TRACEMEM: TRAJECTORY-INDUCED REUSABLE MEMORY FOR MULTIMODAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal reasoning is a key capability for general-purpose AI systems, requiring models to jointly interpret visual evidence, language instructions, and multi-step inference processes. However, multimodal reasoning models often repeat similar mistakes across related samples, suggesting a limited ability to transform past reasoning outcomes into reusable experience. While memory-based methods preserve prior episodes, they typically store them as weakly structured text, making retrieved memories less actionable. Test-time trajectory exploration can reveal successful and failed reasoning patterns, but these trajectories are usually discarded after the current prediction. We propose **TraceMem**, a trajectory-induced memory framework that converts reasoning trajectories into structured memory units with explicit triggers, procedural guidance, and avoidance rules. These units are maintained through four lifecycle-aware memory banks, allowing experience from successful and failed trajectories to be accumulated, updated, and reused. Experiments on four multimodal reasoning benchmarks show that TraceMem consistently improves performance, demonstrating the effectiveness of trajectory-induced structured memory for multimodal reasoning. The code is available at `https://anonymous.4open.science/r/TraceMem-57C6/`.

1 INTRODUCTION

Multimodal reasoning is central to vision-language systems for chart understanding (Masry et al., 2022), document analysis (Mathew et al., 2021), geometry (Lu et al., 2021), and multimodal question answering (Lu et al., 2024; Yue et al., 2024). These models still solve episodes independently and repeat similar mistakes, because failed reasoning outcomes are rarely retained as actionable guidance.

Prior work follows two directions. Retrieval and memory methods supply external context or stored experience (Chen et al., 2022; Yasunaga et al., 2023; Su et al., 2024; Park et al., 2023; Wang et al., 2024a; Bo et al., 2025), but memory is often stored as free-form text rather than an executable procedure. Trajectory methods instead explore or refine the current sample (Wang et al., 2023; Yao et al., 2023a; Madaan et al., 2023; Shinn et al., 2023), but usually discard the resulting evidence after prediction. The missing link is a mechanism that converts successful and failed trajectories into persistent, retrievable, and updateable memory.

These directions leave a concrete gap. Memory methods preserve experience but often not in directly actionable form, while trajectory methods expose useful success and failure signals without retaining them beyond the current episode.

We introduce TraceMem, which represents experience as structured memory units with triggers, procedures, avoidance rules, task tags, utility, and lifecycle status. Its design follows three choices. First, successful and failed trajectories are the source of reusable experience. Second, experience is rewritten into executable guidance rather than free-form text. Third, memory is promoted, suppressed, or deprecated according to downstream utility. These choices turn test-time exploration into persistent experience rather than one-step repair.

We evaluate TraceMem on four multimodal reasoning benchmarks across two Qwen3-VL scales. It improves pass@1 accuracy over direct prompting and memory baselines.

Our contributions are summarized as follows:

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.

- We introduce trajectory-induced reusable memory for multimodal reasoning.
- We propose structured, lifecycle-aware memory banks with explicit triggers, procedures, avoidance rules, task tags, and utility.
- We evaluate TraceMem on four benchmarks and two Qwen3-VL scales.

2 Related Work

2.1 Memory for Multimodal Reasoning

External retrieval and memory have become a standard way to strengthen multimodal reasoning with additional evidence. Representative examples include MuRAG (Chen et al., 2022), RA-CM3 (Yasunaga et al., 2023), and SK-VQA (Su et al., 2024). These methods show that retrieved external knowledge or stored context can improve multimodal inference, but the reused content is often attached to the current sample as evidence or case-level context rather than as an explicit procedural memory. A related line of persistent-memory agents, including Generative Agents (Park et al., 2023), Voyager (Wang et al., 2024a), and, more directly in multimodal reasoning, ViLoMem (Bo et al., 2025), further shows that experience can be accumulated and reused across episodes. Among them, ViLoMem is the closest baseline because it also maintains persistent visual and logical memory. Its logic bank stores textual reasoning guidelines, while its visual bank stores visual guidelines paired with their source images. During generation, ViLoMem attributes a failure to either the visual or logical stream and produces an error type, analysis summary, and a concise corrective guideline. New guidelines are compared with existing entries by text similarity and then merged or inserted, following a grow-and-refine strategy. At retrieval time, visual memory first uses image-embedding similarity and then filters or reranks candidates using the question text, optionally producing question-aware attention maps; logical memory uses text similarity over an analyzer-enriched query. Thus, a ViLoMem memory entry is primarily an error-derived corrective guideline, with visual entries additionally anchored to source images.

TraceMem differs from ViLoMem in the granularity and control policy of memory. Instead of storing one corrective statement per entry, TraceMem represents each item as an action schema $m = (c, p, a, t, u, s)$ with explicit trigger conditions, procedures, avoidance rules, task tags, utility, and lifecycle status. It also explicitly separates success-derived procedures from failure-derived avoidance rules across four outcome- and modality-aware banks. Finally, TraceMem updates each retrieved unit according to its downstream utility and uses the resulting lifecycle state during future retrieval, whereas ViLoMem's grow-and-refine update merges or creates entries based on similarity. Our focus is therefore not only to retrieve past content, but to maintain memory in a form that is easier to trigger, execute, and refine over time.

2.2 Multi-Trajectory Test-Time Reasoning

Test-time reasoning methods improve performance by exploring multiple candidate trajectories during inference. Self-Consistency (Wang et al., 2023) aggregates diverse reasoning paths, Tree of Thoughts (Yao et al., 2023a) casts reasoning as explicit search, Self-Refine (Madaan et al., 2023) iteratively revises intermediate outputs, Reflexion (Shinn et al., 2023) uses verbal feedback to improve later attempts, and Dynamic Cheatsheet (Suzgun et al., 2026) constructs an external memory that stores successful and unsuccessful strategies from past inferences, enabling test-time learning through adaptive memory. These ideas have also influenced longer-horizon agentic systems such as Voyager (Wang et al., 2024a), where trajectories can be consolidated into reusable skills. What remains less explored is how to turn the contrast between successful and failed trajectories into persistent multimodal reasoning experience. Our framework builds on the same source of signal, but shifts the question from how to improve the current answer to what should survive the current episode as reusable experience for later samples.

2.3 Agentic Multimodal Reasoning

Recent multimodal reasoning systems increasingly adopt agentic test-time behavior, combining generation, planning, feedback, and tool use. ReAct (Yao et al., 2023b) and Inner Monologue (Huang et al., 2023) illustrate the broader reasoning-and-feedback paradigm, while MM-REACT (Yang

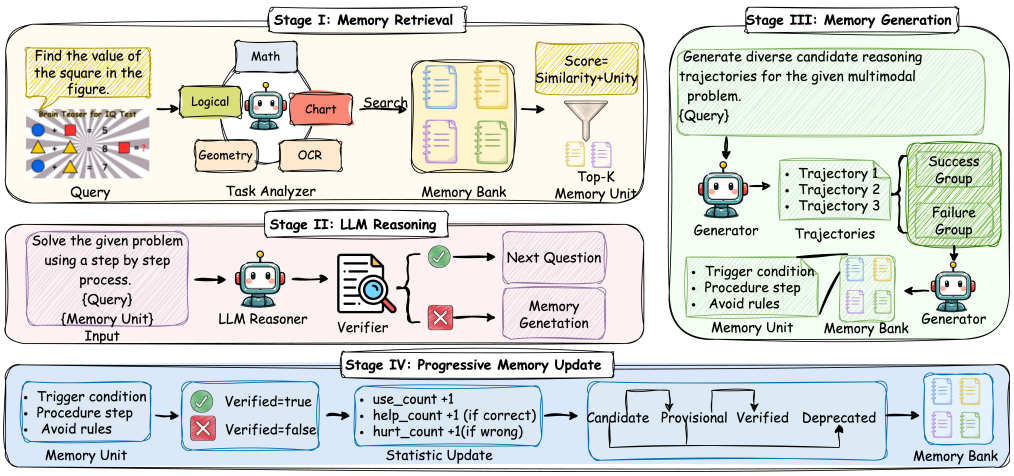

Figure 1: Overview of TraceMem. The framework forms a four-stage loop of memory retrieval, first-pass reasoning, memory induction, and progressive update.

et al., 2023) and Visual Program Distillation (Hu et al., 2024) adapt similar ideas to multimodal settings. Benchmarks such as OSWorld (Xie et al., 2024) further emphasize that strong multimodal performance requires iterative reasoning, verification, and adaptation. Our framework is aligned with this trend toward more deliberative multimodal inference, but differs in emphasis: instead of improving only the current trajectory, it focuses on what reusable reasoning experience should persist after the trajectory ends.

3 Method

3.1 Overview

TraceMem is a closed loop of memory retrieval, first-pass reasoning, trajectory-based induction, and memory evolution. Given an input sample, it retrieves memory from four banks: logical-success, logical-failure, visual-success, and visual-failure. The retrieved units guide the first answer. If verification fails, TraceMem samples additional trajectories, induces new memory from successful and failed groups, and updates the memory that was retrieved. Figure 1 summarizes the pipeline.

Correct answers terminate the episode and update retrieved-memory statistics. Failed answers launch additional reasoning trajectories, from which the system separates successful and failed groups before inducing new memory. Previously retrieved memory is updated using the observed downstream outcome.

3.2 Structured Memory Representation

Each memory item is represented as

$$m = (c, p, a, t, u, s),$$

where c, p, a, t, u, and s denote trigger conditions, procedure steps, avoidance rules, task tags, utility, and lifecycle status. Memory is partitioned into logical/visual and success/failure banks. Success memory stores reusable procedures, failure memory stores avoidance rules, and the modality split prevents visual and logical cues from interfering with one another. The explicit fields make memory executable through procedure-level actions and maintainable through utility and status signals. Detailed bank and field design is provided in Appendix A.2.

In particular, trigger conditions specify when memory should activate; procedure steps summarize reusable reasoning or observation actions; avoidance rules record mistakes to prevent. Task tags capture the dominant problem type, such as chart reasoning, OCR, geometry, or multi-step logic. Utility records historical effectiveness, while lifecycle status distinguishes newly created, tentatively

reliable, mature, and deprecated entries. This decomposition keeps visual and logical guidance separate so that one does not suppress evidence from the other.

This representation improves both executability and maintainability. Retrieved memory already contains procedure-level actions and explicit avoidance constraints, so it can be injected directly into the reasoning prompt. Utility and status fields also make memory quality explicit rather than leaving it implicit in a static text repository.

3.3 Task-Aware Memory Retrieval

Given visual input x and question q, a task analyzer predicts

$$z = (\text{task tags}, \text{question focus}, \\ \text{visual requirement}, \text{logic requirement}).$$

For each bank and candidate memory m, the retrieval score is

$$\text{score}(m) = \alpha \cdot \text{sim}(q, m) + \beta \cdot \text{util}(m), \tag{1}$$

where sim is semantic similarity and util is current utility. Retrieval is bank-wise; at most one positive and one negative item are selected per modality. This keeps prompts compact and prioritizes relevant, historically useful memory. It also separates task awareness, which controls what is relevant, from quality awareness, which favors memory with stronger past utility. Exact coefficients and status bonuses are given in Appendix A.3, with rationale in Appendix A.2.

The analyzer identifies dominant requirements through task tags such as chart, reasoning, OCR, geometry, and counting. The query then combines question text, task tags, and question focus. This design improves over plain memory lookup: retrieval is conditioned on the current task and filtered within each modality-specific bank, rather than returning globally similar but contextually inappropriate memories.

After scoring, TraceMem selects a small number of units and injects them into the initial prompt. Retrieval remains bank-wise so that positive and negative memory are balanced rather than allowing one large bank to dominate. This also keeps the amount of injected context stable across samples. In the default configuration, each modality contributes at most one positive and one negative memory item.

3.4 LLM Reasoning

The query and retrieved memory form a step-by-step prompt for the backbone model. The resulting answer is checked by a verifier combining VLMEval matching, answer parsing, Math-Verify, and an LLM judge fallback. Correct answers update usage statistics; failed answers enter trajectory induction.

3.5 Trajectory Memory Induction

When the first answer fails verification, TraceMem samples candidate reasoning trajectories

$$\mathcal{R} = \{r_1, r_2, \ldots, r_K\}.$$

The verifier partitions them into successful and failed groups, $\mathcal{R}^+$ and $\mathcal{R}^-$. A distillation module then produces

$$\mathcal{M}_{\text{new}} = \Phi(\mathcal{R}^+, \mathcal{R}^-, z),$$

where Φ rewrites short step sequences into procedure or avoidance rules. The induction step produces logical/visual success memory from successful procedures and logical/visual failure memory from repeated mistakes. Trajectories with no successful attempt only produce failure-side memory. Appendix A.3 gives sampling, merge, and filtering rules; Appendix A.2 gives the design rationale.

This step is what distinguishes TraceMem from local answer repair. A successful trajectory is compressed into a trigger-conditioned procedure, while failed attempts are converted into explicit avoidance rules. The two forms are stored separately so that later retrieval can choose whether to imitate a successful strategy or explicitly avoid a known failure pattern.

The distillation module extracts short step sequences from each trajectory before rewriting them. Positive banks receive procedure-oriented entries, while negative banks receive avoidance rules. New entries are merged with highly similar existing memory or inserted as new units when the evidence is sufficiently consistent.

3.6 Progressive Memory Update

Each memory item tracks usage, success, and failure counts, n_{use}, n_{help}, and n_{hurt}. Its utility is

$$u(m) = \frac{n_{\text{help}}(m)}{\max(1, n_{\text{use}}(m))} - \lambda \cdot \frac{n_{\text{hurt}}(m)}{\max(1, n_{\text{use}}(m))},$$

where λ penalizes harmful memory. Items move through *candidate*, *provisional*, *verified*, and *deprecated* states based on utility, and status affects future retrieval. This provides memory self-regulation: reliable items are strengthened while ineffective items are suppressed. Appendix A.3 gives thresholds and bonuses; Appendix A.2 discusses the credit-assignment limitation.

Here n_{use} counts retrievals, n_{help} counts uses associated with verified answers, and n_{hurt} counts uses followed by failure. Newly created memory starts as a candidate and is promoted only after repeated positive evidence. Items with persistently low utility are deprecated rather than permanently removed, preserving the possibility of reassessment if their context reappears.

When several memories are retrieved for one episode, they currently receive the same episode-level feedback. This shared credit assignment is simple and stable, but it can obscure which individual item caused a success or failure. The lifecycle mechanism should therefore be viewed as practical memory regulation rather than exact causal attribution.

In the default schedule, a memory item is promoted from candidate to provisional after two successful uses and to verified after four uses with utility at least 0.2. An item is deprecated if its utility stays at or below -0.1 after at least two uses. These thresholds make promotion and suppression depend on accumulated evidence rather than a single outcome.

4 Experiments

4.1 Experimental Setup

We evaluate the framework on a diverse suite of multimodal reasoning benchmarks. The benchmark set includes MathVista (Lu et al., 2024), MathVision (Wang et al., 2024b), HallusionBench (Guan et al., 2024), and MMStar (Chen et al., 2024). Together, these datasets test knowledge-intensive reasoning, visually grounded mathematical problem solving, robustness to misleading visual evidence, and broad multimodal understanding in realistic settings.

MathVista evaluates mathematical reasoning in visual contexts, while MathVision focuses on visually rich mathematical problems. HallusionBench targets language hallucination and visual illusion, and MMStar measures broad multimodal understanding with reduced language-only shortcuts. Together, these benchmarks cover complementary reasoning and grounding demands.

We instantiate the framework on Qwen3-VL backbones (Bai et al., 2025), specifically Qwen3-VL-235B-A22B-Instruct and Qwen3-VL-8B-Instruct, to test whether the proposed memory mechanism generalizes across model scales within a shared family. For each backbone, we compare against direct single-pass inference, a stronger step-by-step prompt baseline, Dynamic Cheatsheet (Suzgun et al., 2026) as a test-time learning baseline with adaptive external memory, and ViLoMem as the most relevant prior dual-memory framework (Bo et al., 2025). For all comparisons, we keep the benchmark split, backbone identity, and answer metric fixed. Pass@1 accuracy is the primary metric because our goal is to improve the model's first answer rather than only its best-of-many answer after exploration.

To test backbone generality, we also evaluate GPT-4.1 through its official API. We apply the same memory retrieval, verification, and evaluation protocols, with all model calls made through the API rather than local weights.

Table 1: Main results across Qwen3-VL and GPT-4.1 backbones on the benchmarks shared with ViLoMem (Bo et al., 2025). We report pass@1 accuracy on MathVista, MathVision, HallusionBench, and MMStar. TraceMem results are mean ± standard deviation over five runs.

Method	MathVista	MathVision	HallusionBench	MMStar
Qwen3-VL-235B-A22B-Instruct(baseline)	84.90	61.28	63.20	78.40
Qwen3-VL-235B-A22B-Instruct(step)	83.66	62.17	74.58	76.16
Qwen3-VL-235B-A22B-Instruct(Dynamic Cheatsheet)	83.25	60.06	70.62	75.49
Qwen3-VL-235B-A22B-Instruct(ViLoMem)	84.98	62.83	75.21	78.31
Qwen3-VL-235B-A22B-Instruct(Ours)	**85.10 ± 0.07**	**65.46 ± 0.12**	**75.55 ± 0.05**	**79.30 ± 0.04**
Qwen3-VL-8B-Instruct(baseline)	77.20	48.13	61.10	70.91
Qwen3-VL-8B-Instruct(step)	77.80	48.35	73.08	70.22
Qwen3-VL-8B-Instruct(Dynamic Cheatsheet)	74.92	46.81	68.39	69.12
Qwen3-VL-8B-Instruct(ViLoMem)	77.87	49.34	73.19	72.13
Qwen3-VL-8B-Instruct(Ours)	**79.10 ± 0.06**	**53.55 ± 0.07**	**73.59 ± 0.10**	**72.81 ± 0.06**
GPT-4.1(baseline)	70.40	46.12	58.50	69.80
GPT-4.1(step)	74.27	47.47	74.44	70.43
GPT-4.1(Dynamic Cheatsheet)	73.87	48.68	75.30	69.68
GPT-4.1(ViLoMem)	76.88	53.95	75.29	72.43
GPT-4.1(Ours)	**77.89 ± 0.06**	**55.28 ± 0.10**	**76.10 ± 0.07**	**73.07 ± 0.06**

The cross-model study uses memory constructed by the 8B backbone and evaluated by the 235B backbone without changing retrieval or prompting. This separates backbone-specific gains from the transferability of the induced memory.

We use VLMEvalKit (Duan et al., 2024) as the primary evaluation framework and apply Math-Verify plus an LLM-as-a-judge protocol with Qwen3-8B-Instruct when automatic matching is ambiguous. Full prompting templates and additional implementation details are provided in Appendix A.3.

4.2 Main Results

The primary question is whether TraceMem improves pass@1 accuracy beyond both standard prompting and prior memory-based reasoning. Table 1 shows that the answer is yes on both Qwen3-VL backbones. On Qwen3-VL-235B-A22B-Instruct, TraceMem improves MathVista, MathVision, and HallusionBench over the listed baselines; on Qwen3-VL-8B-Instruct, it improves all four benchmarks over the strongest listed baseline. GPT-4.1 with TraceMem improves all four benchmarks over its strongest listed baseline. The gain is consistent across the evaluated model variants.

Looking at individual benchmarks, the strongest and most consistent gains appear on MathVision, where TraceMem delivers the clearest margin over both the no-memory baseline and ViLoMem on both model sizes. We also observe strong improvements on HallusionBench relative to direct single-pass inference. These results suggest that the proposed memory is particularly helpful for tasks that require repeated visual checking, structured reasoning steps, or explicit avoidance of previously observed failure patterns.

4.3 Ablation Study

Table 2 analyzes which parts of the framework are responsible for the final gain. All variants use the same backbone and evaluation setup, while removing one component at a time. The full model achieves the best score of 79.10, outperforming both the baseline and the step baseline. This shows that the improvement does not come from explicit step-by-step prompting alone.

Among the ablated variants, removing success memory causes the largest drop, indicating that reusable procedure-level knowledge extracted from successful trajectories is the most important contributor on MathVista. Removing failure traces also hurts clearly. In this variant, when the first-pass answer is wrong, the framework does not launch the additional trajectory-sampling stage and therefore cannot induce new memory from contrasting failed attempts. This result indicates that failure-side information is not redundant: learning what to avoid contributes beyond storing only successful procedures.

Table 2: Component ablation of TraceMem on four multimodal reasoning benchmarks with Qwen3-VL-8B. Baseline and Step use the same backbone without memory; each remaining row removes one component while keeping the retrieval, verification, and evaluation settings fixed. Higher is better.

Method	MathVista	MathVision	HallusionBench	MMStar
Qwen3-VL-8B-Instruct(baseline)	77.20	48.13	61.10	70.91
Qwen3-VL-8B-Instruct(step)	77.80	48.35	73.08	70.22
TraceMem (w/o lifecycle)	78.70	53.20	72.55	72.61
TraceMem (w/o structured memory)	78.40	53.22	72.66	72.50
TraceMem (w/o failure traces)	77.90	52.97	73.29	72.17
TraceMem (w/o success memory)	77.40	52.86	72.79	72.05
TraceMem (w/o failure memory)	78.50	53.18	73.00	72.53
TraceMem (Ours)	**79.10**	**53.55**	**73.59**	**72.81**

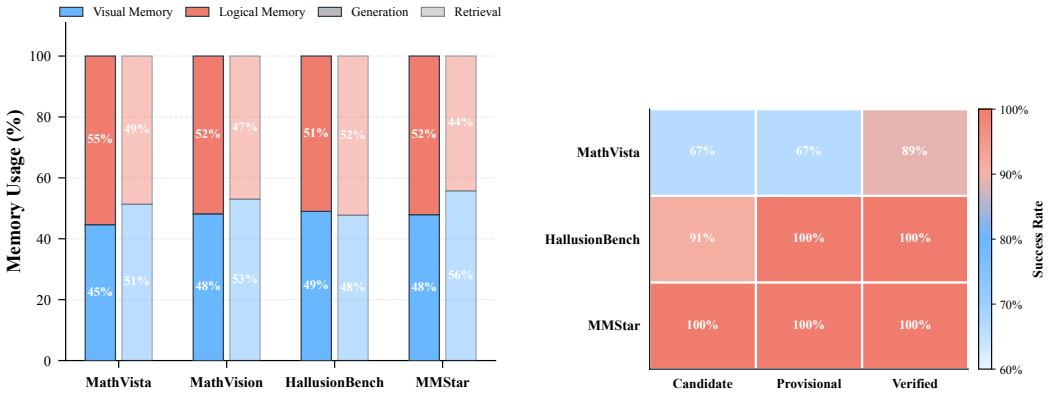

Figure 2: Complementary analyses of TraceMem memory quality. (a) Distribution of visual and logical memory in generated banks and retrieved guidance across benchmarks. (b) Post-retrieval success rate by memory status on MathVista, HallusionBench, and MMStar.

The remaining ablations show that the other design choices also matter, even if their individual effects are smaller. Removing lifecycle updates or the structured memory representation both lowers performance. In the *w/o structured memory* variant, we do not remove memory entirely; instead, we replace structured memory units with free-form text memory distilled from the same trajectories. The drop of this variant supports the role of utility-aware memory maintenance and explicit trigger/procedure/avoidance representation. Removing failure memory also underperforms the full model, suggesting that success and failure memory are complementary rather than interchangeable.

4.4 Memory Modality Analysis

Figure 2(a) analyzes memory usage at the modality level by grouping the four banks into visual and logical memory. Across benchmarks, the generated memory pool remains relatively balanced, while the retrieved memory distribution varies more by task. In particular, MathVista and MMStar retrieve more visual memory, whereas HallusionBench remains closer to balanced. This pattern is consistent with the demands of the benchmarks: MathVista and MMStar more often require explicit grounding, measurement, or visual comparison, while HallusionBench benefits from balancing visual checks with logical verification. The result suggests that TraceMem maintains a broad multimodal memory pool and shifts retrieval toward the modality that is more useful for the current benchmark.

4.5 Retrieval Quality by Memory Status

Figure 2(b) shows that retrieved verified memory is generally associated with the strongest downstream success, while candidate memory is less stable. The pattern is especially clear on MathVista, where verified memory yields a noticeably higher post-retrieval success rate than candidate or provisional memory. This result supports the role of lifecycle-aware memory maintenance in prioritizing more reliable memory during retrieval. It also suggests that the status variable is not merely descriptive: as memory is repeatedly tested, the lifecycle mechanism increasingly separates consistently useful items from immature ones.

4.6 Case Study

Figure 3 provides qualitative case studies from MathVista to illustrate how TraceMem reuses stored experience during inference. Case 1 and Case 2 show positive memory retrieval, where the guidance provides useful procedures for age-gap reasoning and ruler-based measurement. Case 3 and Case 4 show negative memory retrieval, where the guidance helps avoid common mistakes in time reasoning and chart comparison. In both positive and negative cases, the key effect is not generic prompting, but the injection of a compact task-matched procedure or warning. Together, these examples show that retrieved memory acts as targeted task-specific guidance.

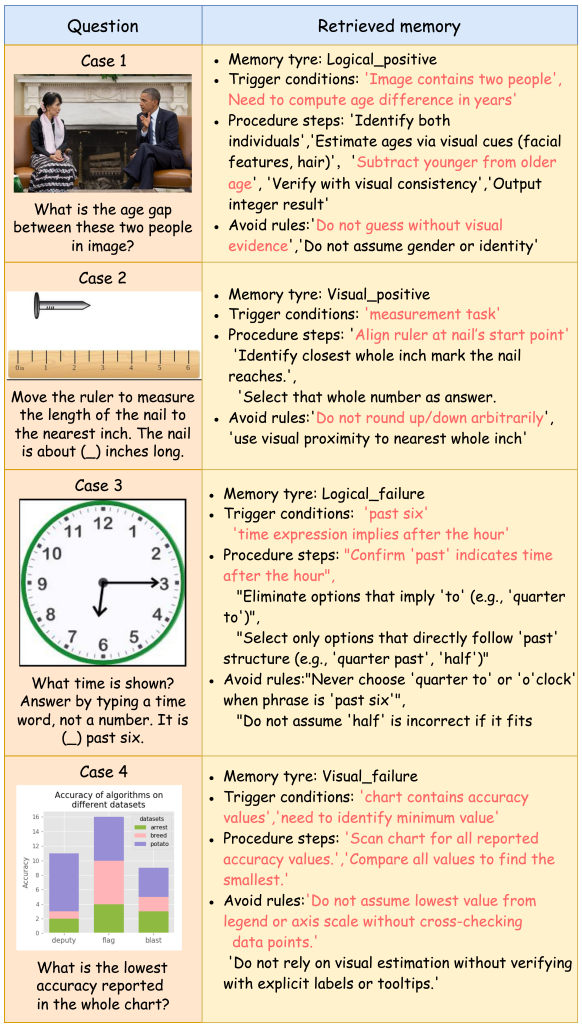

Figure 3: Representative MathVista case studies of logical-positive, visual-positive, logical-negative, and visual-negative memory.

4.7 Cross-Model Memory Transfer

We further test whether the induced memory can transfer across backbones. In this setting, the memory bank is constructed by Qwen3-VL-8B but reused by Qwen3-VL-235B at inference time, while the retrieval and reasoning procedure remains unchanged. We report results on MathVista, MathVision, HallusionBench, and MMStar.

Figure 4 shows that cross-model transfer improves all four benchmarks over the standard same-model TraceMem setting, with the largest average gains on MathVista and HallusionBench. This suggests that the induced memory is not tightly bound to the backbone that produced it and can transfer as compact guidance to a stronger model. The stored memory therefore captures task-level procedures and failure patterns that remain useful even when the consumer backbone changes.

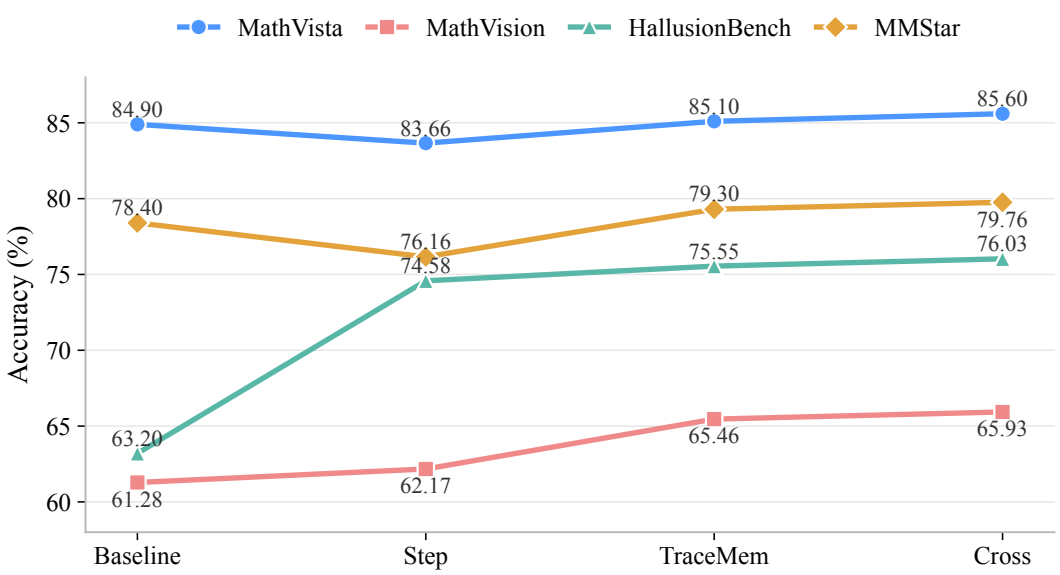

Figure 4: Cross-model memory transfer from Qwen3-VL-8B to Qwen3-VL-235B across four benchmarks. Each line traces one benchmark from direct inference through step prompting and same-model TraceMem to cross-model memory transfer. Cross-model transfer improves all four benchmarks over same-model TraceMem.

5 Conclusion

This paper studies how multimodal reasoners can accumulate reusable experience across episodes instead of solving every sample from scratch. We address this problem with a framework that converts successful and failed reasoning trajectories into persistent and actionable reasoning experience. The key insight is that test-time trajectories are useful not only for repairing the current prediction, but also for forming future-useful experience for later reasoning episodes. By organizing this induced experience into four complementary memory banks and refining it through lifecycle-aware utility updates, the framework turns test-time exploration into a reusable learning signal.

Our empirical study asks whether trajectory-induced structured memory improves pass@1 accuracy over strong direct and memory-based baselines. The current evidence shows consistent improvements across the evaluated benchmarks.

6 Limitations

The framework is memory-centric rather than purely inference-time, which introduces practical constraints. First, the lifecycle mechanism becomes most informative only when the system encounters enough repeated episodes for memory quality to be updated meaningfully; in short runs, the bank may remain too immature for the full design to realize its advantage. Second, memory quality depends on the abstraction quality of induced procedure and avoidance rules: a memory item can be semantically relevant at retrieval time yet still be operationally weak if the induced procedure is too vague or overly tied to a single instance.

The framework is also best matched to settings where correctness can be evaluated relatively clearly and future samples share partial structure with past ones. It may be less effective in open-ended tasks with ambiguous supervision or sparse reusable patterns. Finally, the current lifecycle update uses shared episode-level credit assignment for all retrieved memory items, which may blur which item actually caused a success or failure when multiple memories are injected together.

Reproducibility Statement

Implementation details, decoding hyperparameters, retrieval settings, memory lifecycle thresholds, and full prompt templates are provided in Appendix A.3 and Appendix A.4. The anonymous code repository linked in the abstract provides the implementation for review.

Ethics Statement

TraceMem is designed to improve multimodal reasoning by reusing structured experience from prior episodes, but such memory mechanisms also introduce risks. If incorrect or biased experience is stored and later retrieved, the system may propagate errors across examples rather than treating each case independently. This risk is especially relevant in high-stakes domains where wrong reasoning or misleading visual interpretation can have practical consequences. In addition, the framework may inherit biases or artifacts from the underlying models, benchmarks, and verification signals used during memory construction. For these reasons, memory-augmented reasoning systems should be used with appropriate monitoring and human oversight, particularly in settings where correctness, fairness, or safety is important. ChatGPT was used to polish language and improve readability during manuscript preparation. All content was carefully reviewed and revised by the authors, who take full responsibility for the final text.

AI Use Statement

We used generative AI assistance for the following purposes: language polishing and improving readability; retrieval and discovery of related work; research ideation and execution; drafting sections of the paper. These disclosures cover AI-assisted writing, retrieval and discovery, research ideation or execution, and draft preparation. We reviewed all AI-assisted work and take responsibility for the final content, including text, claims, and artifacts.

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, 2025. URL https://arxiv.org/abs/2511.21631.

Weihao Bo, Shan Zhang, Yanpeng Sun, Jingjing Wu, Qunyi Xie, Xiao Tan, Kunbin Chen, Wei He, Xiaofan Li, Na Zhao, Jingdong Wang, and Zechao Li. Agentic learner with grow-and-refine multimodal semantic memory, 2025. URL https://arxiv.org/abs/2511.21678.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models? In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 27056–27087. Curran Associates, Inc., 2024. doi: 10.52202/079017-0850. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/2f8ee6a3d766b426d2618e555b5aeb39-Paper-Conference.pdf.

Wenhu Chen, Hexiang Hu, Xi Chen, Pat Verga, and William Cohen. MuRAG: Multimodal retrieval-augmented generator for open question answering over images and text. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5558–5570, Abu Dhabi, United Arab Emirates, December

2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.375. URL https://aclanthology.org/2022.emnlp-main.375/.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. VLMEvalKit: An open-source toolkit for evaluating large multi-modality models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 11198–11201. ACM, 2024. doi: 10.1145/3664647.3685520. URL https://doi.org/10.1145/3664647.3685520.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, Dinesh Manocha, and Tianyi Zhou. Hallusionbench: An advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14375–14385, June 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Guan_HallusionBench_An_Advanced_Diagnostic_Suite_for_Entangled_Language_Hallucination_and_CVPR_2024_paper.html.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9590–9601, June 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Hu_Visual_Program_Distillation_Distilling_Tools_and_Programmatic_Reasoning_into_Vision-Language_CVPR_2024_paper.html.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6774–6786, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.528. URL https://aclanthology.org/2021.acl-long.528/.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KUNzEQMWU7.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. Docvqa: A dataset for vqa on document images. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 2199–2208. IEEE, January 2021. doi: 10.1109/wacv48630.2021.00225. URL http://dx.doi.org/10.1109/WACV48630.2021.00225.

Joon Sung Park, Joseph OBrien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST 23, pp. 1–22. ACM, October 2023. doi: 10.1145/3586183.3606763. URL http://dx.doi.org/10.1145/3586183.3606763.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Xin Su, Man Luo, Kris W Pan, Tien Pei Chou, Vasudev Lal, and Phillip Howard. Sk-vqa: Synthetic knowledge generation at scale for training context-augmented multimodal llms, 2024. URL https://arxiv.org/abs/2406.19593.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

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. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95095–95169. Curran Associates, Inc., 2024b. doi: 10.52202/079017-3014. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad0edc7d5fa1a783f063646968b7315b-Paper-Datasets_and_Benchmarks_Track.pdf.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

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, 2023. URL https://arxiv.org/abs/2303.11381.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=5Xc1ecxO1h.

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*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Michihiro Yasunaga, Armen Aghajanyan, Weijia Shi, Richard James, Jure Leskovec, Percy Liang, Mike Lewis, Luke Zettlemoyer, and Wen-Tau Yih. Retrieval-augmented multimodal language modeling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 39755–39769. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/yasunaga23a.html.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9556–9567, June 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Yue_MMMU_A_Massive_Multi-discipline_Multimodal_Understanding_and_Reasoning_Benchmark_for_CVPR_2024_paper.html.

A Appendix

A.1 Extended Related Work

MuRAG (Chen et al., 2022), RA-CM3 (Yasunaga et al., 2023), and SK-VQA (Su et al., 2024) show that retrieved external knowledge or stored context can improve multimodal inference, but the reused content is often attached to the current sample as evidence or case-level context rather than as explicit procedural memory. Persistent-memory agents, including Generative Agents (Park et al., 2023), Voyager (Wang et al., 2024a), and ViLoMem (Bo et al., 2025), accumulate experience across episodes. ViLoMem is the closest baseline: its logic bank stores textual reasoning guidelines, while its visual bank stores visual guidelines paired with source images. It attributes failures to either the visual or logical stream, produces an error type, analysis summary, and corrective guideline, and merges or inserts entries by text similarity following a grow-and-refine strategy. Visual retrieval first uses image-embedding similarity and then question-specific reranking, while logical memory uses text similarity over an analyzer-enriched query.

A.2 Extended Method Details

Structured memory fields. The trigger conditions specify when a memory item should activate. Procedure steps summarize reusable reasoning or observation actions. Avoidance rules record failure patterns to prevent. Task tags provide a compact semantic summary. Utility tracks historical effectiveness, and lifecycle status records whether an item is newly created, tentatively reliable, mature, or deprecated. The four banks separate success from failure and logical from visual memory, reducing retrieval interference between different kinds of guidance.

Retrieval rationale. The analyzer predicts task tags such as chart, reasoning, OCR, geometry, and counting, and constructs a query from the question. Bank-wise retrieval keeps the prompt compact and prevents irrelevant memory from dominating across modalities. High-utility and high-status memory is preferred over new or historically harmful entries.

Induction and credit assignment. The system extracts short step sequences from successful and failed trajectories, rewrites them into procedures or avoidance rules, and merges them into the corresponding bank when no sufficiently similar entry exists. When all sampled trajectories fail, only failure-side memory is added. Retrieved items currently receive shared episode-level feedback; this is practical but can obscure which individual memory caused an outcome. Item-level credit assignment is left for future work.

A.3 Implementation and Reproducibility Details

This section provides additional implementation details for the experimental setup.

Model deployment. For Qwen3-Embedding, we deploy the checkpoint with vLLM to enable efficient batched inference. For Qwen2.5-VL-Embedding, Qwen3-VL-8B-Instruct, and Qwen3-VL-235B-A22B-Instruct, we access the models through their official APIs.

Decoding hyperparameters. For all reasoning models, we use a temperature of 0.7 and a maximum generation length of 8,192 tokens. Within the memory pipeline, the question analyzer uses a temperature of 0.2 and a maximum generation length of 512 tokens; trajectory generation uses a temperature of 0.8 and a maximum generation length of 4,096 tokens; and memory induction uses a temperature of 0.2 and a maximum generation length of 1,024 tokens. After a failed first-pass answer, we sample 4 additional reasoning trajectories.

Prompting configurations. The baseline evaluation directly feeds each benchmark question to the model without additional prompting. The step configuration prepends a simple step-by-step system prompt before the question. In TraceMem, the full prompt templates for task analysis, step-by-step reasoning, trajectory generation, memory induction, and memory-conditioned second-pass reasoning are shown in Figures 5–11.

Evaluation protocol. We use VLMEvalKit (Duan et al., 2024) as the primary evaluation framework. When automatic matching fails or yields ambiguous results, we further apply Math-Verify together with an LLM-as-a-judge protocol to reduce sensitivity to output formatting. The judge model is Qwen3-8B-Instruct, used via API, and it determines whether a model response is semantically correct with respect to the reference answer. The same verification pipeline is used inside TraceMem for first-pass and trajectory-level verification. Concretely, the pipeline applies VLMEval-based matching when available, then answer parsing and Math-Verify, and finally falls back to an LLM judge when the earlier checks remain inconclusive.

Retrieval-score coefficients. The retrieval coefficients in Equation (1) are set to $\alpha = 0.75$ and $\beta = 0.10$. The remaining retrieval and lifecycle hyperparameters are as follows. Memory composition uses top-$k = 2$, with at most one positive and one negative memory selected per modality. The deduplication threshold for bank insertion is set to 0.92. Candidate, provisional, and verified memory are all retrievable, with status bonuses of 0.00, 0.03, and 0.06, respectively.

Trajectory sampling, generation, and memory-bank updates. Successful and failed trajectories are first compressed into short step sequences, and then generated bank-specific memory units. Positive banks focus on reusable procedures, while negative banks focus on avoidance rules. We allow at most one newly distilled memory unit per bank for each failed episode. Negative memory is added only when failure evidence is sufficiently consistent, specifically when at least 3 failed trajectories are observed and the failure ratio is at least 0.6. When a new memory unit is inserted, it is merged with an existing unit if their lexical similarity exceeds the deduplication threshold; otherwise, it is added as a new entry.

Lifecycle update rules. Each retrieved memory unit receives shared episode-level feedback. The utility update uses a hurt penalty of 0.5. A memory unit is promoted from *candidate* to *provisional* after at least 2 uses with positive utility, and from *provisional* to *verified* after at least 4 uses with utility at least 0.2. A memory unit is marked *deprecated* after at least 2 uses if its utility falls to -0.1 or below.

Hurt-penalty sensitivity. We select the hurt penalty λ on held-out calibration splits of MathVista and MathVision using Qwen3-VL-8B, while keeping the retrieval settings, lifecycle thresholds, trajectory count, and evaluation protocol fixed. We compare

Table 3: Sensitivity to the hurt penalty λ on MathVista and MathVision with Qwen3-VL-8B. All other settings are fixed, and each row is evaluated under the same memory-construction protocol.

λ	MathVista	MathVision
0.3	*77.89± 0.06*	*52.98± 0.10*
0.4	*78.45± 0.08*	*53.10± 0.09*
0.5 (default)	*79.10± 0.06*	*53.55± 0.07*
0.6	*78.90± 0.05*	*53.20± 0.08*
0.7	*78.60± 0.07*	*52.80± 0.09*

$\lambda \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$; the default value $\lambda = 0.5$ is used in all main experiments. Table 3 summarizes the selection grid and the resulting accuracy on both datasets.

A.4 Prompt Templates

This subsection collects the full prompt templates used in TraceMem. These figures are included for reproducibility and correspond to the prompting stages summarized in Section 3.

Task_Analyzer_Prompt

```
You are a task analyzer for multimodal reasoning.

 Given the question below, return JSON only with this schema:
 {
  "task_tags": ["tag1", "tag2"],
  "question_focus": "short phrase",
  "requires_visual_grounding": true,
  "requires_multistep_logic": true
 }

 Rules:
 - task_tags should be short lowercase tags.
 - Prefer tags such as chart_reasoning, ocr, geometry, counting,
 multi_step_logic, math_reasoning, real_world_grounding,
table_reasoning,
 comparison.
 - Return valid JSON only.

 Question:
 {question}
```

Figure 5: Prompt template for task analysis in TraceMem.

Step-by-Step Reasoning Prompt

```
Objective:
    Solve the given problem using a step by step process.

 Expected Output Structure:
 Step 1:
 Step 2:
 ...
 Step n: Final Answer: \boxed{answer}

 Question:
```

Figure 6: Prompt template for first-pass step-by-step reasoning.

Trajectory Generation Prompt

```
Generate diverse candidate reasoning trajectories for the given
multimodal
 problem.
 Return JSON only with this schema:
 {
  "trajectories": [
   {"trajectory_id": 1, "prediction": "...", "reasoning": "..."}
  ]
 }

 Requirements:
 - Return exactly {num_candidates} trajectories.
 - Each trajectory must differ materially in evidence usage or
reasoning path.
 - Each prediction should include a final answer.

 Problem:
 {problem}
```

Figure 7: Prompt template for trajectory generation after an initial failure.

Logical Success Memory Prompt

```
You are an expert at extracting reusable LOGICAL success
strategies.

 A model solved the following problem correctly multiple times.
 Your job is to summarize the common logical reasoning
strategies that
 contributed to success.

 <QUESTION>
 {question}
 </QUESTION>

 <SUCCESS_GROUP>
 {success_group}
 </SUCCESS_GROUP>

 Instructions:
 1. Focus only on logical reasoning patterns, not visual reading
behavior.
 2. Identify what successful answers consistently did right.
 3. Abstract the result into short, reusable rules for solving
similar
 problems.
 4. Each rule should be actionable and generalizable.

 Return JSON only:
 {
  "memories": [
   "rule 1",
   "rule 2"
  ]
 }
```

Figure 8: Prompt template for logical success-memory induction.

Visual Success Memory Prompt

```
You are an expert at extracting reusable VISUAL success
strategies.

 A model solved the following multimodal problem correctly
multiple times.
 Your job is to summarize the common visual interpretation
strategies that
 contributed to success.

 <QUESTION>
 {question}
 </QUESTION>

 <IMAGE_INFO>
 {image_info}
 </IMAGE_INFO>

 <SUCCESS_GROUP>
 {success_group}
 </SUCCESS_GROUP>

 Instructions:
 1. Focus only on visual grounding and image interpretation
behaviors.
 2. Identify what successful answers consistently did right when
reading the
 image.
 3. Abstract the result into short, reusable visual rules.
 4. Each rule should be actionable and generalizable.

 Return JSON only:
 {
  "memories": [
   "rule 1",
   "rule 2"
  ]
 }
```

Figure 9: Prompt template for visual success-memory induction.

Logical Failure Memory Prompt

```
You are an expert logical error-pattern analyst.

 A model attempted the same problem multiple times and
produced poor answers.
 Your job is to summarize the COMMON logical mistakes
across these failed
 attempts.

 <QUESTION>
 {question}
 </QUESTION>

 <FAILED_GROUP>
 {failure_group}
 </FAILED_GROUP>

 Instructions:
 1. Focus only on logical failures: wrong reasoning patterns,
formula misuse,
 invalid assumptions, skipped constraints, calculation mistakes.
 2. Find the common root causes across failed attempts.
 3. Convert them into short, generalizable "avoidance rules".
 4. Each rule should clearly describe what NOT to do.

 Return JSON only:
 {
  "memories": [
    "avoidance rule 1",
    "avoidance rule 2"
  ]
 }
```

Figure 10: Prompt template for logical-failure-memory induction.

Visual Failure Memory Prompt

```
You are an expert visual error-pattern analyst.

 A model attempted the same multimodal problem multiple
times and produced poor
 answers.
 Your job is to summarize the COMMON visual mistakes across
these failed
 attempts.

 <QUESTION>
 {question}
 </QUESTION>

 <IMAGE_INFO>
 {image_info}
 </IMAGE_INFO>

 <FAILED_GROUP>
 {failure_group}
 </FAILED_GROUP>

 Instructions:
 1. Focus only on visual failures: misreading numbers, labels,
regions, object
 identity, chart scale, relative positions, or other visual
grounding issues.
 2. Find the common root causes across failed attempts.
 3. Convert them into short, generalizable "avoidance rules".
 4. Each rule should clearly describe what NOT to do.

 Return JSON only:
 {
  "memories": [
   "avoidance rule 1",
   "avoidance rule 2"
  ]
 }
```

Figure 11: Prompt template for visual-failure-memory induction.