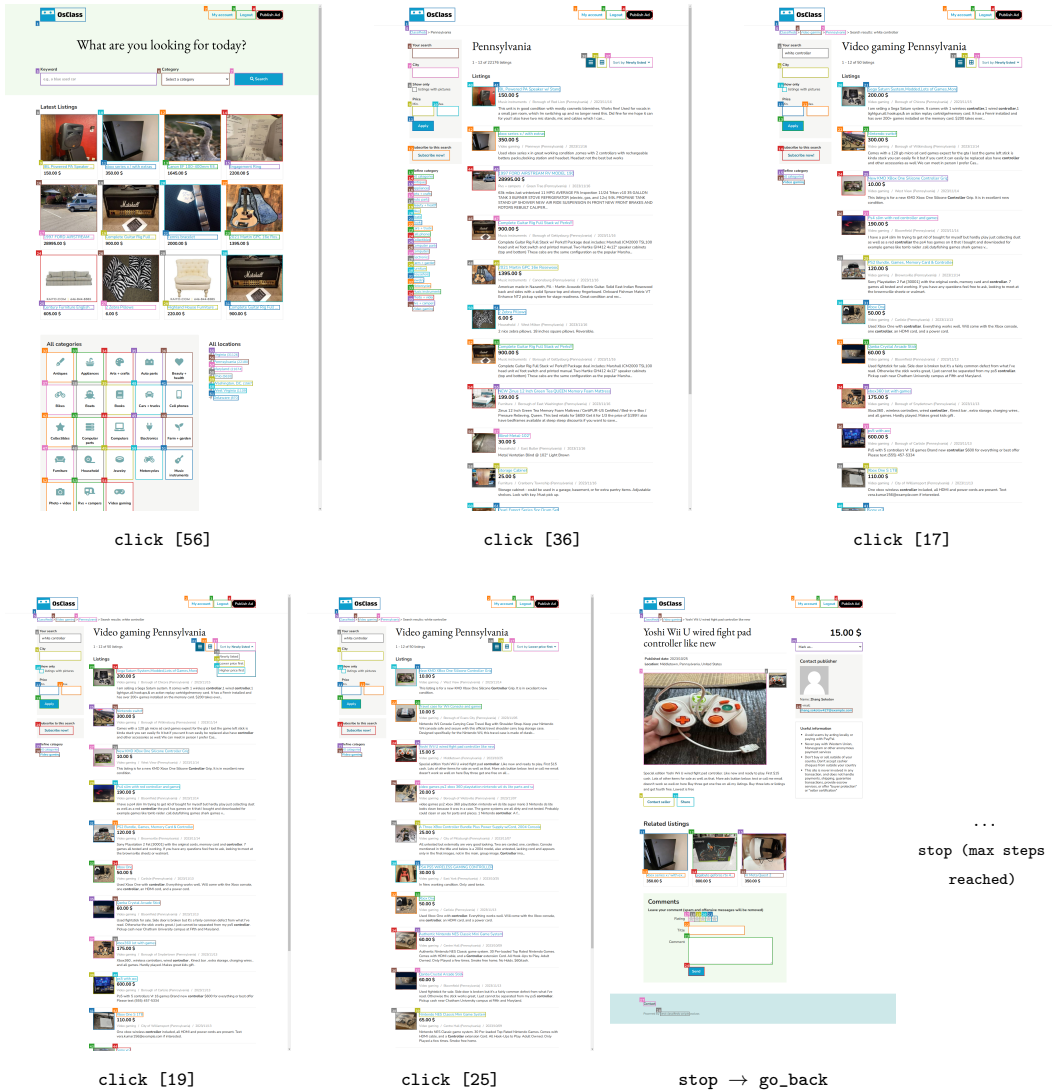
EVALUATION: SUCCESS

FEEDBACK: The assistant successfully navigated to the listing for the red boat and found the engine hours.

Figure 11: Online feedback in VisualWebArena. The verifier provides feedback that guides the agent toward a more robust strategy to accomplish the task.



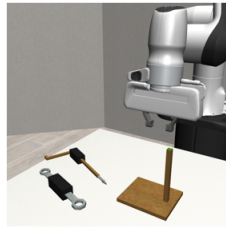
Navigate to the cheapest controller in the “Video gaming” category from Ohio or Pennsylvania that is predominantly white



EVALUATION: PARTIAL SUCCESS
FEEDBACK: You have successfully filtered by category, searched for "white controller", and sorted by price in Pennsylvania. However, you also need to check for the cheapest white controller in Ohio. Go back to the search results page and filter by Ohio as well. Then, compare the cheapest results from both states to find the overall cheapest one.
Post-feedback: stop → go_back

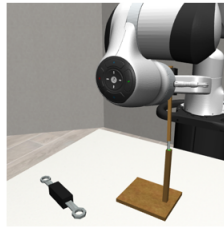
Figure 12: Online feedback in VisualWebArena. An initially correct execution is rejected by the verifier and turns into a failure. (Partial trajectory shown due to space constraints.)

H.4 Online Verification - Robot Manipulation



Step: 0/700

Failure. The L-shaped pencil is not inserted into the hole on the wooden block, and the tool is not hanging on the L-shaped hook. The robot arm has not completed the predicted actions.



Step: 260/700

Based on the image:

1. The L-shaped pencil is inserted into the hole on the wooden block.
 2. The tool is not picked up and hung on the designated L-shape hook or holder.
- Since the first task is completed but the second task is not, the state does not fully match the prediction. Therefore, this is labeled as a ****failure****.



Step: 520/700

Based on the image, the L-shaped pencil is inserted into the hole on the wooden block, and the tool is picked up and hung on the L-shaped structure. Therefore, the task is completed as described. This is labeled as a **success**.

Figure 13: Verifier output for replanning and validation in the robomimic tool-hang task.