MUSEAGENT: A MULTIMODAL REASONING AGENT WITH STATEFUL EXPERIENCES

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal agents have shown growing ability to interleave tool use with multimodal observations for complex visual reasoning, yet effectively reusing experiences from past multimodal interactions remains challenging. Existing memory paradigms struggle in complex multimodal environments: coarse trajectories flood the context window with visual redundancy while obscuring fine-grained errors at intermediate steps; ungrounded scalar feedback offers no procedural guidance for visual tool execution; and holistic representations of multimodal states conflate visual appearance with tool histories, misguiding experience retrieval. To resolve these dilemmas, we introduce MuSEAgent, a multimodal reasoning agent with stateful experiences that reframes multimodal experience reuse across three foundational dimensions: (1) granularity, decomposing multimodal interaction trajectories into atomic state-action transitions anchored to intermediate decision states; (2) supervision, synthesizing outcome-conditioned natural-language visual tactics via hindsight reasoning to provide explicit procedural rules; and (3) representation, organizing heterogeneous multimodal states into a compositional multi-view space, queried via an iterative Deep-and-Wide search to dynamically align visual observations, task semantics, and tool execution histories. Extensive evaluations across four demanding multimodal benchmarks demonstrate that MuSEAgent consistently elevates diverse base models across parameter scales, surpassing trajectory-level baselines by up to 7.97 points and outperforming state-level scalar-value alternatives by 10.42 to 12.00 points. Remarkably, stateful experiences transfer robustly to unseen benchmarks without in-domain interaction, outperforming matched trajectory-level transfer by 7.8 points.

1 INTRODUCTION

Multimodal agents have achieved significant progress in perception and deliberate reasoning across heterogeneous modalities (He et al., 2024; Xie et al., 2024; Yang et al., 2023). A core challenge facing these agents is learning to effectively exploit tools in order to navigate complex, multi-step environments for complex visual reasoning (Lin & Xu, 2025). To tackle complex visual tasks—such as high-resolution images with minute targets or multi-object spatial configurations—agents rely on multi-step tool execution, such as cropping, zooming, localization, and optical character recognition, to iteratively gather visual evidence (Wu & Xie, 2024; Wang et al., 2025; Wei et al., 2026; Zhang et al., 2024). In such long-horizon environments, retrieving and learning from past interaction experiences stored in a memory bank has emerged as a pivotal paradigm for improving autonomous decision-making and generalization without parameter updates (Packer et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2023). However, effectively representing, supervising, and retrieving experiential knowledge in multimodal decision-making introduces fundamental challenges that existing agent memory frameworks fail to resolve.

Realizing effective experience reuse in multimodal environments presents three critical bottlenecks. First, at the level of *experience granularity*, existing paradigms predominantly rely on trajectory-level episodic memories (Shinn et al., 2023; Zhao et al., 2024), bundling multi-turn multimodal interactions into monolithic trajectories. In multimodal domains, visual inputs carry much lower information density than textual tokens, such that retrieving entire interaction trajectories injects extensive visual-textual redundancy, diluting the context window with intermediate multimodal observations (Liu et al., 2024; Chang et al., 2024) while obscuring fine-grained errors at intermediate

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

steps where state-specific tactical knowledge is needed. Second, at the level of *supervision semantics*, turning to state-level memories via classical Markov Decision Process formulations, as exemplified by Memento (Zhou et al., 2025), relies on scalar values that break down in long-horizon multimodal reasoning: intermediate rewards are absent, and stochastic tool executions trigger catastrophic credit-assignment collapse under sparse terminal feedback, forcing such methods to retain only trajectory-end transitions while leaving scalar returns with zero procedural guidance on visual tool invocation. Third, at the level of *state representation*, encoding heterogeneous multimodal states spanning user instructions, visual inputs, task descriptors, and tool execution histories into holistic representations conflates visual appearance with discrete tool execution histories, leading to spurious experience retrieval.

To address these challenges, we introduce **MuSEAgent**, a multimodal reasoning agent with stateful experiences that systematically reframes multimodal experience reuse across granularity, supervision, and representation (Figure 1). MuSEAgent establishes a principled paradigm shift along all three axes: first, for granularity, it decomposes multimodal interaction trajectories into atomic state–action transitions, anchoring reusable experiential units directly to intermediate decision states rather than monolithic trajectories; second, for supervision, it replaces uninformative scalar returns with an outcome-conditioned hindsight reasoning model that filters and annotates high-value transitions with explicit natural-language visual tactics, translating successful breakthroughs into procedural rules and intermediate mistakes into cautionary warnings; and third, for representation, it projects heterogeneous states into a compositional multi-view representation space spanning 15 semantic viewpoints, dynamically aligning visual observations, task semantics, and tool execution histories to resolve representation entanglement.

Operating over this structured experience bank, MuSEAgent guides agent actions at inference time through an iterative Deep-and-Wide search. Rather than querying a fixed monolithic vector, the agent adaptively selects semantic viewpoints to query the experience bank: *Wide Search* retrieves cross-task strategic knowledge across candidate viewpoints to expose the agent to diverse decision contexts, while *Deep Search* iteratively refines retrieval across complementary viewpoints within a single reasoning step, progressively aligning visual observations, task semantics, and tool-syntax constraints before committing to the next action.

Extensive evaluations across four demanding multimodal benchmarks—V* Bench (Wu & Xie, 2024), MME-RealWorld-Lite (Zhang et al., 2024), Zoom-Bench (Wei et al., 2026), and HR-Bench (Wang et al., 2025)—and three base model scales demonstrate the decisive advantages of MuSEAgent. Across all architectures, MuSEAgent consistently elevates base model performance, surpassing trajectory-level baselines by up to +7.97 points on Qwen3-VL-32B-Instruct, +3.25 points on Qwen3-VL-235B-A22B-Instruct (Bai et al., 2025), and +3.81 points on Qwen3.5-397B-A17B (Qwen Team, 2026), and, notably, outperforming state-level scalar-value baselines from Memento (Zhou et al., 2025) by 10.42–12.00 points. Systematic component-wise disentanglement confirms that transition granularity, natural-language supervision, and multi-view state indexing each provide indispensable, statistically significant gains, yielding successive improvements of 4.17 to 6.25 points over their respective counterparts. Finally, these advantages persist under self-hindsight where the agent evaluates its own trajectories, produce non-overlapping 3-seed confidence intervals, and transfer robustly across unseen benchmarks with a 7.8-point advantage over matched trajectory-level transfer.

2 Related Work

Multimodal Agent. Multimodal agents shift visual reasoning from static inference to interactive tool-using decision-making (Xi et al., 2025; Wang et al., 2024). Inspired by ReAct (Yao et al., 2022), prior works coordinate vision modules or structured programs for visual reasoning (Yang et al., 2023; Gupta & Kembhavi, 2023; Surís et al., 2023; Shen et al., 2023), but typically append entire interaction histories (He et al., 2024; Zhang et al., 2025), introducing contextual redundancy that undermines long-horizon consistency. MuSEAgent instead formalizes agentic multimodal reasoning as a Stateful Experience-Based MDP, converting interaction histories into granular state-level experiences.

Experience-driven Agent Learning. Recent memory-augmented agents reuse past trajectories (Shinn et al., 2023; Wang et al., 2023; Zhao et al., 2024; Packer et al., 2023), primarily via

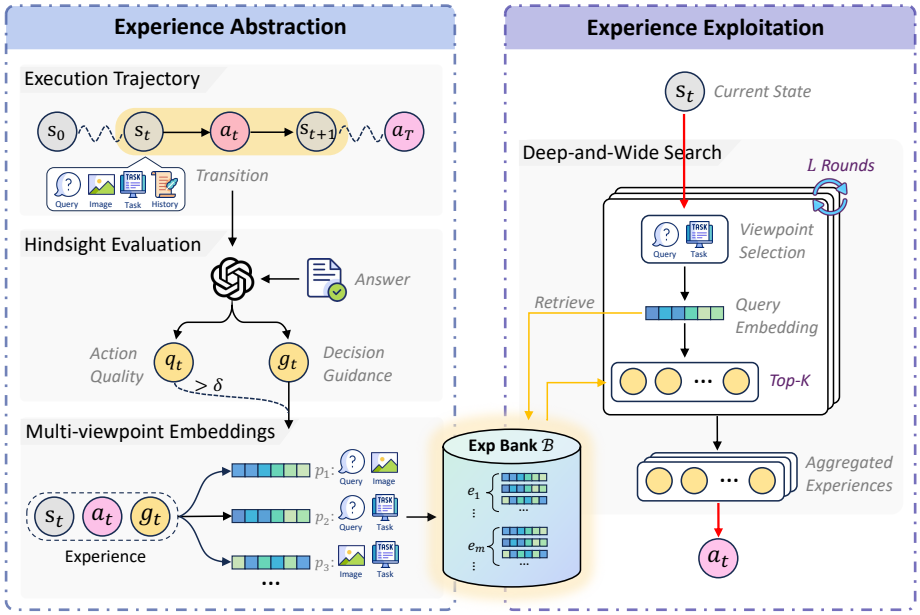


Figure 1: The overview of MuSEAgent. The framework consists of two phases: (1) **Experience Abstraction**, which extracts state-level experiences via hindsight evaluation and builds multi-viewpoint embeddings for each experience; (2) **Experience Exploitation**, where the agent performs a Deep-and-Wide search over the experience bank to determine the next action.

coarse episode retrieval (Yao et al., 2023; Zhu et al., 2023) that injects context noise and offers only coarse task analogies ill-suited for fine-grained multimodal reasoning (Yao et al., 2022; Yang et al., 2023). A complementary direction models agent memory within MDP frameworks with value-guided case retrieval (Zhou et al., 2025); however, sparse environmental rewards and stochastic tool executions cause terminal credit assignment to collapse in multimodal domains, restricting existing instantiations (Zhou et al., 2025) to trajectory-end transitions. MuSEAgent resolves these bottlenecks by supervising transitions with hindsight-distilled natural-language guidance instead of scalar returns.

3 MuSEAgent

We present MuSEAgent (Figure 1), a novel framework that enhances multimodal agent reasoning by abstracting and exploiting stateful experiences. MuSEAgent operates as a multi-step tool-using agent that iteratively invokes multimodal tools and updates its execution history at each reasoning step. We formulate this process as a Stateful Experience-Based MDP (Section 3.1), abstract trajectories into reusable state-level experiences via hindsight reasoning (Section 3.2), and exploit them online through a Deep-and-Wide search (Section 3.3).

3.1 Problem Formulation

We formally characterize the sequential decision-making process of MuSEAgent as a Stateful Experience-Based MDP, in which new problems are solved by retrieving fine-grained experiences attached to similar decision states, rather than entire historical trajectories.

Definition 3.1 (Stateful Experience-Based MDP)**.** Let $u \in \mathcal{U}$ denote the initial natural language instruction. The agent interacts with multimodal observations and execution contexts across sequential reasoning steps. The decision state at step t is defined as $s_t = (u, v_t, d, H_t)$, where u is the user instruction, v_t denotes current visual input, d is an optional task descriptor, and H_t is execution history up to step t. The action space is $a_t \in \mathcal{A}_{\text{tool}} \cup \mathcal{A}_{\text{exp}}$, spanning task execution tools $\mathcal{A}_{\text{tool}}$ and experience retrieval $\mathcal{A}_{\text{exp}}$. State transitions are stochastic due to tool outputs and visual observation noise. An interaction trajectory is denoted $\tau = (s_1, a_1, \ldots, s_T, a_T)$ with horizon T, and binary outcome $o \in \{0, 1\}$ indicating final answer correctness. Each transition serves as an experience unit.

This formulation grounds multimodal agent reasoning around fine-grained decision states, highlighting why experiential units should be anchored to atomic transitions (s_t, a_t, s_{t+1}) rather than monolithic episodes. Furthermore, it formalizes a fundamental dilemma in long-horizon multimodal reasoning: visual tool executions are stochastic and lack intermediate environmental rewards, rendering terminal credit assignment via scalar values inherently ambiguous (Zhou et al., 2025). To overcome this limitation, we propose supervising each transition not with ungrounded scalar returns, but with hindsight-distilled natural-language visual guidance g_t, transforming outcome-conditioned evaluations into explicit procedural rules (Section 3.2).

3.2 Stateful Experience Abstraction via Hindsight Reasoning

Experience Abstraction. To construct a compact and noise-resistant experience bank, we first decompose agent trajectories into atomic state–action transitions (s_t, a_t, s_{t+1}). These transitions serve as candidate experience units that capture localized decision contexts within long multimodal interaction trajectories, including steps where tools return erroneous or unexpected outputs, both informative for hindsight reasoning. Each transition is evaluated using a hindsight reasoning model $\mathcal{Q}_\phi$ conditioned on the realized trajectory outcome o, producing $(q_t, g_t) = \mathcal{Q}_\phi(s_t, a_t, s_{t+1} \mid \tau, o)$, where $q_t \in [0, 10]$ reflects estimated decision quality at state s_t, and g_t summarizes the key decision tactic.

Compositional State Representation. Multimodal agent states are heterogeneous, including textual instructions, visual observations, task descriptors, and execution histories, which provide complementary signals for experience retrieval. To enable flexible querying over such heterogeneous states, we organize each state into multiple semantic viewpoints. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_M\}$ enumerate all $M = 15$ non-empty compositions of the four state components (u, v_t, d, H_t) (e.g., $p_i(s_t) = (u, v_t)$ for perceptual intent, $p_i(s_t) = (d, H_t)$ for tool execution history, or $p_i(s_t) = (u, v_t, d, H_t)$ for full task context). During offline indexing, each experience is associated with multiple embeddings derived under each viewpoint:

$$\mathbf{z}_t^{(i)} = f_\theta(p_i(s_t)), \quad p_i \in \mathcal{P}, \tag{1}$$

where f_θ denotes a multimodal embedding model. At inference time, the agent policy adaptively selects an appropriate viewpoint to query the bank. Although states encompass execution history H_t, querying across decoupled viewpoints prevents retrieval from collapsing into rigid trajectory matching: the agent selectively matches on localized sub-state cues rather than full paths (Section 4.3 and Appendix K).

Graph-based Quality Score Smoothing. Because the hindsight reasoner evaluates each transition independently, raw scores q_t can be noisy: semantically similar states may receive divergent ratings due to stochastic variation in reasoner scoring (Zheng et al., 2023). Rather than relying on repeated sampling, we propagate quality scores over an affinity graph constructed from candidate transitions, leveraging neighbourhood consensus for reliable quality estimates. Let $\mathcal{C} = \{(s_t, a_t, g_t, q_t)\}$ denote candidate transitions. We construct a symmetric affinity matrix $\mathbf{W}$ with entries $W_{ij} = \frac{1}{M}\sum_{k=1}^{M} \mathrm{sim}(\mathbf{z}_i^{(k)}, \mathbf{z}_j^{(k)})$ and form the row-normalized transition matrix $\mathbf{S} = \mathbf{D}^{-1}\mathbf{W}$, where $\mathbf{D} = \mathrm{diag}(\mathbf{W}\mathbf{1})$. Given initial scores $\mathbf{q}^{(0)} = [q_1, \ldots, q_{|\mathcal{C}|}]^\top$, we compute the smoothed scores as the fixed point:

$$\mathbf{q}^* = (1 - \alpha)\,(\mathbf{I} - \alpha\mathbf{S})^{-1}\,\mathbf{q}^{(0)}, \tag{2}$$

where $\alpha \in [0, 1)$ balances fidelity to raw scores against neighbourhood consensus.

Lemma 3.2 (Convergence and Optimality of Quality Smoothing)**.** *For any $\alpha \in [0, 1)$ and initial score vector $\mathbf{q}^{(0)} \in \mathbb{R}^{|\mathcal{C}|}$, the smoothed score $\mathbf{q}^*$ in Eq. equation 2 exists, is unique, and is the unique minimizer of the Laplacian-regularized energy:*

$$\mathcal{E}(\mathbf{q}) = \frac{1}{2}\sum_{i,j} W_{ij}(q_i - q_j)^2 \;+\; \frac{1-\alpha}{\alpha}\sum_i D_{ii}(q_i - q_i^{(0)})^2. \tag{3}$$

The proof is deferred to Appendix A. The smoothed scores $\mathbf{q}^*$ replace raw scores in the admission criterion $\mathcal{B} = \{e_t = (s_t, a_t, g_t) \mid q_t^* \geq \delta\}$ with threshold $\delta = 5.0$. Each retained transition forms a reusable unit (s_t, a_t, g_t) (human verification in Appendix I), computed once offline with negligible overhead.

The complete procedure is summarized in Algorithm 1 of Appendix B.

3.3 Stateful Experience Exploitation via Deep-and-Wide Search

At inference step t, before committing to the next action, the agent consults the experience bank to obtain additional decision guidance. The agent policy $\pi(\cdot \mid s_t)$ first selects a semantic viewpoint $p_i \in \mathcal{P}$ based on the current state s_t. Since each experience e_i in the experience bank $\mathcal{B}$ is indexed with viewpoint-specific embeddings, the selected viewpoint determines which embedding index is queried. The agent then constructs a query embedding and retrieves relevant experiences from the corresponding viewpoint index. To balance generalization and precise state matching, we adopt two complementary retrieval strategies: *Wide Search*, and *Deep Search*.

Wide Search. To identify broadly relevant experiences, Wide Search performs breadth-oriented retrieval under the currently selected viewpoint. Given the query embedding $\mathbf{q}_t^{(i)} = f_\theta(p_i(s_t))$ constructed under the selected viewpoint p_i, the agent retrieves the Top-K most relevant experiences from the corresponding viewpoint index according to cosine similarity:

$$\mathcal{E}_{\text{wide}}(s_t) = \text{Top-K}_{e_m \in \mathcal{B}} \ \text{sim}\left(\mathbf{q}_t^{(i)}, \mathbf{z}_m^{(i)}\right), \tag{4}$$

where $\mathbf{z}_m^{(i)}$ denotes the stored embedding of experience e_m under the selected viewpoint, and K controls the retrieval breadth. By retrieving multiple experiences with similar contextual signals, Wide Search exposes the agent to diverse decision contexts and helps identify reusable reasoning patterns that generalize across related tasks.

Deep Search. When broader retrieval does not provide sufficiently precise guidance, Deep Search performs iterative refinement by querying the experience bank under multiple semantic viewpoints. At each refinement round j, the agent selects a viewpoint $p_{i_j} \in \mathcal{P}$ with query embedding $\mathbf{q}_t^{(i_j)} = f_\theta(p_{i_j}(s_t))$ and retrieves the most relevant experience under that viewpoint:

$$\mathcal{E}_{\text{deep}}(s_t) = \bigcup_{j=1}^{L} \text{Top-1}_{e_m \in \mathcal{B}} \ \text{sim}\left(\mathbf{q}_t^{(i_j)}, \mathbf{z}_m^{(i_j)}\right), \tag{5}$$

where L denotes the maximum number of refinement rounds. Each round emphasizes a different aspect of the state representation, allowing the agent to progressively align task intent, perceptual observations, and historical tool usage patterns. For example, the agent may initiate retrieval under a purely visual viewpoint to resolve bounding-box ambiguities, and subsequently issue a secondary query under the execution histories (i.e., tool) viewpoint to validate tool parameter syntax, complementing Wide Search.

Unified Deep-and-Wide Search. In practice, experience search is performed iteratively under viewpoints selected by the agent: at round j, viewpoint p_{i_j} is chosen, and the agent retrieves Top-K relevant experiences under that view:

$$\mathcal{E}_{\text{deep-wide}}(s_t) = \bigcup_{j=1}^{L} \text{Top-K}_{e_m \in \mathcal{B}} \, \text{sim}\left(\mathbf{q}_t^{(i_j)}, \mathbf{z}_m^{(i_j)}\right), \tag{6}$$

where L is the number of retrieval rounds and K controls the retrieval breadth per round. The retrieved experience strings $\{g_m\}$ are injected into the agent's working context to decide the next action $a_t \sim \pi(\cdot \mid s_t, \mathcal{E}_{\text{deep-wide}})$.

The complete online retrieval-and-decision procedure is summarized in Algorithm 2 of Appendix B. A representative case illustrating both stateful experience abstraction and Deep-and-Wide exploitation is provided in Appendix H.

4 Experiments

By decomposing multimodal interaction trajectories into atomic transitions (§3.2) and indexing them across a compositional multi-view space (§3.3), MuSEAgent enables test-time experience exploitation with step-specific natural-language guidance. To systematically evaluate whether this stateful experience learning paradigm resolves the dilemmas of trajectory redundancy, ungrounded

scalar feedback, and representation entanglement, our empirical investigation addresses four central research questions: **RQ1 (§4.2)** evaluates whether stateful experiences consistently outperform trajectory-level and state-level scalar-value baselines across diverse benchmarks and model scales; **RQ2 (§4.3)** investigates how the depth and breadth of Deep-and-Wide search govern retrieval balance, and what semantic viewpoints drive inference decisions; **RQ3 (§4.4)** examines whether stateful experiences capture modular visual tactics that transfer to held-out benchmarks without in-domain interactions; and **RQ4 (§4.5)** isolates the individual contributions of transition-level granularity, natural-language supervision semantics, and compositional multi-view representations.

4.1 Experimental Setup

Benchmark Datasets. We evaluate on four multiple-choice VQA benchmarks that demand fine-grained visual perception, real-world visual understanding, and complex multimodal reasoning: V* Bench (Wu & Xie, 2024), MME-RealWorld-Lite (Zhang et al., 2024), Zoom-Bench (Wei et al., 2026), and HR-Bench (Wang et al., 2025). Following the standard practice in experience-augmented agents (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2023), where the agent first interacts with a portion of the data to accumulate experiences and is then evaluated on the rest, each benchmark is partitioned into equally sized exploration and evaluation splits (see Table 4 in Appendix C for detailed sample counts): the exploration set collects interaction trajectories for experience bank construction, while the evaluation set benchmarks final accuracy.

Baseline Methods. We compare MuSEAgent against five baselines spanning vanilla reasoning, Tool-Integrated Reasoning (TIR) (Lin & Xu, 2025), trajectory-level experience methods, and state-level value-based memory: Vanilla CoT (Kojima et al., 2022) (single-pass reasoning without tools), ReAct (Yao et al., 2022) (dynamic tool execution without experience reuse), Reflexion (Shinn et al., 2023) (verbal reflections distilled from failed historical trajectories), Expel (Zhao et al., 2024) (trajectory-level insights distilled from both successes and failures), and Memento (Zhou et al., 2025) (state-level value-based memory retaining trajectory-end transitions supervised by scalar returns under matched embedding and retrieval infrastructure). See Appendix D for details.

Implementation Details. We use three agentic MLLMs as base models, Qwen3-VL-32B-Instruct, Qwen3-VL-235B-A22B-Instruct (Bai et al., 2025), and Qwen3.5-397B-A17B (Qwen Team, 2026), with GPT-4o (Hurst et al., 2024) as the default hindsight reasoning model $\mathcal{Q}_\phi$ in Section 4.2, and Qwen3-VL-8B-Embedding (Li et al., 2026) as the retrieval encoder. We set $L = 3$, $K = 3$, and $\delta = 5.0$. Prompts and efficiency details are in Appendix N and Appendix F.

Tool Bank. The agent accesses 13 multimodal tools across visual extraction, spatial manipulation, and cross-modal alignment (Appendix E), shared by all baselines. An audit ruling out external search shortcuts is in Appendix J.

Experimental Protocol. We evaluate under two deployment protocols: (1) *In-domain experience accumulation* (§4.2), where the bank is constructed from the target benchmark's exploration set; and (2) *Cross-dataset transfer* (§4.4), where the bank is pooled from sibling benchmarks held out from the target, evaluating zero-shot experience transfer without in-domain interaction. Accuracy is evaluated on held-out evaluation splits.

4.2 Overall Performance

Table 1 presents the main results across various methods and base models. From these results, we derive three key findings:

Stateful experiences consistently outperform trajectory-level and scalar-value baselines. MuSEAgent achieves the highest average accuracy across all evaluated benchmarks and base models. On Qwen3-VL-32B-Instruct, MuSEAgent reaches 65.30% average accuracy, surpassing the strongest trajectory baselines (Reflexion at 57.33% and Expel at 56.96%) by +7.97 points. Notably, MuSEAgent consistently outperforms the state-level value-based baseline Memento across all three model scales, leading by +7.11 points on 32B (surpassing Memento by +10.42 on V* Bench and +12.00 on HR-Bench), +2.26 on 235B, and +1.60 on 397B on average. Because Memento also operates on state transitions, this performance gap directly confirms that outcome-conditioned natural-language visual tactics provide actionable procedural knowledge that ungrounded scalar values cannot convey in sparse-reward multimodal settings.

Table 1: Performance comparison of MuSEAgent against baselines on various benchmarks under three base models. We report the accuracy (%) for all tasks. Best results are highlighted in **bold**.

Method	Exp. Level	V* Bench			MME-RealWorld-Lite			Zoom-Bench	HR-Bench	Avg.
		Direct Attr.	Rel. Pos.	Micro Avg.	Percep.	Reas.	Micro Avg.			
Qwen3-VL-32B-Instruct										
Vanilla CoT (Kojima et al., 2022)	N/A	27.59	57.89	39.58	28.55	28.53	28.54	31.83	51.00	37.74
w/ ReAct (Yao et al., 2022)	N/A	62.07	84.21	70.83	44.44	36.80	41.46	44.69	66.00	55.75
w/ Reflexion (Shinn et al., 2023)	Trajectory	74.13	76.32	75.00	46.15	34.13	41.46	48.87	64.00	57.33
w/ Expel (Zhao et al., 2024)	Trajectory	70.69	78.95	73.96	42.39	**40.27**	41.56	46.30	66.00	56.96
w/ Memento (Zhou et al., 2025)	State	68.97	84.21	75.00	42.91	36.27	40.31	51.45	66.00	58.19
w/ MuSEAgent	State	**86.21**	**84.21**	**85.42**	**46.32**	38.13	**43.13**	**54.66**	**78.00**	**65.30**
Qwen3-VL-235B-A22B-Instruct										
Vanilla CoT (Kojima et al., 2022)	N/A	24.14	47.37	33.33	33.68	29.87	32.19	36.98	54.00	39.13
w/ ReAct (Yao et al., 2022)	N/A	44.83	68.42	54.17	42.74	35.20	39.79	43.73	63.00	50.17
w/ Reflexion (Shinn et al., 2023)	Trajectory	48.28	86.84	63.54	43.42	31.73	38.85	42.44	69.00	53.46
w/ Expel (Zhao et al., 2024)	Trajectory	53.45	73.68	61.46	42.39	34.67	39.38	43.09	72.00	53.98
w/ Memento (Zhou et al., 2025)	State	50.00	89.47	65.63	41.54	34.67	38.85	43.41	72.00	54.97
w/ MuSEAgent	State	**55.17**	**92.11**	**69.79**	**42.91**	**35.73**	**40.10**	**45.02**	**74.00**	**57.23**
Qwen3.5-397B-A17B										
Vanilla CoT (Kojima et al., 2022)	N/A	39.66	60.53	47.92	39.32	36.80	38.33	39.23	61.00	46.62
w/ ReAct (Yao et al., 2022)	N/A	79.31	71.05	76.04	50.26	44.00	47.81	59.81	75.00	64.67
w/ Reflexion (Shinn et al., 2023)	Trajectory	77.59	73.68	76.04	51.97	43.47	48.65	61.74	72.00	64.61
w/ Expel (Zhao et al., 2024)	Trajectory	81.03	76.32	79.17	53.68	42.93	49.48	59.16	76.00	65.95
w/ Memento (Zhou et al., 2025)	State	82.76	76.32	80.21	53.85	45.33	50.52	65.92	76.00	68.16
w/ MuSEAgent	State	**84.48**	**78.95**	**82.29**	**54.87**	**46.40**	**51.56**	**67.20**	**78.00**	**69.76**

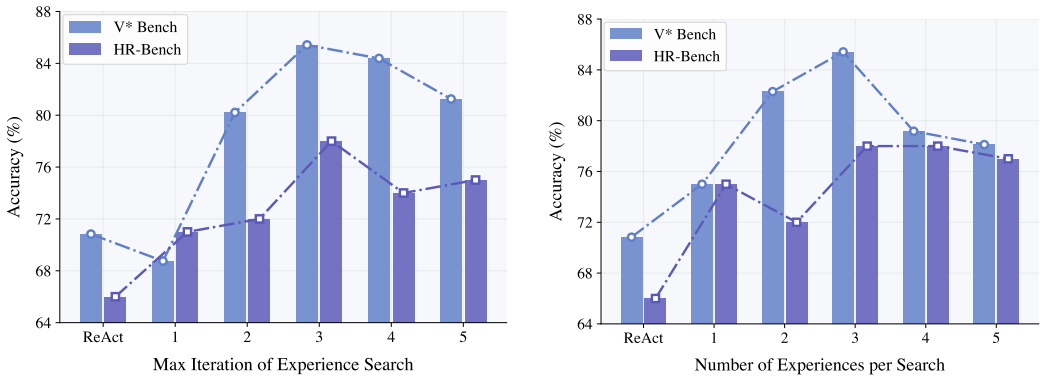
Stateful-experience gains hold across base-model capacities and explain base-model behavior. MuSEAgent improves over the strongest trajectory baseline across all three architectures, yielding +7.97, +3.25, and +3.81 points on 32B, 235B, and 397B models respectively, with absolute accuracy reaching 69.76% on 397B. While Qwen3-VL-235B-A22B-Instruct exhibits lower baseline accuracy across all methods compared to the dense 32B model, an audit of the execution logs reveals that this stems from base-model tool execution inertia rather than retrieval failure: the 235B model skips execution tools entirely and attempts to answer directly from parametric memory in 38%–68% of queries across benchmarks, compared to only 12% for the 32B model. Because fine-grained perception strictly requires spatial cropping and zooming, bypassing tools bottlenecked performance; nevertheless, MuSEAgent still delivers consistent improvements (+3.25 points) over trajectory baselines on this model.

Statistical reliability on small evaluation sets. Because the evaluation splits for V* Bench (96 questions) and HR-Bench (100 questions) are relatively compact, we conduct multi-seed evaluations over three random seeds on Qwen3-VL-32B-Instruct to verify that observed gains are not artifacts of run-to-run variance. As reported in Table 7 of Appendix I, MuSEAgent achieves 84.38 $\pm$ 1.47% on V* Bench and 77.00 $\pm$ 1.41% on HR-Bench, compared to 72.92 $\pm$ 3.07% and 65.00 $\pm$ 2.94% for Expel. The 11–12 point margins between MuSEAgent and Expel are several times the standard deviations, producing non-overlapping confidence intervals across both benchmarks (e.g., Expel's upper bound of 75.99% on V* Bench remains well below MuSEAgent's lower bound of 82.91%). This demonstrates that our performance advantages are statistically robust against decoding and tool-output stochasticity.

4.3 Investigation on Deep-and-Wide Experience Search

To probe how depth and breadth shape stateful retrieval, we scale one dimension at a time on Qwen3-VL-32B-Instruct while fixing the other at 3. As Figure 2 shows, both dimensions yield substantial gains over the zero-experience ReAct baseline and follow a non-monotonic trend: accuracy rises sharply from 0 to 3, peaks at 3 (85% on V* Bench, 78% on HR-Bench), and then slightly declines at 4 or 5, while every configuration still beats the experience-free baseline. Wide Search shows a parallel pattern from 1 to 5 retrievals per query, echoing the depth–breadth duality of the design. All other components are fixed at default to isolate this effect.

This non-monotonicity reveals a key property of stateful retrieval: more experience is not strictly better, since beyond a moderate budget, redundant guidance crowds the context window and dilutes



(a) Deep Search (b) Wide Search

Figure 2: Scaling behavior of Deep-and-Wide search on V* Bench and HR-Bench (Qwen3-VL-32B-Instruct), varying (a) iterations L (K=3) and (b) retrievals K (L=3).

Table 2: Matched cross-dataset experience transfer across held-out sibling exploration trajectories without in-domain interaction on Qwen3-VL-32B-Instruct. Best results are in **bold**.

Method (Cross-Dataset)	Exp. Level	V* Bench	MME-RealWorld	Zoom-Bench	HR-Bench	Avg.
w/ Reflexion (Shinn et al., 2023)	Trajectory	69.79	38.33	47.27	55.00	52.60
w/ Expel (Zhao et al., 2024)	Trajectory	66.67	39.06	44.69	57.00	51.86
w/ MuSEAgent	State	**78.12**	**40.00**	**51.45**	**69.00**	**59.64**

the signal aligned with the current state. The Deep-and-Wide design therefore works because it balances coverage with parsimony, leveraging multiple semantic viewpoints without overwhelming the agent's instruction-following capacity.

Viewpoint Selection Dynamics and Component Drivers. Inference-time viewpoint selection frequencies across benchmarks are concentrated: the top two combinations—query with execution history (48.03%) and query with visual observation (23.97%)—account for over 72% of all retrieval steps, while the full-context view (u, v_t, d, H_t) is selected in only 5.75% of steps (9 combinations account for $< 0.35\%$ each). Single-component retrieval on V* Bench reveals that execution history (81.25%) and task descriptors (77.08%) carry the strongest retrieval signals, followed by visual observations (73.96%) and queries alone (65.62%). This empirical distribution confirms that experience retrieval relies on decoupled sub-state cues rather than monolithic trajectories, validating our compositional formulation (Appendix K).

4.4 Cross-Dataset Experience Transfer

To evaluate whether stateful experiences capture reusable tactical skills rather than dataset-specific memorization, we test under a matched cross-dataset transfer protocol on Qwen3-VL-32B-Instruct: for each benchmark, the experience bank is pooled from exploration trajectories of the other three sibling benchmarks, with zero in-domain interaction. As shown in Table 2, MuSEAgent achieves 59.64% average accuracy, consistently outperforming matched trajectory-level transfer across every benchmark (+7.04 points over Reflexion and +7.78 over Expel), with prominent margins on HR-Bench (+12.00) and V* Bench (+8.33). This comparison isolates experience granularity: trajectory-level memories drag along rigid, dataset-specific operator sequences that transfer poorly across distinct visual distributions, whereas atomic state-action transitions capture modular tool-use tactics that remain actionable in novel contexts. Remarkably, MuSEAgent using transferred experiences (59.64%) even surpasses trajectory-level baselines operating with in-domain exploration data (Reflexion at 57.33% and Expel at 56.96%, Table 1), confirming that state-level representations fundamentally enhance multimodal memory transfer.

Table 3: Ablations on Qwen3-VL-32B-Instruct over component disentanglement, experience source, and hindsight model $\mathcal{Q}_\phi$. Accuracy (%) on V* Bench and HR-Bench; best in **bold**.

Method	Exp. Level	Variant	V* Bench			HR-Bench
			Direct Attr.	Rel. Pos.	Micro Avg.	
(a) Component-wise Disentanglement						
w/ MuSEAgent	State	General RAG (single view, single round)	62.07	78.95	68.75	69.00
w/ MuSEAgent	State	+ Multi-view indexing only	67.24	84.21	73.96	72.00
w/ MuSEAgent	State	+ Iterative retrieval only (L=3)	77.59	81.58	79.17	68.00
w/ MuSEAgent	State	Trajectory-end + NL guidance	79.31	84.21	81.25	74.00
w/ MuSEAgent	State	Trajectory-end + Scalar value (Memento)	68.97	84.21	75.00	66.00
w/ MuSEAgent	State	Full Model (All transitions + NL + Deep-Wide)	**86.21**	84.21	**85.42**	**78.00**
(b) Experience Source						
w/ MuSEAgent	State	Correct only	62.07	**86.84**	71.88	72.00
w/ MuSEAgent	State	Incorrect only	75.86	76.32	76.04	76.00
w/ MuSEAgent	State	Correct + Incorrect	**86.21**	84.21	**85.42**	**78.00**
(c) Hindsight Reasoning Model $\mathcal{Q}_\phi$						
w/ MuSEAgent	State	Qwen3-VL-32B-Instruct (self)	75.86	81.58	78.12	75.00
w/ MuSEAgent	State	Qwen3-VL-235B-A22B-Instruct	74.14	81.58	77.08	76.00
w/ MuSEAgent	State	Qwen3.5-397B-A17B	77.59	**92.11**	83.33	73.00
w/ MuSEAgent	State	GPT-4o	**86.21**	84.21	**85.42**	**78.00**

4.5 Ablation Studies

Systematic component-wise disentanglement. Block (a) of Table 3 isolates each component on Qwen3-VL-32B-Instruct with Top-K=3: multi-view indexing improves over general RAG by +5.21 points (V* Bench) and +3.00 (HR-Bench); iterative Deep Search adds +10.42 on V* Bench; isolating transition granularity against trajectory-end transitions yields consistent gains of +4.17 and +4.00 points; and natural-language guidance outperforms scalar values (Memento) by +6.25 and +8.00 points under identical granularity.

Failure-derived experiences match or exceed success-derived ones. Block (b) of Table 3 compares trajectory sources. Success-only experiences yield 71.88% on V* Bench and 72.00% on HR-Bench, whereas failure-only experiences reach 76.04% and 76.00%, confirming that correcting reasoning errors provides stronger guidance than reinforcing successful paths. Combining both achieves peak accuracy (85.42% and 78.00%), confirming that diverse banks allow agents to learn optimal strategies while avoiding pitfalls.

Stateful experience gains hold under self-hindsight, ruling out a distillation confound. Block (c) varies $\mathcal{Q}_\phi$ to test reliance on external teachers. Under *self-hindsight*, where Qwen3-VL-32B-Instruct evaluates its own trajectories, MuSEAgent reaches 78.12% on V* Bench and 75.00% on HR-Bench, surpassing the strongest trajectory baselines (Reflexion at 75.00% and Expel at 66.00%) by +3.12 and +9.00 points. Scaling $\mathcal{Q}_\phi$ to Qwen3.5-397B-A17B and GPT-4o raises accuracy to 83.33% and 85.42% on V* Bench. Stronger teachers thus provide an upper bound rather than a precondition: stateful experience retrieval does not depend on proprietary distillation and can be deployed entirely using in-house inference stacks (Appendix F; see Appendix G and Appendix L for δ-sensitivity and failure analysis).

5 Conclusion

We present MuSEAgent, an experience-augmented multimodal reasoning agent that reframes experience reuse across three foundational dimensions: decomposing interaction trajectories into atomic transitions, supervising them with outcome-conditioned natural-language tactics via hindsight reasoning, and organizing heterogeneous states into a compositional multi-view space queried through Deep-and-Wide search. Across diverse benchmarks and architectures, stateful guidance consistently outperforms trajectory-level and scalar-value baselines, with gains holding under self-hindsight, exhibiting non-overlapping confidence intervals, and transferring robustly to unseen benchmarks without in-domain experience accumulation. Future work will explore online experience construction, enabling agents to continuously update memory during open-ended interaction.

AI USE STATEMENT

Generative AI tools and foundation models were used solely as experimental research artifacts under study, as well as for language polishing and grammatical proofreading of author-written prose. All experimental designs, analyses, and claims were formulated and verified by the authors.

ETHICS STATEMENT

This work investigates experience reuse for multimodal agents on publicly available research benchmarks. It involves no human subjects, sensitive personal data, or harmful environmental deployments. All datasets and model checkpoints are utilized in strict compliance with their respective research licenses.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, full algorithmic workflows are provided in Appendix B, baseline configurations in Appendix D, tool specifications in Appendix E, and detailed system prompts in Appendix N. All benchmarks and evaluation protocols are public research standards, and full code will be made publicly available upon publication.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14953–14962, 2023.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

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.

Mingxin Li, Yanzhao Zhang, Dingkun Long, Keqin Chen, Sibo Song, Shuai Bai, Zhibo Yang, Pengjun Xie, An Yang, Dayiheng Liu, et al. Qwen3-vl-embedding and qwen3-vl-reranker: A unified framework for state-of-the-art multimodal retrieval and ranking. *arXiv preprint arXiv:2601.04720*, 2026.

Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning. *arXiv preprint arXiv:2508.19201*, 2025.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

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.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11888–11898, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Wenbin Wang, Liang Ding, Minyan Zeng, Xiabin Zhou, Li Shen, Yong Luo, Wei Yu, and Dacheng Tao. Divide, conquer and combine: A training-free framework for high-resolution image perception in multimodal large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 7907–7915, 2025.

Lai Wei, Liangbo He, Jun Lan, Lingzhong Dong, Yutong Cai, Siyuan Li, Huijia Zhu, Weiqiang Wang, Linghe Kong, Yue Wang, Zhuosheng Zhang, and Weiran Huang. Zooming without zooming: Region-to-image distillation for fine-grained multimodal perception. *arXiv preprint arXiv:2602.11858*, 2026.

Penghao Wu and Saining Xie. V?: Guided visual search as a core mechanism in multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13084–13094, 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

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.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. Retroformer: Retrospective large language agents with policy gradient optimization. *arXiv preprint arXiv:2308.02151*, 2023.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025.

Yi-Fan Zhang, Huanyu Zhang, Haochen Tian, Chaoyou Fu, Shuangqing Zhang, Junfei Wu, Feng Li, Kun Wang, Qingsong Wen, Zhang Zhang, et al. Mme-realworld: Could your multimodal llm challenge high-resolution real-world scenarios that are difficult for humans? *arXiv preprint arXiv:2408.13257*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *ProQuest number: information to all users*, 2002.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.

A PROOF OF LEMMA 3.2

Proof. This result is a direct consequence of the label propagation framework (Zhu & Ghahramani, 2002; Zhou et al., 2003).

Existence and Uniqueness. Since $\mathbf{S} = \mathbf{D}^{-1}\mathbf{W}$ is a row-stochastic matrix, its spectral radius satisfies $\rho(\mathbf{S}) \leq 1$. For $\alpha \in [0, 1)$, the matrix $\alpha\mathbf{S}$ has spectral radius strictly less than 1, so $(\mathbf{I}-\alpha\mathbf{S})$ is invertible and the Neumann series $\sum_{n=0}^{\infty} \alpha^n \mathbf{S}^n$ converges to $(\mathbf{I}-\alpha\mathbf{S})^{-1}$. This establishes existence, uniqueness, and convergence of the equivalent iterates $\mathbf{q}^{(n+1)} = \alpha\mathbf{S}\mathbf{q}^{(n)} + (1-\alpha)\mathbf{q}^{(0)}$.

Optimality. Setting $\nabla_{\mathbf{q}}\mathcal{E} = \mathbf{0}$ in Eq. equation 3 yields

$$(\mathbf{D} - \mathbf{W})\mathbf{q} + \tfrac{1-\alpha}{\alpha}\mathbf{D}(\mathbf{q} - \mathbf{q}^{(0)}) = \mathbf{0},$$

which simplifies to $\mathbf{q} = \alpha\mathbf{S}\mathbf{q} + (1-\alpha)\mathbf{q}^{(0)}$, the fixed-point equation whose unique solution is $\mathbf{q}^*$. Since the graph Laplacian $\mathbf{D} - \mathbf{W}$ is positive semidefinite and the fidelity term $\frac{1-\alpha}{\alpha}\sum_i D_{ii}(q_i - q_i^{(0)})^2$ is strictly convex, $\mathcal{E}$ is strictly convex, and $\mathbf{q}^*$ is the unique global minimizer. □

B ALGORITHMS

We provide the full procedures of the two phases of MuSEAgent and clarify how each algorithmic step instantiates the design choices motivated in Section 3. The two phases are deliberately decoupled: Algorithm 1 runs once offline over an exploration set of trajectories to materialize a reusable, indexable experience bank, while Algorithm 2 runs once per reasoning step at inference time and only touches the bank through similarity lookups. This decoupling is what allows hindsight scoring (which is comparatively expensive) to be amortized across all downstream queries, and makes cross-dataset transfer (Section 4.4) a drop-in operation: only the bank pointer changes.

Offline experience abstraction (Algorithm 1). The procedure ingests historical trajectories $\mathcal{T}$ and emits a quality-filtered, multi-viewpoint-indexed experience bank $\mathcal{B}$. We decompose every τ into atomic transitions (s_t, a_t, s_{t+1}), so that the basic unit of memory is a localized decision context rather than a full episode. No trajectory-level prefiltering is applied: both successful and failed trajectories enter the pipeline, since failed ones often contain locally optimal sub-decisions and successful ones occasionally include detours. The hindsight reasoner $\mathcal{Q}_\phi$ resolves this at the transition level, producing a scalar quality q_t and a natural-language guidance g_t conditioned on the realized outcome o. Transitions with $q_t < \delta$ are discarded; transitions from failed trajectories are retained whenever $q_t \geq \delta$, since their guidance encodes *what to avoid* rather than *what to imitate*. Each surviving transition is wrapped into $e_t = (s_t, a_t, g_t)$ and indexed under all M viewpoints, because the inference-time viewpoint is state-adaptive and unknown offline; a missing index would force a re-embedding pass at deployment. The dominant offline cost is one call to $\mathcal{Q}_\phi$ per atomic transition plus $M \cdot |\mathcal{B}|$ embedding evaluations, both fully parallelizable and incurred once per exploration set.

Online Deep-and-Wide search (Algorithm 2). At inference step t, the agent consults $\mathcal{B}$ through up to L refinement rounds. In round j, the policy $\pi(\cdot \mid s_t)$ samples a viewpoint p_{i_j} emphasizing a different facet of s_t (e.g., perceptual intent under (u, v_t), tool-execution history under (d, H_t), or full task context (u, v_t, d, H_t)) and constructs $\mathbf{q}_t^{(i_j)}$. Viewpoint selection is performed by the base model via prompting, which keeps the framework compatible with any instruction-following base model and requires no additional training. Wide Search then retrieves the Top-K experiences from the matching viewpoint index by cosine similarity. The retrieved set $\mathcal{E}^{(j)}$ is unioned (with deduplication) into $\mathcal{E}_{\text{deep-wide}}$; union rather than replacement lets later rounds compensate for blind spots of earlier viewpoints. After L rounds, the retrieved guidances are concatenated into the working context and a_t is sampled jointly on s_t and the aggregated guidance. When $\mathcal{E}_{\text{deep-wide}}$ is empty, the agent falls back to its ReAct-style policy on s_t alone, so retrieval failure never blocks decision-making. With L and K small constants and embeddings pre-computed, the per-step cost is $\mathcal{O}(L \cdot |\mathcal{B}|)$ in the worst case, keeping deployment overhead bounded (Appendix F). Together, the two algorithms cleanly separate expensive hindsight-based bank construction from lightweight online retrieval, so improvements to the experience bank, the embedding model, or the base agent compose without architectural changes, and each component can be upgraded independently.

Algorithm 1 Stateful Experience Abstraction via Hindsight Reasoning

Require: Historical trajectories $\mathcal{T} = \{\tau_1, \tau_2, \dots\}$, quality threshold δ, smoothing parameter $\alpha \in [0, 1)$, candidate semantic viewpoints $\mathcal{P} = \{p_1, \dots, p_M\}$.
Ensure: Updated Experience Bank $\mathcal{B}$.

 # Phase 1: Hindsight scoring and multi-viewpoint indexing.
Initialize candidate pool $\mathcal{C} \leftarrow \emptyset$
for each trajectory $\tau \in \mathcal{T}$ **do**
  for each atomic transition $(s_t, a_t, s_{t+1}) \in \tau$ **do**
    Compute hindsight quality score and guidance summary: $(q_t, g_t) \leftarrow \mathcal{Q}_\phi(s_t, a_t, s_{t+1})$
    Construct candidate experience $c_t \leftarrow (s_t, a_t, g_t, q_t)$
    Build multi-viewpoint embeddings: $\mathbf{z}_t^{(i)} \leftarrow f_\theta(p_i(s_t)),\ \forall p_i \in \mathcal{P}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{c_t\}$
  end for
end for
 # Phase 2: Quality score smoothing.
Build affinity matrix: $W_{ij} \leftarrow \frac{1}{M} \sum_{k=1}^{M} \mathrm{sim}(\mathbf{z}_i^{(k)}, \mathbf{z}_j^{(k)})$
Row-normalize: $\mathbf{S} \leftarrow \mathbf{D}^{-1}\mathbf{W}, \quad \mathbf{D} = \mathrm{diag}(\mathbf{W1})$
Compute smoothed scores: $\mathbf{q}^* \leftarrow (1 - \alpha)\,(\mathbf{I} - \alpha\mathbf{S})^{-1}\,\mathbf{q}^{(0)}$
 # Phase 3: Quality-filtered bank construction.
$\mathcal{B} \leftarrow \{e_t = (s_t, a_t, g_t) \mid q_t^* \geq \delta,\ c_t \in \mathcal{C}\}$
return $\mathcal{B}$

Algorithm 2 Experience Exploitation via Deep-and-Wide Search

Require: Experience Bank $\mathcal{B}$, current state s_t, retrieval breadth K, max refinement rounds L, candidate semantic viewpoints $\mathcal{P} = \{p_1, \dots, p_M\}$.
Ensure: Execution action a_t.

Initialize unified guidance set $\mathcal{E}_{\text{deep-wide}} \leftarrow \emptyset$
 # Deep Search: Iterative Semantic Viewpoint Refinement
for round $j = 1$ to L **do**
  Sample semantic viewpoint and formulate query embedding: $p_{i_j} \sim \pi(\cdot \mid s_t), \quad \mathbf{q}_t^{(i_j)} \leftarrow f_\theta\big(p_{i_j}(s_t)\big)$
  # Wide Search: Top-K Retrieval under Current Viewpoint
  Retrieve top-K experiences:

$$\mathcal{E}^{(j)} \leftarrow \text{Top-K}_{e_m \in \mathcal{B}}\left(\mathrm{sim}\left(\mathbf{q}_t^{(i_j)}, \mathbf{z}_m^{(i_j)}\right)\right)$$

  Aggregate retrieved experiences: $\mathcal{E}_{\text{deep-wide}} \leftarrow \mathcal{E}_{\text{deep-wide}} \cup \mathcal{E}^{(j)}$
end for
 # Action Generation
Generate execution action: $a_t \sim \pi(\cdot \mid s_t, \mathcal{E}_{\text{deep-wide}})$
return a_t

C Datasets and Metrics

We summarize the benchmarks and corresponding metrics used to evaluate the fine-grained perception and high-resolution reasoning capabilities of multimodal models below. Detailed question counts across exploration and evaluation splits are presented in Table 4.

V* Bench (Wu & Xie, 2024). This Visual Question Answering (VQA) benchmark is designed to evaluate the detailed visual grounding capabilities of multimodal models on high-resolution images. It consists of 191 carefully curated samples divided into two fine-grained sub-tasks: attribute recognition and spatial relationship reasoning. All questions are presented in a multiple-choice format. The benchmark tests whether models can accurately perceive small, critical visual details that standard vision encoders typically overlook. We use the overall accuracy across these sub-tasks as the final score.

Table 4: Statistics of Benchmark Datasets across exploration and evaluation splits.

Split	V* Bench			MME-RealWorld-Lite			Zoom-Bench	HR-Bench
	Direct Attr.	Rel. Pos.	Total	Percept.	Reas.	Total		
Exploration	57	38	95	584	375	959	310	100
Evaluation	58	38	96	585	375	960	311	100

MME-RealWorld-Lite (Zhang et al., 2024). This is an accelerated evaluation subset of the large-scale MME-RealWorld benchmark, which tests models on high-resolution images across challenging, practical real-world scenarios. By efficiently subsampling up to 50 instances across 43 tasks, the Lite version maintains diverse scenario coverage with reduced computational cost. For our evaluation, we select the multiple-choice questions from this dataset, totaling 1,919 items. We use the average task accuracy as the final score.

Zoom-Bench (Wei et al., 2026). Consisting of 845 instances, this VQA benchmark is designed to assess fine-grained multimodal perception. It specifically focuses on challenging scenarios where the decisive visual evidence is extremely small and easily overshadowed by the global context, evaluating a model's ability to accurately capture minute details. In our evaluation, we select the multiple-choice subset, which contains 621 questions. We use the overall VQA accuracy as the final score.

HR-Bench (Wang et al., 2025). This benchmark is deliberately constructed to measure multimodal models' perception and reasoning capabilities on true high-resolution images. It focuses on fine-grained single-instance and cross-instance perception tasks, systematically evaluating how well models extract and reason over intricate visual details typically lost during standard image down-sampling. In our evaluation, we select 200 multiple-choice questions based on 8K-resolution images from HR-Bench, and use the overall accuracy on these high-resolution questions as the final score.

D Baseline Methods

The baseline methods we compare against in our experiments are summarized as follows:

Vanilla CoT (Kojima et al., 2022). Chain-of-Thought prompting directs LLMs/VLMs to generate a sequence of intermediate reasoning steps before producing a final answer. This method relies solely on the model's pre-trained parametric knowledge to decompose complex problems into sequential logical deductions. Operating without external tool invocations or environmental feedback, Vanilla CoT serves as a baseline to evaluate the intrinsic reasoning capacity and standalone problem-solving performance of language models.

ReAct (Yao et al., 2022). This framework integrates reasoning and acting within LLMs/VLMs. Instead of relying solely on internal knowledge, ReAct enables models to generate reasoning traces and task-specific actions in an interleaved manner. This process allows the agent to formulate and adjust plans while interacting with external environments, such as knowledge bases or APIs, to retrieve real-time observations. By grounding its reasoning in external information, ReAct provides a mechanism to reduce factual errors and maintain interpretability during multi-step decision-making tasks.

Reflexion (Shinn et al., 2023). Reflexion equips agents with a mechanism for improving their behavior through verbal self-reflection. After completing a task attempt, an evaluator analyzes the resulting trajectory and produces natural language reflections based on environmental feedback. These reflections are stored in an episodic memory buffer and provided as additional context in subsequent attempts, enabling the agent to revise its reasoning strategy and avoid repeating previous mistakes. In our implementation, reflective experiences are extracted only from failed trajectories.

Expel (Zhao et al., 2024). Expel is a method that enables agents to distill reusable knowledge from their interactions with the environment. By analyzing task trajectories, the agent extracts generalized rules or strategies that can guide future decision making. Unlike approaches that focus primarily on failure analysis, Expel leverages information from both successful and unsuccessful experiences to identify patterns that improve planning and reasoning. The extracted knowledge is stored in an

experience pool and incorporated into future task executions. In our implementation, experiences are extracted from both successful and failed trajectories.

E TOOL DESCRIPTIONS

We equip our model with a diverse set of tools to interact with visual inputs, perform calculations, and access external information. These tools are categorized into four functional modules below.

E.1 IMAGE MANIPULATION TOOLS

Zoom In. This tool scales a specific region of an image based on a provided 2D bounding box and a numerical zoom factor. It outputs the magnified image segment to facilitate the inspection of minute details.

```
class ZoomInTool(BasicTool):
    name = "zoom_in"
    description_en = "Zoom in on a region by cropping and upscaling. Use
    localize_objects first if bbox is unknown."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "bbox_2d": {
                "type": "array",
                "items": {"type": "number"},
                "minItems": 4,
                "maxItems": 4,
                "description": "Bounding box [left, top, right, bottom],
    values between 0 and 1"
            },
            "zoom_factor": {
                "type": "number",
                "description": "Zoom factor (must be > 1)",
                "exclusiveMinimum": 1.0
            }
        },
        "required": ["image", "bbox_2d", "zoom_factor"]
    }
    example = '{"image": "img_0", "bbox_2d": [0.25, 0.25, 0.75, 0.75], "
    zoom_factor": 2.0}'

    def call(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)

        image_path = params_dict["image"]
        bbox_2d = params_dict["bbox_2d"]
        zoom_factor = params_dict["zoom_factor"]

        if not all(0 <= x <= 1.0 for x in bbox_2d):
            return {"error": "bbox_2d values must be between 0 and 1"}
        if bbox_2d[0] >= bbox_2d[2] or bbox_2d[1] >= bbox_2d[3]:
            return {"error": "invalid bbox_2d: left >= right or top >=
    bottom"}
        if zoom_factor <= 1:
            return {"error": "zoom_factor must be > 1"}

        try:
            image = image_processing(image_path)
        except Exception as e:
```

```
            return {"error": f"failed to load image: {e}"}

        W, H = image.size
        crop_box = (int(bbox_2d[0] * W), int(bbox_2d[1] * H), int(bbox_2d
    [2] * W), int(bbox_2d[3] * H))
        cropped = image.crop(crop_box)

        new_size = (int(cropped.width * zoom_factor), int(cropped.height
    * zoom_factor))
        zoomed = cropped.resize(new_size, Image.LANCZOS)

        return {"output_image": zoomed}

    def generate_description(self, properties, observation):
        img = properties.get("image", "image")
        bbox_2d = properties.get("bbox_2d", [])
        factor = properties.get("factor", 2)
        return f"Zoomed in {img} at bbox_2d {bbox_2d} by {factor}x"
```

Crop. This tool extracts a sub-region from the original image using specified 2D bounding box coordinates. It removes peripheral visual context to isolate and return the target area as a new image.

```
class CropTool(BasicTool):
    name = "crop"
    description_en = "Crop a region from an image. Use localize_objects
    first if bbox is unknown."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "bbox_2d": {
                "type": "array",
                "items": {"type": "number"},
                "minItems": 4,
                "maxItems": 4,
                "description": "Bounding box as [left, top, right, bottom
    ], values between 0 and 1"
            }
        },
        "required": ["image", "bbox_2d"]
    }
    example = '{"image": "img_0", "bbox_2d": [0.1, 0.2, 0.5, 0.6]}'

    def call(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)

        image_path = params_dict["image"]
        bbox_2d = params_dict["bbox_2d"]

        if not all(0 <= x <= 1.0 for x in bbox_2d):
            return {"error": "bbox_2d values must be between 0 and 1"}
        if bbox_2d[0] >= bbox_2d[2] or bbox_2d[1] >= bbox_2d[3]:
            return {"error": "invalid bbox_2d: left >= right or top >=
    bottom"}

        try:
            image = image_processing(image_path)
        except Exception as e:
            return {"error": f"failed to load image: {e}"}
```

```
        W, H = image.size
        crop_box = (int(bbox_2d[0] * W), int(bbox_2d[1] * H), int(bbox_2d
    [2] * W), int(bbox_2d[3] * H))
        cropped = image.crop(crop_box)

        return {"output_image": cropped}

    def generate_description(self, properties, observation):
        img = properties.get("image", "image")
        bbox_2d = properties.get("bbox_2d", [])
        return f"Cropped {img} at bbox_2d {bbox_2d}"
```

Visualize Regions. This tool overlays annotated bounding boxes and their corresponding text labels directly onto an input image. It generates a new visual output that explicitly highlights the specified regions of interest for structural grounding.

```
class VisualizeRegionsOnImageTool(BasicTool):
    name = "visualize_regions"
    description_en = "Draw bounding boxes and labels on an image. Each
    region is defined by normalized coordinates [x1, y1, x2, y2] where
    values are between 0 and 1. Use localize_objects first if bbox is
    unknown."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "regions": {
                "type": "array",
                "description": "List of regions to label, each with '
    bbox_2d' and optional 'label'",
                "items": {
                    "type": "object",
                    "properties": {
                        "bbox_2d": {
                            "type": "array",
                            "description": "Normalized bounding box [x1,
    y1, x2, y2] where values are between 0 and 1",
                            "items": {"type": "number"},
                            "minItems": 4,
                            "maxItems": 4
                        },
                        "label": {
                            "type": "string",
                            "description": "Optional label text for the
    region"
                        }
                    },
                    "required": ["bbox_2d"]
                }
            },
            "color": {
                "type": "string",
                "description": "Color of the bounding boxes (default: '
    yellow')"
            },
            "width": {
                "type": "integer",
                "description": "Width of the bounding box lines (default:
     4)"
            }
        },
```

```
        "required": ["image", "regions"]
    }
    example = '{"image": "img_0", "regions": [{"bbox_2d": [0.1, 0.1, 0.5,
     0.5], "label": "Object A"}, {"bbox_2d": [0.6, 0.6, 0.9, 0.9], "label
    ": "Object B"}]}'

    def call(self, params: Union[str, Dict]) -> str:
        if not PIL_AVAILABLE:
            return {"error": "PIL (Pillow) is not installed. Please
    install it with: pip install Pillow"}

        params_dict = self.parse_params(params)

        image_path = params_dict["image"]
        regions = params_dict["regions"]
        color = params_dict.get("color", "yellow")
        width = params_dict.get("width", 4)

        try:
            image = image_processing(image_path)
            W, H = image.size

            img_labeled = image.copy()
            draw = ImageDraw.Draw(img_labeled)

            try:
                font = ImageFont.truetype('/usr/share/fonts/truetype/
    dejavu/DejaVuSansMono-Bold.ttf', 16)
            except (OSError, IOError):
                try:
                    font = ImageFont.truetype('/System/Library/Fonts/
    Helvetica.ttc', 16)
                except (OSError, IOError):
                    font = ImageFont.load_default()

            text_color = 'black'

            for obj in regions:
                bbox_2d = obj['bbox_2d']
                bbox_pixels = (bbox_2d[0] * W, bbox_2d[1] * H, bbox_2d[2]
     * W, bbox_2d[3] * H)

                draw.rectangle(bbox_pixels, outline=color, width=width)

                x1, y1, x2, y2 = bbox_pixels
                label = obj.get('label', '')

                if label:
                    try:
                        bbox_text = draw.textbbox((0, 0), label, font=
    font)
                        w = bbox_text[2] - bbox_text[0]
                        h = bbox_text[3] - bbox_text[1]
                    except AttributeError:
                        w, h = font.getsize(label)

                    if x1 + w > W or y2 + h > H:
                        draw.rectangle((x1, y2 - h, x1 + w, y2), fill=
    color)
                        draw.text((x1, y2 - h), label, fill=text_color,
    font=font)
                    else:
                        draw.rectangle((x1, y2, x1 + w, y2 + h), fill=
    color)
```

```
                        draw.text((x1, y2), label, fill=text_color, font=
    font)

            return {"output_image": img_labeled}

        except FileNotFoundError:
            return {"error": f"Image file not found at {image_path}"}
        except Exception as e:
            return {"error": str(e)}

    def generate_description(self, properties, observation):
        img = properties.get("image", "image")
        regions = properties.get("regions", [])
        num_regions = len(regions) if isinstance(regions, list) else 0
        return f"Visualized {num_regions} regions on {img}"
```

E.2 Visual Perception Tools

OCR. Optical Character Recognition scans an input image to detect and extract visible text elements. It returns the recognized textual content to assist in reading and parsing tasks.

```
OCR_PROMPT = "OCR:"
MAX_PIXELS_OCR = 1280 * 28 * 28

class OCRTool(ModelBasedTool):
    name = "ocr"
    model_id = "ocr"

    description_en = "Extract text from an image. Returns empty if no
    text found."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            }
        },
        "required": ["image"]
    }
    example = '{"image": "img_0"}'

    def load_model(self, device: str) -> None:
        self.model = AutoModelForImageTextToText.from_pretrained(
            PADDLE_OCR_VL_MODEL_PATH, torch_dtype=torch.bfloat16
        ).to(device).eval()
        self.processor = AutoProcessor.from_pretrained(
    PADDLE_OCR_VL_MODEL_PATH)
        self.device = device
        self.is_loaded = True

    def _call_impl(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)
        image_path = params_dict["image"]
        tmp_file_to_remove = None

        try:
            full_path = image_path if os.path.exists(image_path) else
    get_full_path_data(image_path)
            if full_path and os.path.exists(full_path):
                image_file = full_path
            else:
                image = image_processing(image_path)
```

```
                if isinstance(image, Image.Image):
                    tmp = tempfile.NamedTemporaryFile(suffix=".png",
delete=False)
                    tmp_file_to_remove = tmp.name
                    try:
                        image.save(tmp.name)
                        image_file = tmp.name
                    finally:
                        tmp.close()
                elif isinstance(image, str):
                    resolved = image if os.path.exists(image) else
get_full_path_data(image)
                    if resolved is None:
                        raise FileNotFoundError(f"Image not found: {
image_path}")
                    image_file = resolved
                else:
                    raise ValueError(f"Unexpected image type: {type(image
)}")

            image = Image.open(image_file).convert("RGB")

            messages = [
                {
                    "role": "user",
                    "content": [
                        {"type": "image", "image": image},
                        {"type": "text", "text": OCR_PROMPT},
                    ]
                }
            ]
            inputs = self.processor.apply_chat_template(
                messages,
                add_generation_prompt=True,
                tokenize=True,
                return_dict=True,
                return_tensors="pt",
                images_kwargs={
                    "size": {
                        "shortest_edge": self.processor.image_processor.
min_pixels,
                        "longest_edge": MAX_PIXELS_OCR,
                    }
                },
            )

            if "pixel_values" in inputs and not isinstance(inputs["
pixel_values"], torch.Tensor):
                inputs["pixel_values"] = torch.from_numpy(inputs["
pixel_values"]).to(self.device)
            if "image_grid_thw" in inputs and not isinstance(inputs["
image_grid_thw"], torch.Tensor):
                inputs["image_grid_thw"] = torch.from_numpy(inputs["
image_grid_thw"]).to(self.device)

            inputs = {
                k: v.to(self.device) if isinstance(v, torch.Tensor) else
v
                for k, v in inputs.items()
            }

            outputs = self.model.generate(**inputs, max_new_tokens=512)
            extracted_text = self.processor.decode(
                outputs[0][inputs["input_ids"].shape[-1] : -1]
            ).strip()
```

```
            return {"extracted text": extracted_text}
        except FileNotFoundError as e:
            return {"error": f"Image not found: {str(e)}"}
        except Exception as e:
            return {"error": f"OCR error: {str(e)}\n{traceback.format_exc
    ()}"}
        finally:
            if tmp_file_to_remove and os.path.exists(tmp_file_to_remove):
                try:
                    os.unlink(tmp_file_to_remove)
                except OSError:
                    pass
```

Localize Objects. This tool identifies the spatial locations of user-specified object categories within an image. It processes the semantic queries and outputs the 2D bounding boxes corresponding to the detected entities.

```
class LocalizeObjectsTool(ModelBasedTool):
    name = "localize_objects"
    model_id = "sam3"

    description_en = "Localize objects and return their bounding boxes.
    Use this to get bbox for other region-based tools. Note: Cannot
    detect text labels or annotation markers (e.g., 'A', 'B', 'point 1')
    drawn on images."

    parameters = {
        "type": "object",
        "properties": {
            "image": {"type": "string", "description": "Image ID (e.g., '
    img_0')"},
            "objects": {"type": "array", "items": {"type": "string"}, "
    description": "A list of object names to localize. e.g. ['dog', 'cat
    ', 'person']."}
        },
        "required": ["image", "objects"]
    }
    example = '{"image": "img_0", "objects": ["dog", "cat"]}'

    def load_model(self, device: str) -> None:
        self.processor = Sam3Processor.from_pretrained(SAM3_MODEL_PATH)
        self.model = Sam3Model.from_pretrained(SAM3_MODEL_PATH).to(device
    )
        self.device = device
        self.is_loaded = True

    def _call_impl(self, params: Union[str, Dict]) -> str:
        params_dict = self.parse_params(params)
        image_path = params_dict["image"]
        objects = params_dict["objects"]

        if not isinstance(objects, list) or len(objects) == 0:
            return {
                "error": "objects must be a non-empty list of strings"
            }

        try:
            image = image_processing(image_path)
            W, H = image.size

            regions = []
            obj_cnt = {}

            for obj_name in objects:
```

```
            inputs = self.processor(images=image, text=obj_name,
return_tensors="pt").to(self.device)

            with torch.no_grad():
                outputs = self.model(**inputs)

            results = self.processor.
post_process_instance_segmentation(
                outputs,
                threshold=0.5,
                mask_threshold=0.5,
                target_sizes=inputs.get("original_sizes").tolist()
            )[0]

            boxes = results["boxes"]
            scores = results["scores"]

            for box, score in zip(boxes, scores):
                score_val = score.item()
                if score_val < 0.50:
                    continue

                box_list = box.tolist()
                bbox = [
                    round(float(box_list[0]) / W, 4),
                    round(float(box_list[1]) / H, 4),
                    round(float(box_list[2]) / W, 4),
                    round(float(box_list[3]) / H, 4)
                ]

                obj_cnt[obj_name] = obj_cnt.get(obj_name, 0) + 1
                label_out = f"{obj_name}-{obj_cnt[obj_name]}" if
obj_cnt[obj_name] > 1 else obj_name
                regions.append({
                    "label": label_out,
                    "bbox_2d": bbox,
                    "score": round(score_val, 4)
                })

        visualize_tool = VisualizeRegionsOnImageTool()
        visualize_params = {
            "image": image_path,
            "regions": [{"bbox_2d": r["bbox_2d"], "label": r["label"
]} for r in regions]
        }
        output_image_result = visualize_tool.call(visualize_params)
        output_image = output_image_result.get("output_image")

        return {
            "output_image": output_image,
            "regions": regions
        }

    except FileNotFoundError as e:
        return {"error": f"Image file not found: {str(e)}"}
    except Exception as e:
        return {"error": f"Error localizing objects: {str(e)}"}

def generate_description(self, properties, observation):
    img = properties.get("image", "image")
    objects = properties.get("objects", [])
    if isinstance(objects, list):
        objects_str = ", ".join(objects) if objects else "objects"
    else:
        objects_str = str(objects)
```

```
        return f"Localized {objects_str} in {img}"
```

Estimate Region Depth. This tool calculates the spatial depth values within a designated bounding box. It aggregates the pixel-level depth maps and outputs statistical results, such as the mean depth of the specified area.

```
class EstimateRegionDepthTool(ModelBasedTool):
    name = "estimate_region_depth"
    model_id = "depth_anything"

    description_en = "Estimate depth of a region. Smaller value = closer.
     Use localize_objects first if bbox is unknown."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "bbox_2d": {
                "type": "array",
                "items": {"type": "number"},
                "minItems": 4,
                "maxItems": 4,
                "description": "Bounding box [x1, y1, x2, y2] in
    normalized coordinates (0-1)"
            },
            "mode": {
                "type": "string",
                "enum": ["mean", "min", "max"],
                "description": "Mode to compute depth: mean (average),
    min (closest), or max (farthest)",
                "default": "mean"
            }
        },
        "required": ["image", "bbox_2d"]
    }
    example = '{"image": "img_0", "bbox_2d": [0.1, 0.2, 0.5, 0.6], "mode
    ": "mean"}'

    def load_model(self, device: str) -> None:
        model_id = DEPTH_ANYTHING_PATH
        self.image_processor = AutoImageProcessor.from_pretrained(
    model_id)
        self.model = AutoModelForDepthEstimation.from_pretrained(model_id
    )
        self.model.to(device)
        self.model.eval()
        self.device = device
        self.is_loaded = True

    def _call_impl(self, params: Union[str, Dict]) -> str:
        params_dict = self.parse_params(params)
        image_path = params_dict["image"]
        bbox_2d = params_dict["bbox_2d"]
        mode = params_dict.get("mode", "mean")

        if not all(0 <= x <= 1.0 for x in bbox_2d):
            return {"error": "bbox_2d values must be between 0 and 1"}
        if bbox_2d[0] >= bbox_2d[2] or bbox_2d[1] >= bbox_2d[3]:
            return {"error": "invalid bbox_2d: left >= right or top >=
    bottom"}

        try:
```

```
            image = image_processing(image_path)
            W, H = image.size

            x1 = int(bbox_2d[0] * W)
            y1 = int(bbox_2d[1] * H)
            x2 = int(bbox_2d[2] * W)
            y2 = int(bbox_2d[3] * H)

            inputs = self.image_processor(images=image, return_tensors="
    pt")
            inputs = {k: v.to(self.device) for k, v in inputs.items()}

            with torch.no_grad():
                outputs = self.model(**inputs)

            processed_outputs = self.image_processor.
    post_process_depth_estimation(
                outputs,
                target_sizes=[(H, W)],
            )
            depth_map = processed_outputs[0]["predicted_depth"].squeeze()
    .detach().cpu().numpy()
            depth_map = depth_map.max() - depth_map

            region_depth = depth_map[y1:y2, x1:x2]

            if mode == "mean":
                depth_value = float(np.mean(region_depth))
            elif mode == "min":
                depth_value = float(np.min(region_depth))
            elif mode == "max":
                depth_value = float(np.max(region_depth))
            else:
                depth_value = float(np.mean(region_depth))

            return {"estimated depth": round(depth_value, 4)}

        except FileNotFoundError as e:
            return {"error": f"Image file not found: {str(e)}"}
        except Exception as e:
            return {"error": f"Error estimating depth: {str(e)}"}
```

Estimate Object Depth. This tool predicts the depth value of a specific semantic object within the visual scene. It combines object recognition with monocular depth estimation to locate the target and determine its distance.

```
class EstimateObjectDepthTool(BasicTool):
    name = "estimate_object_depth"
    description_en = "Estimate depth of an object by text description.
    Smaller value = closer."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "object": {
                "type": "string",
                "description": "A short description of the object to get
    the depth from"
            },
            "mode": {
                "type": "string",
```

```
                "enum": ["mean", "min", "max"],
                "description": "Mode to compute depth: mean (average),
    min (closest), or max (farthest)",
                "default": "mean"
            }
        },
        "required": ["image", "object"]
    }
    example = '{"image": "img_0", "object": "a black cat", "mode": "mean
    "}'

    def call(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)
        image_path = params_dict["image"]
        object_desc = params_dict["object"]
        mode = params_dict.get("mode", "mean")

        try:
            localize_tool_class = TOOL_REGISTRY.get("localize_objects")
            if localize_tool_class is None:
                return {
                    "error": "localize_objects tool is not available"
                }

            localize_tool = localize_tool_class()
            localize_params = {
                "image": image_path,
                "objects": [object_desc]
            }
            localize_result = localize_tool.call(localize_params)

            if isinstance(localize_result, str):
                localize_data = json.loads(localize_result)
            else:
                localize_data = localize_result

            if "error" in localize_data or len(localize_data.get("regions
    ", [])) == 0:
                return {
                    "error": "Object not found"
                }

            regions = localize_data["regions"]
            best_match_idx = np.argmax([region.get("score", 0) for region
     in regions])
            bbox_2d = regions[best_match_idx]["bbox_2d"]

            estimate_depth_tool_class = TOOL_REGISTRY.get("
    estimate_region_depth")
            if estimate_depth_tool_class is None:
                return {"error": "estimate_region_depth tool is not
    available"}

            estimate_depth_tool = estimate_depth_tool_class()
            depth_params = {
                "image": image_path,
                "bbox_2d": bbox_2d,
                "mode": mode
            }
            return estimate_depth_tool.call(depth_params)

        except Exception as e:
            return {"error": f"Error estimating object depth: {str(e)}"}
```

E.3 Semantic Similarity Tools

Image-to-Image Similarity. This tool computes the visual embedding distance between a source image and a set of reference images. It outputs similarity scores to evaluate the visual resemblance among different image inputs.

```
class GetImageToImagesSimilarityTool(ModelBasedTool):
    name = "get_image2images_similarity"
    model_id = "clip"

    description_en = "Compute similarity between one image and multiple
    images."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "other_images": {
                "type": "array",
                "items": {"type": "string"},
                "description": "The other images to compare to the
    reference image"
            }
        },
        "required": ["image", "other_images"]
    }
    example = '{"image": "img_0", "other_images": ["img_1", "img_2"]}'

    def load_model(self, device: str) -> None:
        self.model, _, self.preprocess = open_clip.
    create_model_and_transforms(CLIP_VERSION, pretrained=CLIP_PRETRAINED)
        self.model.eval()
        self.model = self.model.to(device)
        self.tokenizer = open_clip.get_tokenizer(CLIP_VERSION)
        self.device = device
        self.is_loaded = True

    def _call_impl(self, params: Union[str, Dict]) -> str:
        params_dict = self.parse_params(params)
        image_path = params_dict["image"]
        other_images = params_dict["other_images"]

        if not isinstance(other_images, list) or len(other_images) == 0:
            return {
                "error": "other_images must be a non-empty list of image
    paths"
            }

        try:
            reference_image = image_processing(image_path)
            reference_image_tensor = self.preprocess(reference_image).
    unsqueeze(0).to(self.device)

            other_images_processed = []
            for img_path in other_images:
                img = image_processing(img_path)
                other_images_processed.append(img)

            other_images_tensor = torch.stack([
                self.preprocess(img).to(self.device) for img in
    other_images_processed
            ])
```

```
            with torch.no_grad(), torch.cuda.amp.autocast():
                reference_features = self.model.encode_image(
    reference_image_tensor)
                other_features = self.model.encode_image(
    other_images_tensor)

                reference_features /= reference_features.norm(dim=-1,
    keepdim=True)
                other_features /= other_features.norm(dim=-1, keepdim=
    True)

                similarity_scores = reference_features @ other_features.T

            sim_scores = [round(score.item(), 2) for score in
    similarity_scores[0]]

            best_image_index = torch.argmax(similarity_scores, dim=1).
    item()

            return {
                "similarity scores": sim_scores,
                "best match": other_images[best_image_index]
            }

        except FileNotFoundError as e:
            return {"error": f"Image file not found: {str(e)}"}
        except Exception as e:
            return {"error": f"Error computing image similarity: {str(e)}
    "}
```

Image-to-Text Similarity. This tool evaluates the semantic alignment between a given image and multiple textual descriptions. It returns scores indicating how accurately each text matches the visual content.

```
class GetImageToTextsSimilarityTool(ModelBasedTool):
    name = "get_image2texts_similarity"
    model_id = "clip"

    description_en = "Compute similarity between one image and multiple
    texts (image  texts)."

    parameters = {
        "type": "object",
        "properties": {
            "image": {
                "type": "string",
                "description": "Image ID (e.g., 'img_0')"
            },
            "texts": {
                "type": "array",
                "items": {"type": "string"},
                "description": "A list of texts to compare to the
    reference image"
            }
        },
        "required": ["image", "texts"]
    }
    example = '{"image": "img_0", "texts": ["a cat", "a dog"]}'

    def load_model(self, device: str) -> None:
        self.model, _, self.preprocess = open_clip.
    create_model_and_transforms(CLIP_VERSION, pretrained=CLIP_PRETRAINED)
        self.model.eval()
        self.model = self.model.to(device)
```

```
        self.tokenizer = open_clip.get_tokenizer(CLIP_VERSION)
        self.device = device
        self.is_loaded = True

    def _call_impl(self, params: Union[str, Dict]) -> str:
        params_dict = self.parse_params(params)
        image_path = params_dict["image"]
        texts = params_dict["texts"]

        if not isinstance(texts, list) or len(texts) == 0:
            return {
                "error": "texts must be a non-empty list of strings"
            }

        try:
            reference_image = image_processing(image_path)
            reference_image_tensor = self.preprocess(reference_image).
    unsqueeze(0).to(self.device)

            text_tokens = self.tokenizer(texts).to(self.device)

            with torch.no_grad(), torch.cuda.amp.autocast():
                image_features = self.model.encode_image(
    reference_image_tensor)
                text_features = self.model.encode_text(text_tokens)

                image_features /= image_features.norm(dim=-1, keepdim=
    True)
                text_features /= text_features.norm(dim=-1, keepdim=True)

                similarity_scores = image_features @ text_features.T

            sim_scores = [round(score.item(), 2) for score in
    similarity_scores[0]]

            best_text_index = torch.argmax(similarity_scores, dim=1).item
    ()

            return {
                "similarity scores": sim_scores,
                "best match": texts[best_text_index]
            }

        except FileNotFoundError as e:
            return {"error": f"Image file not found: {str(e)}"}
        except Exception as e:
            return {"error": f"Error computing image-to-text similarity:
    {str(e)}"}
```

Text-to-Image Similarity. This tool measures how well a set of images corresponds to a specific text query. It calculates alignment scores for each image, functioning as a targeted text-based image retrieval metric.

```
class GetTextToImagesSimilarityTool(ModelBasedTool):
    name = "get_text2images_similarity"
    model_id = "clip"

    description_en = "Compute similarity between one text and multiple
    images (text  images)."

    parameters = {
        "type": "object",
        "properties": {
            "text": {
                "type": "string",
```

```
                "description": "The reference text"
            },
            "images": {
                "type": "array",
                "items": {"type": "string"},
                "description": "A list of images to compare to the
reference text"
            }
        },
        "required": ["text", "images"]
    }
    example = '{"text": "a black and white cat", "images": ["img_0", "
img_1"]}'

    def load_model(self, device: str) -> None:
        self.model, _, self.preprocess = open_clip.
create_model_and_transforms(CLIP_VERSION, pretrained=CLIP_PRETRAINED)
        self.model.eval()
        self.model = self.model.to(device)
        self.tokenizer = open_clip.get_tokenizer(CLIP_VERSION)
        self.device = device
        self.is_loaded = True

    def _call_impl(self, params: Union[str, Dict]) -> str:
        params_dict = self.parse_params(params)
        text = params_dict["text"]
        images = params_dict["images"]

        if not isinstance(images, list) or len(images) == 0:
            return {
                "error": "images must be a non-empty list of image paths"
            }

        try:
            text_tokens = self.tokenizer([text]).to(self.device)

            images_processed = []
            for img_path in images:
                img = image_processing(img_path)
                images_processed.append(img)

            images_tensor = torch.stack([
                self.preprocess(img).to(self.device) for img in
images_processed
            ])

            with torch.no_grad(), torch.cuda.amp.autocast():
                text_features = self.model.encode_text(text_tokens)
                image_features = self.model.encode_image(images_tensor)

                text_features /= text_features.norm(dim=-1, keepdim=True)
                image_features /= image_features.norm(dim=-1, keepdim=
True)

                similarity_scores = text_features @ image_features.T

            sim_scores = [round(score.item(), 2) for score in
similarity_scores[0]]

            best_image_index = torch.argmax(similarity_scores, dim=1).
item()

            return {
                "similarity scores": sim_scores,
                "best match": images[best_image_index]
```

```
            }

        except FileNotFoundError as e:
            return {"error": f"Image file not found: {str(e)}"}
        except Exception as e:
            return {"error": f"Error computing text-to-image similarity:
    {str(e)}"}
```

E.4 Reasoning and External Tools

Calculator. This tool evaluates standard arithmetic expressions provided as strings. It processes mathematical operators and numbers to return the exact numerical calculation result.

```
class CalculatorTool(BasicTool):
    name = "calculator"
    description_en = "Calculate mathematical expressions. Supports basic
    arithmetic (+, -, *, /), exponentiation (**), and parentheses."

    parameters = {
        "type": "object",
        "properties": {
            "expression": {
                "type": "string",
                "description": "The mathematical expression to calculate"
            }
        },
        "required": ["expression"]
    }
    example = '{"expression": "123 * 456 + 789"}'

    def call(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)
        expression = params_dict["expression"]

        if not self._is_safe_expression(expression):
            return {"error": "Expression contains invalid or unsafe
    characters"}

        try:
            result = eval(expression, {"__builtins__": {}}, {})
            return {"calculation result": result}
        except ZeroDivisionError:
            return {"error": "Division by zero"}
        except SyntaxError:
            return {"error": "Invalid syntax in expression"}
        except Exception as e:
            return {"error": str(e)}

    def _is_safe_expression(self, expr: str) -> bool:
        safe_pattern = re.compile(r'^[\d\s\+\-\*\/\(\)\.\*\*]+$')
        return bool(safe_pattern.match(expr))
```

Solve Math Equation. This tool processes and solves algebraic equations or math-related word problems presented in natural language formats. It executes the mathematical logic and outputs the analytical or numerical solution.

```
class SolveMathEquationTool(BasicTool):
    name = "solve_math_equation"
    description_en = (
        "Solve mathematical equations and problems using WolframAlpha. "
        "Supports algebra, calculus, and symbolic/numeric math queries."
    )

    parameters = {
```

```
        "type": "object",
        "properties": {
            "query": {
                "type": "string",
                "description": "A mathematical question or equation to
    solve",
            }
        },
        "required": ["query"],
    }
    example = '{"query": "x^2 + 2x + 1 = 0, what is x?"}'

    _key_index = 0
    _key_lock = threading.Lock()

    @staticmethod
    def _load_wolfram_keys_from_env() -> List[str]:
        keys = []

        api_keys_str = os.getenv("WOLFRAM_ALPHA_API_KEYS", "").strip()
        if api_keys_str:
            for k in api_keys_str.split(","):
                k = k.strip()
                if k and k not in keys:
                    keys.append(k)
            return keys

        if not DOTENV_AVAILABLE:
            return keys

        project_root = Path(__file__).resolve().parent.parent
        candidate_env_files = [
            project_root / "api" / "utils" / "keys.env",
            project_root / "keys.env",
            project_root / ".env",
        ]
        for env_path in candidate_env_files:
            if env_path.exists():
                load_dotenv(env_path, override=False)
                api_keys_str = os.getenv("WOLFRAM_ALPHA_API_KEYS", "").
    strip()
                if api_keys_str:
                    for k in api_keys_str.split(","):
                        k = k.strip()
                        if k and k not in keys:
                            keys.append(k)
                if keys:
                    break

        return keys

    @classmethod
    def _get_next_key(cls, keys: List[str]) -> str:
        if not keys:
            return ""
        with cls._key_lock:
            key = keys[cls._key_index % len(keys)]
            cls._key_index = (cls._key_index + 1) % len(keys)
            return key

    def __init__(self, cfg=None, use_zh=False):
        super().__init__(cfg, use_zh)
        self.api_keys: List[str] = []
        self.max_retries = 3
```

```
        if cfg is not None:
            cfg_keys = cfg.get("wolfram_api_keys", [])
            if isinstance(cfg_keys, str):
                cfg_keys = [k.strip() for k in cfg_keys.split(",") if k.
    strip()]
            elif isinstance(cfg_keys, list):
                cfg_keys = [k.strip() for k in cfg_keys if isinstance(k,
    str) and k.strip()]
            self.api_keys = cfg_keys
            self.max_retries = cfg.get("wolfram_max_retries", 3)

        if not self.api_keys:
            self.api_keys = self._load_wolfram_keys_from_env()

    @staticmethod
    def _looks_like_interpretation(text: str) -> bool:
        if not text:
            return True
        t = text.strip().lower()
        patterns = (
            "input interpretation",
            "solve ",
            " for x",
            " for y",
            " for z",
        )
        return any(p in t for p in patterns)

    @staticmethod
    def _parse_xml_response(xml_text: str) -> Dict:
        root = ET.fromstring(xml_text)
        success_raw = root.attrib.get("success", "false")
        success = str(success_raw).lower() == "true"
        if not success:
            return {"error": "Your Wolfram query is invalid. Please try a
     new query."}

        answer = ""
        for pod in root.findall("pod"):
            title = pod.attrib.get("title", "")
            if title == "Solution":
                subpods = pod.findall("subpod")
                if subpods:
                    plaintext = subpods[0].findtext("plaintext") or ""
                    answer = plaintext
            if title in {"Results", "Solutions"}:
                subpods = pod.findall("subpod")
                for i, sub in enumerate(subpods):
                    text = sub.findtext("plaintext") or ""
                    answer += f"ans {i}: {text}\n"
                break

        if not answer:
            preferred_titles = {
                "Result",
                "Results",
                "Solution",
                "Solutions",
                "Exact result",
                "Decimal approximation",
            }
            for pod in root.findall("pod"):
                title = pod.attrib.get("title", "")
                if title not in preferred_titles:
                    continue
```

```
                sub = pod.find("subpod")
                if sub is not None:
                    text = (sub.findtext("plaintext") or "").strip()
                    if text and not SolveMathEquationTool.
_looks_like_interpretation(text):
                        answer = text
                        break

        if not answer or SolveMathEquationTool._looks_like_interpretation
(answer):
            return {"error": "No good Wolfram Alpha result was found."}
        return {"result": answer.strip()}

    @staticmethod
    def _is_retryable_error(error_msg: str) -> bool:
        error_lower = error_msg.lower()
        retryable_patterns = ["rate limit", "quota", "too many requests",
 "403", "429"]
        return any(p in error_lower for p in retryable_patterns)

    def _query_via_http(self, query: str, api_key: str) -> Dict:
        try:
            resp = requests.get(
                "https://api.wolframalpha.com/v2/query",
                params={"appid": api_key, "input": query},
                timeout=30,
            )
            if resp.status_code != 200:
                return {
                    "error": (
                        f"WolframAlpha HTTP error: code={resp.status_code
}, "
                        f"body_preview={resp.text[:240]}"
                    ),
                    "_retryable": resp.status_code in (403, 429),
                }
            return self._parse_xml_response(resp.text)
        except Exception as e:
            return {
                "error": f"HTTP request failed: {type(e).__name__}: {e}",
                "_retryable": self._is_retryable_error(str(e)),
            }

    def _query_with_retry(self, query: str) -> Dict:
        tried_keys = set()
        last_error = None

        for _ in range(min(self.max_retries, len(self.api_keys))):
            current_key = self._get_next_key(self.api_keys)
            if current_key in tried_keys:
                continue
            tried_keys.add(current_key)

            result = self._query_via_http(query, current_key)
            if "error" not in result or not result.pop("_retryable",
False):
                return result
            last_error = result

        if last_error:
            last_error.pop("_retryable", None)
        return last_error or {"error": "All API keys exhausted."}

    def call(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)
```

```
        query = params_dict["query"].strip()

        if not query:
            return {"error": "Query cannot be empty."}

        if not self.api_keys:
            return {
                "error": (
                    "WOLFRAM_ALPHA_API_KEYS is not set. "
                    "Set it in config or environment (.env / keys.env)."
                )
            }

        return self._query_with_retry(query)
```

Web Search. This tool queries external search engines using provided text keywords to retrieve up-to-date factual information. It returns the top relevant text snippets to supplement the internal knowledge of the agent.

```
class WebSearchTool(BasicTool):
    name = "web_search"
    description_en = (
        "Search the web for information using a text query. "
        "Returns the top-k results, each with a title and snippet."
    )

    parameters = {
        "type": "object",
        "properties": {
            "query": {
                "type": "string",
                "description": "Search query text",
            },
            "top_k": {
                "type": "integer",
                "description": "Number of results to return (1-5, default
    : 3)",
            },
        },
        "required": ["query"],
    }
    example = '{"query": "Which city is the capital of China?", "top_k":
    3}'

    def __init__(self, cfg: Optional[Dict] = None, use_zh: bool = False):
        super().__init__(cfg, use_zh)
        self._max_retries = 3
        self._retry_delay = 2

    def call(self, params: Union[str, Dict]) -> Dict:
        params_dict = self.parse_params(params)
        query = params_dict.get("query", "").strip()
        top_k = max(1, min(int(params_dict.get("top_k", 3)), 5))

        if not query:
            return {"error": "Parameter 'query' is required"}

        try:
            return self._text_search(query, top_k)
        except Exception as e:
            logging.error(f"Web search failed: {e}")
            return {"error": f"Search failed: {str(e)}"}

    async def call_async(self, params: Union[str, Dict]) -> Dict:
        return await asyncio.to_thread(self.call, params)
```

```
    def _text_search(self, query: str, top_k: int) -> Dict:
        url = "https://google.serper.dev/search"
        headers = {
            "X-API-KEY": GOOGLE_SEARCH_API_KEY,
            "Content-Type": "application/json",
        }
        payload = {
            "q": query,
            "gl": "cn",
            "hl": "zh-cn",
            "location": "China",
            "num": top_k,
        }

        for attempt in range(self._max_retries):
            try:
                resp = requests.post(
                    url, headers=headers, data=json.dumps(payload),
    timeout=30
                )
                resp.raise_for_status()
                data = resp.json()
                results = data.get("organic", [])[:top_k]

                if not results:
                    return {"search results": "No results found"}

                lines = []
                for i, res in enumerate(results, 1):
                    title = res.get("title", "")
                    snippet = res.get("snippet", "")
                    lines.append(f"[{i}] Title: {title}")
                    lines.append(f"Snippet: {snippet}")
                    lines.append("")

                return {"search results": "\n".join(lines).strip()}
            except Exception as e:
                logging.warning(
                    f"Request failed (attempt {attempt + 1}/{self.
    _max_retries}): {e}"
                )
                if attempt < self._max_retries - 1:
                    time.sleep(self._retry_delay)
                else:
                    raise
```

F Efficiency Analysis

This section provides a detailed breakdown of the online and offline costs introduced by stateful experience retrieval. Throughout, we use Qwen3-VL-32B-Instruct as the base model and Qwen3-VL-8B-Embedding as the retrieval encoder, both served via vLLM on a single node with eight H800 GPUs, with 10-way client concurrency for both offline bank construction and online evaluation. Table 5 summarizes the per-benchmark efficiency profile.

Online retrieval overhead. The added per-query cost of MuSEAgent over a tool-using ReAct baseline comes entirely from the Deep-and-Wide search and not from extra base-model invocations: the per-step retrieval-view selection is produced jointly with the action plan within a single base-model call. At each retrieval-active step, the agent issues at most $L = 3$ viewpoint embeddings under the default configuration (Section 4.2), and embedding inference under vLLM is more than an order of magnitude faster than a base-model call on the same hardware, so retrieval latency is dominated by these embeddings rather than by base inference. Bank similarity search runs in-

Table 5: Per-benchmark efficiency profile of MuSEAgent with Qwen3-VL-32B-Instruct on 8×H800 under vLLM (10-way client concurrency). Bank build time uses GPT-4o as the hindsight reasoner (the default in Section 4.2); evaluation wall-clock is end-to-end including all tool calls and Deep-and-Wide search.

Metric	V* Bench	MME-RW-Lite	Zoom-Bench	HR-Bench	Total
Exploration trajectories	95	959	310	100	1,464
Bank size $\|\mathcal{B}\|$	355	3,352	1,008	401	5,116
Bank build time	5 min	48 min	16 min	5 min	74 min
Evaluation queries	96	960	311	100	1,467
Avg. agent steps / query	6.6	9.1	7.8	7.9	—
Evaluation wall-clock	6 min	4h 37m	33 min	32 min	5h 48m

memory across the $M = 15$ viewpoint indices via batched FP16 cosine matmul on a single GPU and contributes a negligible fraction of step latency. The injected payload is at most $L \cdot K = 9$ short guidance snippets, bounded by L and K rather than by $|\mathcal{B}|$, so neither retrieval latency nor token overhead grows as more exploration trajectories are added.

Offline experience construction. The offline cost is incurred once per exploration trajectory and is fully amortized across all subsequent evaluation queries. The $1{,}464$ exploration trajectories collected from Table 4 decompose into roughly $8{,}800$ atomic transitions, of which $5{,}116$ pass the $\delta = 5.0$ admission threshold and populate the four per-benchmark banks (Table 5). Each atomic transition is evaluated by $\mathcal{Q}_\phi$ to produce its quality score and guidance summary, conditioned on the realized trajectory outcome. Under the default setting in Section 4.2 where GPT-4o serves as the hindsight reasoner, building all four banks takes 74 minutes of wall-clock time and totals around $\$20$ in API spend at standard GPT-4o rates, after which the same banks are reused across all subsequent evaluation queries. Under the *self-hindsight* setting (Section 4.5), where Qwen3-VL-32B-Instruct itself serves as the hindsight reasoner, this monetary cost is replaced by additional vLLM inference time on the same in-house hardware; either path produces a comparable bank without further offline cost.

Online evaluation cost is benchmark-dominated. Across the four benchmarks (Table 5), the per-benchmark wall-clock spans from 6 minutes (V* Bench, 96 queries) to 4 hours 37 minutes (MME-RealWorld-Lite, 960 queries). The roughly $46\times$ gap mirrors the corresponding $\sim 10\times$ growth in evaluation queries combined with the $\sim 1.4\times$ increase in average agent steps per query, rather than any retrieval-side scaling: average steps per query stay within 6.6–9.1 across all benchmarks, and the bank similarity search remains in-memory regardless of $|\mathcal{B}|$. The dominant per-query cost is therefore base-model inference, not Deep-and-Wide search.

Scaling behavior. Because L and K are fixed configuration parameters, both the per-query retrieval latency and the per-query token overhead remain flat as the experience bank grows. Adding more exploration trajectories therefore enlarges the offline construction cost linearly while leaving online inference cost unchanged, which makes the offline cost increasingly amortizable as the bank is queried by more deployment traffic.

G Quality Score Threshold Sensitivity

The quality threshold δ controls the admission criterion of the experience bank: only transitions whose hindsight score $q_t \geq \delta$ are retained. A strict threshold retains only near-optimal sub-decisions, whereas a permissive threshold admits a broader range of partially successful or cautionary moves. To examine whether the default threshold $\delta = 5.0$ constitutes a genuine performance optimum rather than an artifact of partial exploration, we conduct an extended sweep over $\delta \in \{9.0, 7.0, 5.0, 3.0, 1.0\}$ with Qwen3-VL-32B-Instruct on V* Bench and HR-Bench, holding all other components at default.

An inverted-U curve confirms an optimal quality-diversity trade-off. As presented in Table 6, accuracy follows an inverted-U trajectory across both benchmarks, peaking sharply at the default threshold $\delta = 5.0$ (85.42% on V* Bench and 78.00% on HR-Bench) and dropping on both sides. When the threshold is overly strict ($\delta \geq 7.0$), the bank retains only textbook successes while discarding near-misses, corrective adjustments, and tool-failure warnings that carry intermediate scores.

Table 6: Extended sensitivity of MuSEAgent (Qwen3-VL-32B-Instruct) to the quality score threshold δ. We report accuracy (%) across both benchmarks. Best results are highlighted in **bold**.

Quality Threshold δ	Admission Criterion	V* Bench (Micro Avg.)	HR-Bench	Average
$\delta = 9.0$	Highly selective	75.00	73.00	74.00
$\delta = 7.0$	Selective	78.12	74.00	76.06
$\delta = 5.0$ (Default)	Balanced	**85.42**	**78.00**	**81.71**
$\delta = 3.0$	Permissive	78.12	72.00	75.06
$\delta = 1.0$	Highly permissive	76.04	61.00	68.52

Discarding these leaves the agent vulnerable to repeated missteps during online inference, pulling accuracy down to 75.00% on V* Bench and 73.00% on HR-Bench. Conversely, when the threshold is excessively relaxed ($\delta \leq 3.0$), low-quality and uninformative transitions enter the experience bank, diluting high-value tactical signals with noisy advice. This degradation is especially severe on HR-Bench at $\delta = 1.0$, where accuracy plummets to 61.00% (-17.00 points below default). These empirical results confirm that $\delta = 5.0$ operates as a genuine sweet spot: it is permissive enough to preserve cautionary guidance from failed attempts while maintaining the quality necessary to prevent retrieval pollution.

H Case Study

Figure 3 traces a representative V* Bench instance end-to-end, showing how the abstraction phase populates the bank and the exploitation phase consumes it on a structurally similar query.

Offline: hindsight scoring turns one trajectory into differentiated experiences. The left panel records a fine-grained attribute recognition task where the agent runs object localization, image cropping, OCR, and text-image similarity matching to identify the color of a specified umbrella. Instead of storing this trajectory as a single retrievable unit, MuSEAgent evaluates each state-action pair independently: the initial localization is scored 9 because attribute reasoning is impossible without first isolating the target region, while OCR on the cropped region is scored 1 because text extraction is irrelevant for a purely visual color attribute. The two scores produce two qualitatively different experiences, one prescribing localization-before-attribute-analysis and one warning against OCR on visual-only queries, that would be entangled and washed out under trajectory-level retrieval.

Online: multi-viewpoint retrieval converges on a consistent action. On a similar query, the right panel shows the agent decomposing its current state into distinct semantic viewpoints (e.g., query with visual input, query with task description) and querying the bank under each. Different viewpoints surface different past experiences, but their natural-language guidance independently agrees on the same prescription, spatial localization before detailed visual analysis, which the agent then commits to as its action plan. This convergence across viewpoints is exactly the signal the union-based aggregation in Algorithm 2 is designed to amplify: agreement under heterogeneous queries is stronger evidence than a single high-similarity match.

What the case validates. The two panels concretize three design choices: (i) hindsight scoring attaches differentiated quality to atomic decisions rather than caching monolithic trajectories; (ii) low scores on ineffective tool calls (e.g., OCR scoring 1) preserve cautionary signals that trajectory-level retrieval would absorb as noise; and (iii) Deep-and-Wide search leverages multi-viewpoint convergence to remain robust against the noisy, non-deterministic outputs typical of multimodal tool use.

I Statistical Significance and Human Verification

I.1 Multi-Seed Statistical Significance Evaluation

Because the evaluation splits for V* Bench (96 questions) and HR-Bench (100 questions) are relatively compact, we conduct multi-seed evaluations over three random seeds on Qwen3-VL-32B-Instruct to verify that observed gains are stable against run-to-run variance and stochastic tool decoding. As presented in Table 7, the 11–12 point margins between MuSEAgent and Expel are

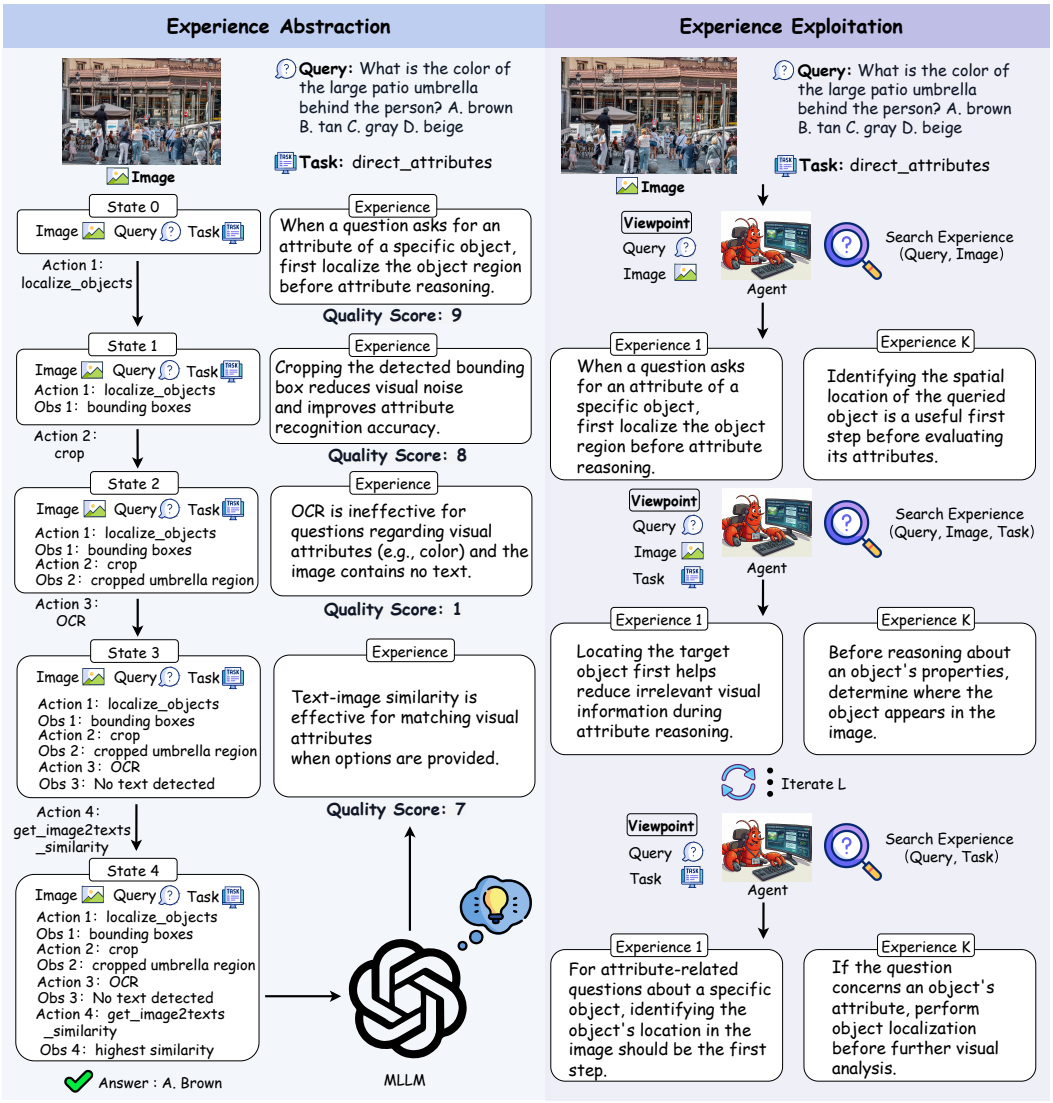


Figure 3: A case of stateful experience abstraction and exploitation.

several times the standard deviations, producing non-overlapping confidence intervals across both benchmarks (e.g., Expel's upper bound of 75.99% on V* Bench remains well below MuSEAgent's lower bound of 82.91%).

Table 7: Multi-seed evaluation (mean $\pm$ std across 3 random seeds) on V* Bench and HR-Bench with Qwen3-VL-32B-Instruct. Non-overlapping intervals confirm that performance gains are statistically robust against run-to-run variance.

Method	V* Bench (Micro Avg.)	HR-Bench
w/ Expel (Zhao et al., 2024)	72.92 ± 3.07	65.00 ± 2.94
w/ MuSEAgent	$\mathbf{84.38 \pm 1.47}$	$\mathbf{77.00 \pm 1.41}$

I.2 Human Verification of Hindsight Scoring Quality

To rigorously assess the fidelity and safety of hindsight scoring produced by the multimodal judge $\mathcal{Q}_\phi$ (Section 3.2), we conduct a human verification study on a uniformly sampled subset of 200 atomic state–action transitions across all four evaluation benchmarks. Three human annotators independently evaluated each transition on the same $[0, 10]$ quality scale and labeled each generated

natural-language guidance string along two operational criteria: (i) **Guidance correctness**: whether the generated advice conveys valid, procedurally sound guidance for the current decision state; and (ii) **Answer leakage**: whether the guidance prematurely reveals ground-truth target identities or options, which would trivially shortcut the task rather than fostering genuine reasoning. Each continuous rating is converted into a binary keep-or-discard decision at the default threshold $\delta = 5.0$, with annotator judgments aggregated by majority consensus.

Table 8: Human verification of hindsight scoring and guidance quality across 200 sampled state transitions evaluated by three independent annotators (majority consensus).

Evaluation Metric	**Agreement / Prevalence Rate**
Human–$\mathcal{Q}_\phi$ agreement on keep/discard decision ($q_t \geq 5.0$)	88.5%
Experiences providing procedurally correct guidance	91.0%
Experiences exhibiting ground-truth answer leakage	1.5%

High decision alignment and minimal answer leakage. As summarized in Table 8, human annotators exhibit an 88.5% agreement with $\mathcal{Q}_\phi$ regarding whether an atomic transition satisfies the admission criterion $\delta \geq 5.0$. Because the experience bank is populated via thresholded filtering rather than exact cardinal scores, this high alignment indicates that bank composition faithfully reflects human judgment regarding decision quality. Furthermore, 91.0% of surviving guidance strings provide verified, actionable advice tailored to visual tool execution, while ground-truth answer leakage is restricted to merely 1.5% of cases. This confirms that hindsight-distilled guidance conveys modular visual tactics rather than memorized task answers.

J Tool-Policy Shift and Web Search Safety Analysis

Because retrieved experiences provide explicit tactical advice regarding tool selection, a natural concern is whether performance gains stem from an unintended policy shift toward external knowledge retrieval—specifically, querying the web search tool to retrieve benchmark answers directly rather than reasoning over visual observations. To rule out this degenerate shortcut, we audit tool invocation logs across all methods and conduct an ablation where the web search tool is disabled entirely.

Table 9: Average web search invocations per query across all four evaluation benchmarks, and impact of disabling web search on V* Bench and HR-Bench with Qwen3-VL-32B-Instruct.

(a) Web search calls per query across benchmarks.

Method	**Web Calls / Query**
ReAct (Yao et al., 2022)	0.0082
Reflexion / Expel (Shinn et al., 2023; Zhao et al., 2024)	0.0086
MuSEAgent	**0.0057**

(b) Ablation: removing web search.

Configuration	**V* Bench**	**HR-Bench**
w/ Web Search	**85.42%**	**78.00%**
w/o Web Search	84.38%	76.00%
Δ Drop	-1.04%	-2.00%

MuSEAgent invokes external search the least and relies strictly on visual tools. As reported in Table 9(a), every evaluated method invokes web search fewer than once per hundred queries, with MuSEAgent calling it the least (0.0057 calls per query, representing under 0.5% of all executed tool calls). Furthermore, when web search is completely disabled (Table 9(b)), MuSEAgent's accuracy drops by merely 1.04 points on V* Bench and 2.00 points on HR-Bench, remaining more than 10 points above the strongest trajectory-level baselines. This confirms that MuSEAgent's gains are driven by targeted spatial perception tools (such as localization, cropping, and zooming) rather than external search leakage.

K Viewpoint Selection Dynamics and Single-Component Analysis

To provide complete transparency into the compositional representation space $\mathcal{P}$, we report the full distribution of inference-time viewpoint selections across all evaluated benchmarks, alongside an ablation measuring the standalone retrieval power of each individual state component on V* Bench with Qwen3-VL-32B-Instruct.

Table 10: Inference-time viewpoint selection frequencies across all benchmarks (left) and standalone single-component retrieval accuracy on V* Bench (right).

(a) Selection frequencies among 15 viewpoints.

Semantic Viewpoint Composition	Selection %
Query + Execution History	48.03%
Query + Visual Observation	23.97%
Query + Task Descriptor + Execution History	9.08%
Query + Visual + Execution History	6.29%
Query + Task Descriptor + Visual	6.11%
Full Context (u, v_t, d, H_t)	5.75%
Remaining 9 combinations (summed)	0.77%

(b) Single-component view retrieval.

Query Component	V* Bench
Query u only	65.62%
Visual v_t only	73.96%
Task d only	77.08%
History H_t only	**81.25%**
Deep-and-Wide (Ours)	**85.42%**

Empirical insights on representation decoupling. The full distribution in Table 10(a) reveals two critical properties: first, policy selections are highly concentrated, with the top two decoupled compositions (Query + History and Query + Vision) accounting for 72.00% of all retrieval actions; second, the monolithic full-context viewpoint is chosen in only 5.75% of steps. Meanwhile, Table 10(b) demonstrates that execution history (81.25%) and task descriptors (77.08%) carry substantially stronger retrieval signals than queries alone (65.62%). Combining these complementary facets via Deep-and-Wide search reaches 85.42%, outperforming every single-component view.

L Failure Mode and Qualitative Negative Analysis

To examine the failure modes of stateful experience retrieval, we audit execution traces where MuSEAgent underperforms the experience-free ReAct baseline. While such occurrences are rare across our benchmark logs, they localize an informative failure mechanism: *off-target tactical guidance*.

Off-target guidance can induce context loss on relational tasks. A representative failure instance from V* Bench involves evaluating relative spatial relations between two clearly visible objects: *"Is the red armchair to the left or right of the standing lamp?"*. Under the zero-experience ReAct baseline, the base model inspects the full visual observation and answers correctly in a single step without invoking tools, as both objects are prominently visible and their spatial configuration is immediately apparent. Under MuSEAgent, however, the initial retrieval round queries the experience bank and retrieves advice distilled from fine-grained attribute-recognition experiences, which strongly advises: *"When evaluating object attributes, first localize and crop the target entity to eliminate visual distractors."* Following this off-target guidance, the agent invokes the localization tool on the armchair and crops tightly around it, inadvertently discarding the second object (the standing lamp) required for the spatial relation comparison. Unable to compare the relative positions from the cropped view, the agent hallucinates an incorrect spatial answer.

Analysis and mitigation. This failure mode demonstrates that while natural-language guidance provides concrete procedural directives, following plausible advice that does not fit the global task structure can lead to premature context reduction or unnecessary tool calls. Deep-and-Wide search mitigates this risk by querying multiple semantic viewpoints and requiring multi-view consensus; however, when an off-target viewpoint produces a high-similarity match, it can still occasionally misdirect downstream tool selection. Developing adaptive confidence gating to dynamically suppress retrieval when direct visual observation is already sufficient represents an important direction for future work.

M Limitations

While MuSEAgent achieves consistent empirical gains across diverse multimodal reasoning benchmarks and base model architectures, several limitations warrant discussion. First, the current experience bank is constructed offline from a fixed exploration set and remains frozen during evaluation; developing continual, online experience learning where agents autonomously update, prune, and refine their memory during deployment represents an important direction for future exploration. Second, while our benchmarks focus on complex visual perception and high-resolution spatial rea-

soning, extending stateful experience abstraction to dynamic, open-ended interactive environments (such as web navigation or operating system control) requires modeling state changes resulting from persistent external actions. Finally, extending the compositional state representation to temporally extended video streams or 3D multimodal environments presents an exciting opportunity to scale stateful memory beyond static visual tasks.

N PROMPTS

N.1 EVALUATION PROMPT

Evaluation Prompt

Role: You are a multimodal assistant with tool access and memory.

Task: Analyze images and questions carefully, decide whether tool usage is necessary, and use tools strategically when they help obtain information that cannot be reliably determined through direct observation.

Tool Usage Guidelines:
Before using tools, analyze and plan:

- Carefully analyze the task requirements and image content
- Identify what information is required to answer the question
- Evaluate which tools are suitable for the current task
- Plan the sequence of tool usage if multiple tools are needed
- Consider how outputs from multiple tools can be combined to derive the final answer

When tools are appropriate:

- When examining details that are difficult to see directly (e.g., small text, tiny objects, subtle textures)
- When precise localization or counting is required
- When the image content is complex and requires step-by-step regional analysis

When tools are not appropriate:

- When cropping or zooming would remove important contextual information
- When the answer can already be determined through direct observation

Best practices when using tools:

- Clear objectives: know exactly what information you want before calling a tool
- Accurate regions: ensure the selected region corresponds to the part that requires analysis
- Analyze outputs critically and combine them with direct observation
- Do not replace visible evidence with speculative inference

If a tool call fails or returns an error, adapt by trying alternative tools or adjusting your strategy.

Image Reference:
All images are referenced with unique IDs (img_0, img_1, ...). When calling tools, use these IDs to specify which image to operate on.

[If stateful experience search is enabled]:
Experience Retrieval:
Before each action step, you MUST call `search_experiences` to retrieve advice from similar previously solved tasks.

- Choose a retrieval view that best matches the current information need
- Multiple retrieval calls are allowed if necessary
- Use retrieved advice to guide decision making before taking the next action

N.2 Prompt for Stateful Experience Abstraction

Prompt for Stateful Experience Abstraction

Role: You are evaluating a complete agent reasoning trace.
[If the task was solved CORRECTLY]: The task was solved CORRECTLY. The process may be highly efficient, or it may contain wasteful, redundant, or error-prone steps. Judge each step by the quality and relevance of its output.
[If the task was solved INCORRECTLY]: The task was solved INCORRECTLY. Your goal is to identify which steps were most responsible for the incorrect outcome. A wrong final answer does NOT mean every step was bad — many steps may have been perfectly reasonable. Focus on pinpointing the critical steps that introduced errors, ignored evidence, or led the reasoning astray.

Task:
{images_note}
Question: {question} {type_line}
{tools_section}

Trajectory:
The initial state s_0 is the question and images above. Each subsequent state is what the agent has seen so far, including all previous actions and observations.
{traj_text}

Instructions:
[If the task was solved CORRECTLY]: For each (state, action) pair, provide:
1. q_value (0-10): Rate the quality of this action at this state. Judge whether it produced accurate, relevant information that actually contributed to reaching the answer.

- **9-10 (Essential):** Produced decisive information that directly shaped the answer, or answered correctly at the right time
- **7-8 (Helpful):** Useful output that advanced the solution, with minor room for improvement in tool choice or parameters
- **5-6 (Reasonable):** Valid approach, but output had little actual influence on the final answer; a more direct path existed
- **3-4 (Wasteful):** Produced no new information, duplicated prior knowledge, or was an unnecessary detour
- **0-2 (Harmful):** Produced errors, misleading output, or repeated a known failure

[If the task was solved INCORRECTLY]: Rate how much this step CONTRIBUTED TO the incorrect outcome. A high score means this step was a critical cause of the error.

- **9-10 (Critical error):** Directly caused or cemented the wrong answer (e.g., misread evidence, wrong tool, flawed conclusion)
- **7-8 (Significant):** Substantially misled the reasoning, even if not the sole cause of failure
- **5-6 (Moderate):** Somewhat contributed to the error, but other steps were more decisive
- **3-4 (Minor):** Had little impact on the incorrect outcome; the step was mostly neutral
- **0-2 (Irrelevant):** This step was reasonable and did not contribute to the failure at all

2. experience (1-2 sentences): Actionable advice that a FUTURE agent would see BEFORE making this decision. This advice will be injected into the agent's context at runtime to guide its next action.
Guidelines:

- Reference the task type and tool strategy, but keep the advice generalizable — do not mention specific objects, labels, or values from this particular trace.
- NEVER mention or hint at the correct answer.
- **[If the task was solved CORRECTLY]:** Focus on STRATEGY: recommend the specific tool and approach that works, or advise answering directly if tools are unnecessary. If the step was an error, redundant, or unnecessary, the experience MUST be cautionary.
- **[If the task was solved INCORRECTLY]:** For high `q_value` steps (critical errors), the experience MUST be a cautionary warning — explain what went wrong and what to avoid. For low `q_value` steps, provide brief neutral or positive guidance.

Examples:

- **Good (Strategy):** "For visual similarity tasks, use `get_image2images_similarity` to get objective scores rather than estimating visually."
- **Good (Direct Answer):** "The observation already contains enough information to answer directly. No further tool calls are needed."
- **Good (Correction):** "When observations conflict with prior reasoning, re-examine the evidence before committing to an answer."
- **Bad (Too Vague):** "The agent should use the right tool." / "The agent should be more careful."
- **Bad (Leaks Answer):** "The answer is likely A based on the scores."

Output Format:
Output JSON array:
```
[{"state": 0, "q_value": 8, "experience": "..."}, ...]
```

N.3 Prompt for Trajectory-level Experience Abstraction

Prompt for Trajectory-level Experience Abstraction

Role: You are evaluating a complete agent reasoning trace.

[If the task was solved CORRECTLY]: The task was solved CORRECTLY. Your goal is to summarize what strategy worked well across the reasoning trajectory. Focus on effective tool usage, evidence interpretation, and reasoning strategies that helped reach the correct answer.
[If the task was solved INCORRECTLY]: The task was solved INCORRECTLY. Your goal is to identify what went wrong in the reasoning trajectory and what a future agent should do differently to avoid similar mistakes.

Task:
{images_note}
Question: {question} {type_line}
{tools_section}

Trajectory:
{traj_text}

Instructions:
Write exactly **1–2 concise sentences** of actionable experience for future agents facing similar tasks.
[If the task was solved CORRECTLY]:

- Summarize the strategy that worked well (e.g., tool choice, evidence usage, reasoning approach)
- Highlight patterns that helped reach the correct answer efficiently
- Focus on practices that future agents should replicate

[If the task was solved INCORRECTLY]:

- Identify what went wrong in tool usage, reasoning, or evidence interpretation
- Explain what a future agent should do differently
- Highlight common mistakes or misleading reasoning patterns to avoid

General Guidelines:

- Keep the advice generalizable to similar tasks
- Do not mention specific objects, labels, or values from the trajectory
- NEVER reveal or hint at the correct answer

Output Format:
Output JSON object:
```
{"experience": "your 1-2 sentence experience"}
```