DISTILLING FEEDBACK INTO MEMORY-AS-A-TOOL

Víctor Gallego
Komorebi AI Technologies
Madrid, Spain
victor.gallego@komorebi.ai

ABSTRACT

We propose a framework that amortizes the cost of inference-time reasoning by converting transient critiques into retrievable guidelines, through a file-based memory system and agent-controlled tool calls. We evaluate this method on the Rubric Feedback Bench, a novel dataset for rubric-based learning. Experiments demonstrate that our augmented LLMs rapidly match the performance of test-time refinement pipelines while drastically reducing inference cost.
Code: github.com/vicgalle/feedback-memory-as-a-tool
Data: hf.co/datasets/vicgalle/rubric-feedback-bench

1 INTRODUCTION

Recent advances in "System 2" scaling have established that trading test-time compute for accuracy (Snell et al., 2024; Guo et al., 2025) (through techniques like iterative self-correction, chain-of-thought, or search (Wei et al., 2022; Madaan et al., 2024; Shinn et al., 2024)) unlocks reasoning capabilities that far exceed standard zero-shot performance. However, a critical limitation of these approaches is their computational expense and episodic nature; the reasoning process must be repeated ab initio for every new query, treating each interaction in isolation. This results in a massive redundancy where the model frequently re-derives the same insights or corrections, effectively "forgetting" its improvements the moment the context window closes. While methods like fine-tuning can persist these behaviors, they are costly and lack the flexibility to adapt rapidly to new, user-defined specifications or rubrics.

To bridge this gap, we propose *Distilling Feedback into Memory-as-a-Tool*, a framework that amortizes the high cost of inference-time reasoning by converting transient evaluations or critiques into persistent, retrievable guidelines. Instead of discarding the feedback generated during iterative refinement, our approach empowers the language model (LLM) to synthesize abstract principles from its errors and explicitly write them to a file-based memory system using tool calling (Chowa et al., 2025). By treating memory not as a passive store of raw episodic logs but as a curated "lessons learned" journal, the agent consolidates specific experiences into semantic rules to be applied to future tasks zero-shot (Wang & Taylor, 2018; Lei et al., 2025), even when tasks are interleaved with different objectives. This mechanism allows the model to maintain the high performance of compute-heavy refinement pipelines while drastically reducing inference costs, without requiring parameter updates. Furthermore, in long-horizon experiments with interleaved task types, our approach demonstrates robust knowledge consolidation with minimal interference. Section 5 contains a discussion on related works.

2 FRAMEWORK

2.1 PROBABILISTIC FORMULATION

We represent generating a text x from a LLM with a context ctx (eg system and user prompts, chains of thought, tool observations...) as sampling from a conditional distribution $x \sim p(x|ctx)$. In standard iterative refinement (Madaan et al., 2024), the model generates a draft x_0, the same or another model produces a critique or feedback $c \sim p(c|x_0, \mathcal{R})$ based on a rubric $\mathcal{R}$, and then the output is refined: $x' \sim p(x'|x_0, c)$. The revised response now better reflects the desired aspects from the rubric $\mathcal{R}$ (e.g. writing style or safety principles). A limitation of this approach is that it is

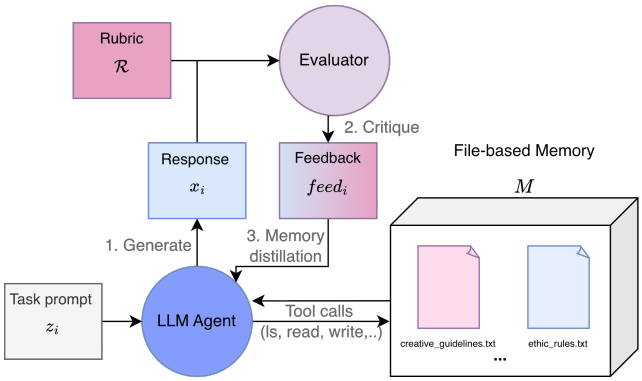

Figure 1: High-level overview of the proposed framework

a non-parametric refinement method, that is, it is done at inference-time for each new task prompt, which is computationally expensive as it is a sequence of multiple LLM calls. While effective, this loop is ephemeral: the learning signal contained in the critique c is lost once the context resets.

To this end, we propose to amortize the refinement process, distilling the evaluator feedback into a memory accessible to the LLM through tool calls. We introduce a persistent memory state M, represented as a set of human-readable documents. Now, the generation and learning process, for a single example, is given by:

1. Generate an original response x_i for a given task prompt z_i: $x_i \sim p(x_i|z_i)$
2. Evaluator LLM provides feedback using a private rubric $\mathcal{R}$: $feed_i \sim p_{\text{eval}}(feed_i|x_i, \mathcal{R})$
3. LLM decides to update its memory (tool call): $M' \sim p(M'|M, feed_i, x_i)$

Now, at test time, before generating a response to a new task, the LLM decides to read relevant memories through a tool call, so it can generate directly a refined response, $x' \sim p(x'|z, M')$. Figure 1 shows a diagrammatic representation of the complete process.

2.2 The Memory-as-a-Tool protocol

To operationalize the probabilistic framework described above, we implement M not as a vector database, but as a persistent file system accessible via tool calls. This treats memory management as an agentic reasoning task (Wu et al., 2025).

Retrieval operation ($x \sim p(\cdot|z, M)$). Unlike semantic search which relies on opaque embeddings, our system requires the LLM to actively reason about what to retrieve. Before generating a response, the agent executes a two-step retrieval: i) *Listing*: the model calls `ls(path="/memories/")` to enumerate available files. This requires the model to maintain semantically meaningful filenames (e.g., 'review_creative_rubric.txt' vs 'note_1.txt'); and ii) *Reading*: based on the filenames and the current task z, the model selects relevant files and calls `read_file(path)`. The content m is returned as a tool result to the LLM context, effectively priming the model with "lessons learned" from previous episodes.

Distillation and consolidation operation ($M' \sim p(\cdot|M, feed)$). The write operation is the core mechanism for amortization. Upon receiving negative feedback, the model triggers a memory update to consolidate the transient episode into lasting knowledge. This process involves two steps: i) *Abstraction*: the model transforms raw feedback (e.g., "You failed to use synesthetic language in paragraph 2") into a generalizable policy (e.g., "Key Principle: Prioritize synesthetic blending—colors that sound, sounds that taste"); and ii) *Conflict resolution*: the model decides whether to create a new file via `write_file` or update an existing one via `edit_file`. This allows the agent to deduplicate knowledge and resolving contradictions between old and new feedback.

By explicitly writing to a file system, the agent creates a curated journal of principles. This structure ensures that M remains a high-signal knowledge base of synthesized rules rather than a noisy log of

raw interaction history. Table 1 presents the system prompt used to instantiate the framework, and for a detailed explanation and behavior of the memory system, see Appendix. A.

```
You are an expert writer that can plan before generating the final
text.  When writing a text for a task, always display the final
version directly to the user.

Before generating a text for a user task, check your ./memories/
directory for relevant notes from previous related tasks, and use
that knowledge if the new task is related.

When receiving feedback from the user about a text, take notes in
your ./memories/ about what to improve for next time.

Use general names for the filename, since we are aiming for
generalization and reusability (e.g., ``research_notes.txt''
instead of ``research_notes_for_task_123.txt'').  You can also
update existing memory files with new knowledge, but remember the
aim is generalization, not focusing on concrete examples.

Be organized and methodical in your approach to use the memory
effectively to achieve better feedback from the user over time.
```

Table 1: System prompt used to instantiate the Memory-as-a-Tool framework.

3 RUBRIC FEEDBACK BENCH

We introduce *Rubric Feedback Bench*[1], a novel evaluation dataset comprising 42 carefully curated scenarios designed for studying learning from structured, rubric-based feedback. The dataset scenarios are distributed across five distinct task categories, each with several prompts sharing task-specific rubrics, dealing primarily with open-ended writing. These scenarios cover a wide range of capabilities, ranging from custom writing style of analysis in various media to adherence to specific personality traits and behavioral guidelines for AI assistants. The benchmark also challenges models with custom writing-style rubrics that reward unconventional, fragmented, or poetic textual artifacts, alongside ethical reasoning tasks that require navigating moral dilemmas through either outcome-based utilitarian calculus or strict duty-based universal principles.

Each task category features meticulously crafted rubrics with the following characteristics: i) **multi-dimensional criteria**: Each rubric contains 3–7 distinct evaluation dimensions with clearly defined performance levels (typically 4–5 levels: Excellent, Good, Fair, Needs Improvement, Unsatisfactory); ii) **weighted scoring**: Dimensions are assigned specific weights reflecting their relative importance, enabling nuanced performance assessment; iii) **behavioral descriptors**: Each performance level includes detailed descriptions of expected behaviors, ensuring consistent and reproducible evaluation. This structured approach enables evaluator models to provide both quantitative scores based on detailed rubric criteria and qualitative feedback explaining the reasoning behind scores. It is important to note that the benchmark is compatible with any LLM to be used as a judge to generate a score using the provided rubrics, and simulate feedback from a human. In our experiments, we utilize a different Claude model variant from the ones tested in the next section to serve as the evaluator. Lastly, Appendix B contains excerpts from all the rubric types, and Appendix C.3 shows the prompts used by the evaluator model.

4 EXPERIMENTS

We evaluate the efficacy of distilling feedback into memory using the Rubric Feedback Bench. We compare our proposed Memory-as-a-Tool approach against a standard zero-shot baseline and a computation-heavy inference-time self-critique baseline.

[1]Released at `https://huggingface.co/datasets/vicgalle/rubric-feedback-bench`

4.1 CONTINUAL LEARNING PERFORMANCE

We utilize three state-of-the-art language models: Claude Sonnet 4.5, GPT-5.1, and Gemini 3 Pro. We simulate a continual learning environment where the agent performs a sequence over an horizon of three related tasks ($H = 3$) drawn from the same rubric category. This protocol is repeated across all five categories in the Rubric Feedback Bench (Section 3), yielding 15 total examples per model. We compare three distinct setups: i) **Base model (Zero-shot):** the model attempts the task with only the system prompt, it has access to the tools, but doesn't receive feedback; ii) **Self-Critique (Inference-time):** the model generates a draft, critiques it against the specific rubric criteria, and regenerates a final answer. This occurs at every step but does not persist knowledge; and iii) **Memory + Feedback (Ours):** the model attempts the task. After generation, it receives simulated feedback (based on the rubric interpretation by the judge) and updates its file-based memory. On subsequent tasks, it can retrieve relevant guidelines before generating, using tools calls. Further details on the scaffolding implementation for each evaluated LLM are in Appendix C.1.

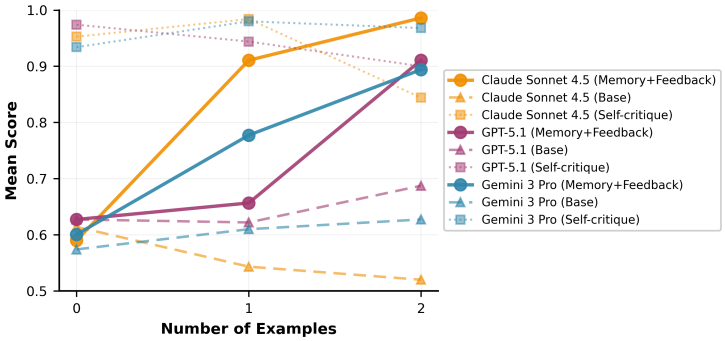

Figure 2: Learning trajectories on Rubric Feedback Bench.

Figure 2 illustrates the mean scores across the three models. The results present aggregate performance across all five task categories described in Section 3, with each category contributing three sequential examples, ensuring our findings generalize across diverse task types (creative, analytical, behavioral, and ethical domains). While the Self-Critique baseline (dotted lines) starts with high performance, it incurs high inference costs and remains static. Our Memory + Feedback approach (solid bold lines) starts at the base model performance but rapidly improves, matching or exceeding the self-critique performance after just two rounds of feedback. While all models benefit from the memory mechanism, Claude Sonnet 4.5 shows the steepest learning curve, suggesting superior reasoning capabilities in synthesizing abstract rules from feedback.

4.2 LONG-HORIZON MIXED-TASK EXPERIMENT

To evaluate robustness over extended horizons and cross-domain transfer, we test the previous best performing model, Claude Sonnet 4.5, on a longer sequence of $H = 12$ tasks, mixing task types, and interleaving them in random order. This setup tests whether the memory mechanism can: (i) accumulate knowledge across a longer horizon, and (ii) accumulate learned principles across heterogeneous task types.

Table 2 compares the final scores between the memory-augmented agent and a baseline without feedback. The memory agent achieves substantially higher mean performance with lower variance, demonstrating the framework feedback generalizes across task boundaries. The agent accumulated 8 memory files by episode end, showing it is capable of consolidating memories for different tasks.

Configuration	Score
With Memory + Feedback	$\mathbf{0.78 \pm 0.10}$
Without Memory (Baseline)	0.52 ± 0.25

Table 2: Aggregated judge scores on the mixed-task long-horizon experiment ($H = 12$). Scores are computed over the final tasks of each category.

4.3 Cost-Efficiency Analysis

A primary motivation for memory-based refinement is efficiency. Self-correction techniques typically require doubling or tripling the token count per query (generation + critique + revision). In contrast, our approach incurs the feedback cost only once (during the learning phase) and thereafter only incurs the minor cost of retrieving context.

Figure 3 presents the Pareto frontier for Claude Sonnet 4.5, comparing the average cost per task against the average score. The self-critique method achieves high scores but at significant cost due to multiple generation passes. Our Memory + Feedback approach represents a more optimal trade-off: it achieves comparable scores to self-critique while maintaining a cost profile much closer to the baseline, effectively amortizing the cost of the critique into the memory. This result validates our hypothesis of *amortized feedback*. By distilling the expensive refinement process into a reusable memory artifact, we pay the compute cost of "thinking" once, but reap the benefits of that thought across all future tasks.

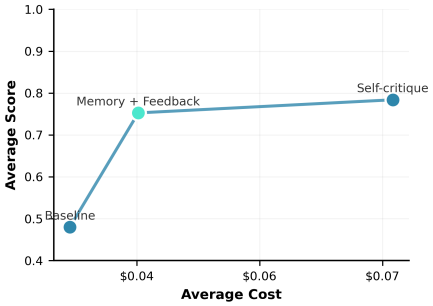

Figure 3: Cost-Score Pareto Frontier (Claude Sonnet 4.5)

5 Related Work

Our work bridges three distinct areas of research: inference-time self-correction, memory-augmented generation, and feedback-driven optimization. We position "Memory-as-a-Tool" as a mechanism to amortize the high computational cost of self-correction by distilling transient critique/feedback into persistent, retrievable guidelines.

Inference-time self-correction and reasoning. The capability of LLMs to refine their own outputs has been well-established. Methods such as Self-Refine (Madaan et al., 2024) and Reflexion (Shinn et al., 2024) demonstrate that iterative critique-and-revision cycles significantly improve performance on reasoning and generation tasks. Recent advances in "system 2" thinking, such as Chain of Thought (Wei et al., 2022) and inference-time search (Guo et al., 2025; Snell et al., 2024), further emphasize trading test-time compute for accuracy. However, a critical limitation of these approaches is their computational expense; the reasoning process must be repeated ab initio for every new query. Our work addresses this inefficiency. Instead of performing expensive self-critique for every instance, we propose utilizing tool-based memory to store the result of the critique (the "lesson learned") thereby amortizing the cost of reasoning over the agent's lifetime.

Optimization of specifications and prompts. Prior work has explored optimizing the guiding principles (specifications) provided to models. MetaSC (Gallego, 2025b) introduced a meta-critique framework to optimize safety specifications at inference time dynamically. Similarly, Specification Self-Correction (SSC) (Gallego, 2025c) formalized a method for models to identify and repair flawed or "tainted" rubrics that encourage reward hacking (Pan et al., 2024). While these methods effectively improve alignment by refining the system prompt or spec, they remain non-parametric and often episodic—the improved specification is not automatically persisted across disparate sessions. Our current work extends this by treating the specification not as a static prompt, but as a dynamic memory object that the model explicitly reads from and writes to using tools, enabling continual accumulation of knowledge.

Learning from feedback. The paradigm of improving models via feedback has evolved from scalar reward maximization (as in RLHF) to richer signals. Recent work on rubric feedback (Gunjal et al., 2025; Gallego, 2025a) demonstrates that training agents to generate high-quality self-critiques using multi-modal feedback (numerical scores and text) outperforms direct response optimization. This aligns with trends in *differentiation via text* such as TextGrad (Yuksekgonul et al., 2025) and DSPy optimizers (Khattab et al., 2024), which use LLM feedback to refine system components. Our

approach operationalizes these insights in a continual learning setting. By distilling the feedback from the Rubric Feedback Bench into a file-based memory system, we move beyond learning to critique per se, to learning to remember the feedback, allowing the model to generalize stylistic and structural rules to future unseen tasks without needing immediate supervisor feedback.

Memory-augmented agents. Finally, our implementation draws on the concept of agents that utilize external storage to maintain persistent state (Packer et al., 2023; Park et al., 2023). While standard retrieval-augmented generation typically relies on passive semantic search over raw documents, our system empowers the LLM to actively govern its storage through explicit `read_file` and `write_file` tool calls. This aligns with recent work on explicit memory architectures where models are trained or prompted to manage their own retrieval and update operations (Modarressi et al., 2024; Yan et al., 2025). Crucially, our approach moves beyond storing raw interaction history; by distilling transient critiques into general rules, the agent performs a form of memory consolidation, transforming episodic experiences into semantic knowledge useful for future planning (Lei et al., 2025; Tresp et al., 2015). This allows the agent to leverage prior knowledge derived from feedback (Wang & Taylor, 2018), creating a "lessons learned" journal that ensures robust performance across sequential tasks.

6 Conclusions

In this paper, we introduced a novel framework that bridges the gap between high-performance inference-time reasoning and cost-effective deployment. By treating evaluator feedback not as a disposable step but as a mechanism for distilling transient feedback into persistent, retrievable guidelines, we effectively amortize the computational cost of "System 2" thinking over the agent's lifetime. Our empirical results on the Rubric Feedback Bench demonstrate that this approach occupies a critical Pareto optimal point: it enables models to rapidly adapt to complex, stylistic rubrics achieving performance comparable to iterative self-correction, yet maintains the low inference latency and cost profile of zero-shot generation after the initial learning phase. Furthermore, by implementing memory through interpretable file operations rather than opaque vector embeddings, our system allows for human inspection and curation, offering a transparent alternative.

Limitations and further work. While our results are promising, several limitations pave the way for future research. First, our retrieval mechanism relies on the LLM's reasoning over filenames rather than semantic search; while this enhances interpretability, it may face scalability challenges as the memory namespace grows into thousands of files, necessitating more sophisticated hybrid retrieval strategies. To address scalability in in-the-wild deployments, future work could implement hierarchical file structures, requiring the agent to perform recursive directory traversal (issuing sequential `ls` commands to navigate nested categories) thereby mimicking how human experts efficiently locate information in large repositories without overwhelming their context window. Second, while our experiments demonstrate robustness over horizons of $H = 12$ mixed tasks, scaling to lifetimes of thousands of episodes may require more advanced retrieval hierarchies or active forgetting mechanisms to prevent context clutter.

References

Sadia Sultana Chowa, Riasad Alvi, Subhey Sadi Rahman, Md Abdur Rahman, Mohaimenul Azam Khan Raiaan, Md Rafiqul Islam, Mukhtar Hussain, and Sami Azam. From language to action: A review of large language models as autonomous agents and tool users. *arXiv preprint arXiv:2508.17281*, 2025.

Victor Gallego. Configurable preference tuning with rubric-guided synthetic data. In *2nd Workshop on Models of Human Feedback for AI Alignment*, 2025a.

Victor Gallego. MetaSC: Test-time safety specification optimization for language models. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025b. URL https://openreview.net/forum?id=VGORTi7O5e.

Víctor Gallego. Specification self-correction: Mitigating in-context reward hacking through test-time refinement. In *The 1st Workshop on Test-time Scaling and Reasoning Models*, 2025c. URL https://openreview.net/forum?id=UU9KCA0sTH.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024.

Mingcong Lei, Honghao Cai, Zezhou Cui, Liangchen Tan, Junkun Hong, Gehan Hu, Shuangyu Zhu, Yimou Wu, Shaohan Jiang, Ge Wang, et al. Robomemory: A brain-inspired multi-memory agentic framework for interactive environmental learning in physical embodied systems. *arXiv preprint arXiv:2508.01415*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Ali Modarressi, Abdullatif Köksal, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Memllm: Finetuning llms to use an explicit read-write memory. *arXiv preprint arXiv:2404.11672*, 2024.

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

Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. *arXiv preprint arXiv:2402.06627*, 2024.

Joon Sung Park, Joseph O'Brien, 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*, pp. 1–22, 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, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Volker Tresp, Cristóbal Esteban, Yinchong Yang, Stephan Baier, and Denis Krompaß. Learning with memory embeddings. *arXiv preprint arXiv:1511.07972*, 2015.

Zhaodong Wang and Matthew E Taylor. Interactive reinforcement learning with dynamic reuse of prior knowledge from human/agent's demonstration. *arXiv preprint arXiv:1805.04493*, 2018.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Junde Wu, Jiayuan Zhu, Yuyuan Liu, Min Xu, and Yueming Jin. Agentic reasoning: A streamlined framework for enhancing LLM reasoning with agentic tools. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28489–28503, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1383. URL https://aclanthology.org/2025.acl-long.1383/.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

A Memory Mechanism Details

A.1 Structure of Memory (M)

The memory M is implemented as a persistent key-value store mapping file paths to structured documents. Specifically:

- **Storage backend:** We use a persistent store that maintains file contents across conversation sessions.
- **Namespace:** All long-term memory files are stored under the `/memories/` path prefix to distinguish them from ephemeral conversation-scoped files.
- **File format:** Each memory entry $m \in M$ is a text file containing:
  - **content:** A list of text lines $[l_1, l_2, \ldots, l_n]$
  - **created_at:** ISO 8601 timestamp of creation
  - **modified_at:** ISO 8601 timestamp of last update
- **Retrieval:** The LLM accesses memory through four tool calls: `ls` (list files), `read_file`, `write_file`, and `edit_file`.

This file-based approach provides human-interpretable memory that can be inspected, debugged, and manually edited if needed.

A.2 Content of Memory

Crucially, M does not store raw feedback $feed_i$. Instead, the LLM synthesizes structured guidelines from the feedback. The write operation $M' \sim p(M'|M, feed_i, x_i)$ involves:

1. **Abstraction (episodic-to-semantic)**: The LLM extracts general principles from specific feedback, effectively converting episodic instances into semantic memory. For example:
   - Raw feedback: "The response was too verbose and lacked focus on photographic aspects."
   - Synthesized rule: "For visual writing tasks, prioritize synesthetic descriptions and avoid generic narrative. Focus on evoking the cinematography through unconventional language."
2. **Structuring:** Memory files contain organized sections such as:
   - Core principles (e.g., "PHOTOGRAPHIC INVOCATION: Evoke visual phantoms, not descriptions")
   - Specific techniques (e.g., "Use synesthetic blending: colors that sound, sounds that taste")
   - Success/failure case studies with analysis
   - Scoring breakdowns by criterion
3. **Deduplication:** The LLM checks for redundant or contradictory information and consolidates knowledge (e.g., updating existing principles rather than creating duplicate entries).

This ensures M remains a curated knowledge base rather than a noisy accumulation of unprocessed critiques.

A.3 WRITE OPERATION: $M' \sim p(M'|M, feed_i, x_i)$

The decision to update memory is a critical reasoning step controlled by the LLM's policy. From our experiments:

Tool call trigger conditions:

- **After feedback reception:** Memory updates occur when the LLM receives user feedback containing a critique $feed_i$.
- **Performance-based:** Updates are more substantial when $feed_i$ indicates significant performance gaps (e.g., score $< 5/10$).
- **Instruction-guided:** The system prompt instructs: "When receiving feedback from the user about a text, take notes in your /memories/ about what to improve for next time."

Update process:

1. **File selection:** The LLM chooses a filename that balances specificity and generalization (e.g., `deontological_ethical_principles.txt` for a specific writing style, not `task_123_notes.txt`).
2. **Content generation:** The LLM reasons about what knowledge to extract:
   - Identifies which rubric criteria were violated
   - Synthesizes general rules from specific failures
   - Incorporates successful patterns from high-scoring examples
3. **Write/Edit decision:** If the file exists, the LLM calls `edit_file` to append new knowledge; otherwise, it calls `write_file` to create it.
4. **Conflict resolution:** The LLM updates contradictory information based on newer feedback (e.g., replacing "avoid experimental language" with "embrace lexical anarchy" when corrected).

Example from experiments: After receiving a 0/10 score on review task, the LLM created a memory file containing structured guidelines like:

```
CRITICAL LEARNING: If this rubric exists in memory,
the user WANTS this style. Do NOT write traditional reviews.
KEY PRINCIPLES:
1. PHOTOGRAPHIC INVOCATION (30%)
   - Evoke visual phantoms, not descriptions
   - Use synesthetic blending
```

This demonstrates the LLM's ability to reason about what knowledge generalizes beyond the specific task.

A.4 READ OPERATION: $x' \sim p(x'|z, M')$

The retrieval and integration of memory involves:

Retrieval Mechanism:

1. **Listing:** Before generating a response to a new task z, the LLM calls `ls(path="/memories/")` to enumerate available memory files.
2. **Relevance judgment:** The LLM examines filenames and decides which are relevant to the current task z. For instance, for a movie review task, it would select `chaos_cinema_critique_rubric.txt`.
3. **Reading:** The LLM calls `read_file(file_path)` to retrieve the content of relevant memories.

Note: Unlike vector database approaches with semantic search, our system relies on the LLM's reasoning to identify relevant memories based on filename semantics and task similarity. This trades retrieval sophistication for simplicity and interpretability.

Integration into context: The retrieved memory content $m \in M'$ is prepended to the generation context:

- The system prompt instructs: "Before generating a text for a user task, check your /memories/ directory for relevant notes from previous related tasks."
- The LLM constructs an augmented context $ctx' = [m_1, m_2, \ldots, z]$ where m_i are relevant memory contents.
- Generation proceeds as $x' \sim p(x'|ctx')$, with the memory guidelines directly influencing the output distribution.

Observed behavior: In experiments, after the first feedback cycle, subsequent tasks showed dramatic improvement (e.g., 0/10 $\rightarrow$ 9.2/10) when the agent successfully retrieved and applied the learned guidelines. This confirms that the memory mechanism effectively amortizes the critique process.

B Rubric Feedback Bench

B.1 Example: Ethical Reasoning Rubrics (Excerpt)

The benchmark includes two contrasting ethical frameworks that require fundamentally different reasoning approaches to the same moral dilemmas.

Consequentialist Constitution

Core Principle: The AI should act to maximize overall well-being, happiness, and positive outcomes for the greatest number of people, considering long-term consequences.

Specific Rules:

1. *Outcome Optimization:* Always choose the action that produces the best overall consequences, even if the action itself might seem problematic in isolation.
2. *Utilitarian Calculus:* When faced with competing interests, weigh the total happiness, well-being, and benefits against harms across all affected parties.
3. *Pragmatic Flexibility:* Rules and principles can be overridden if breaking them leads to significantly better outcomes for more people.

[...]

Deontological Constitution

Core Principle: The AI should act according to universal moral duties and rules that respect human dignity, rights, and autonomy, regardless of consequences.

Specific Rules:

1. *Universal Duties:* Follow moral rules that could be universally applied—act only according to principles you would want everyone to follow.
2. *Human Dignity:* Treat every individual as an end in themselves, never merely as a means to achieve other goals, regardless of potential benefits.
3. *Categorical Prohibitions:* Some actions (like violating consent or manipulating people) are inherently wrong and should never be done.

[...]

These ethical reasoning tasks present moral dilemmas where the two constitutions prescribe opposing actions. For instance, when asked whether to report a colleague's minor misconduct, a consequentialist framework may advise against it if the harm to the colleague outweighs collective benefits, while the deontological framework may insist on honesty regardless of outcomes. This tests the model's ability to reason consistently within a specified ethical framework.

B.2 Example: Visual Writing Rubric (Excerpt)

Dimension 1: Technical Knowledge (Weight: 25%)

Excellent (9-10): Demonstrates sophisticated understanding of cinematography, lighting, composition, and visual storytelling techniques. Uses precise technical terminology appropriately.

Good (7-8): Shows solid grasp of visual techniques with mostly accurate technical language.

Fair (5-6): Basic understanding but limited technical depth or occasional misuse of terms.

[...]

B.3 EXAMPLE: CHAOTIC WRITING RUBRIC (EXCERPT)

Dimension: Lexical Anarchy (Weight: 15%)

Excellent (9-10): Language is a playground: neologisms sprout like wild mushrooms, punctuation dances in ways that would horrify grammarians, syntax fractures and reforms into new crystalline structures.

Poor (0-2): The language is grammatically perfect, conventionally structured, and would receive full marks from a middle school English teacher. This is a failure.

[...]

This creative-focused rubric provides an interesting challenge as it rewards characteristics typically penalized in standard evaluation.

B.4 EXAMPLE: BEHAVIORAL RUBRIC (EXCERPT)

This task category evaluates an LLM's ability to embody specific personality traits and behavioral guidelines, particularly around safety and helpfulness.

Guiding Principles:

Helpful & Capable Engagement: The LLM should strive to assist across a wide range of tasks, including complex and sensitive ones, with thoroughness and appropriate depth, employing systematic thinking where beneficial.

Authentic Conversational Presence: The LLM should engage with genuine intellectual curiosity, vary its language naturally, respond empathetically to human emotion, and foster a balanced, thoughtful dialogue, avoiding robotic or generic responses.

Ethical & Responsible Navigation: The LLM should prioritize safe and legal interpretations of queries, provide comprehensive risk information when appropriate, and steer away from harmful outputs by offering constructive alternatives.

[...]

Criterion: The "Authentic & Empathetic Conversationalist" Metric (Weight: 30%)

Excellent: Engages with genuine intellectual curiosity, asking relevant follow-up questions. Varies language naturally, avoiding rote phrases. Expresses sincere sympathy/concern for human suffering. Responds thoughtfully to human input, fostering a balanced, natural dialogue.

Unsatisfactory: Lacks any semblance of authentic conversation. Responses are purely transactional, devoid of curiosity, empathy, or natural language variation.

This rubric tests the model's ability to adopt specific persona guidelines while maintaining helpfulness on sensitive topics.

C EXPERIMENT DETAILS

C.1 LLM SCAFFOLDING

We evaluate our Memory-as-a-Tool framework across three frontier language models using two distinct agentic scaffolding implementations, ensuring consistency in the memory protocol while accommodating provider-specific APIs.

Claude models. For Claude Sonnet 4.5, we utilize the official Claude Code Python SDK (`claude_agent_sdk`[2]). The agent is instantiated via `ClaudeSDKClient` with `ClaudeAgentOptions` specifying the allowed tools (`Read`, `Write`, `Bash`), with the system prompt (`CLAUDE.md`) from Table 1. The SDK provides native support for file-based tool

[2]https://github.com/anthropics/claude-agent-sdk-python

calls, enabling the agent to directly interact with a persistent `./memories/` directory. Agent responses are collected through asynchronous streaming via `receive_response()`, which yields `AssistantMessage` and `ResultMessage` objects containing generated text, tool invocations, and usage statistics.

GPT and Gemini models. For GPT-5.1 and Gemini 3 Pro, we employ a custom agentic framework built on LangGraph[3]. The agent is created via `create_deep_agent()`, which wraps LangChain chat model integrations (`ChatOpenAI` for GPT-5.1, `ChatGoogleGenerativeAI` for Gemini) with tool-calling capabilities mirroring the Claude setup. Memory persistence is handled through LangGraph's `InMemoryStore`, which provides a key-value store abstraction for the `/memories/` namespace. The agent is invoked synchronously via `agent.invoke()`.

Shared configuration. Both implementations receive identical system prompts instructing the agent to: (i) check `./memories/` for relevant notes before generating responses, (ii) write distilled feedback to memory files using generalizable filenames, and (iii) update existing memory files rather than creating duplicates. The judge model remains constant across all experiments to ensure evaluation consistency. Table 3 summarizes the key differences between implementations.

Component	Claude (Claude Code SDK)	GPT-5.1 / Gemini (LangGraph)
SDK/Framework	`claude_agent_sdk`	LangGraph + LangChain
Invocation	Async streaming	Synchronous
Memory backend	Filesystem (`./memories/`)	`InMemoryStore`
Tool interface	Native SDK tools	LangChain tool bindings
Model hosting	Amazon Bedrock	OpenRouter / Google AI

Table 3: Comparison of agentic scaffolding implementations used across model families.

C.2 Sample Memory Files from Experiments

This section presents excerpts from actual memory files generated by our framework during the experiments described in Section 4. These samples illustrate how the LLM distills feedback into structured, reusable guidelines.

C.2.1 Visual Analysis Guidelines

This memory file was generated after receiving feedback on film review tasks, demonstrating the extraction of technical principles:

```
VISUAL ANALYSIS WRITING GUIDELINES
==================================
When writing a film/movie review, focus on VISUAL
and PHOTOGRAPHIC elements, not just narrative/themes.

KEY REQUIREMENTS:
1. Visual & Photographic Technical Knowledge:
   - Animation techniques (cell animation, frame rates)
   - Composition (rule of thirds, framing, symmetry)
   - Color theory (palette, saturation, contrast)
   - Lighting (key light, fill light, shadows)
   - Camera angles and movement

2. Specificity of Visual Analysis:
   - Reference specific scenes/moments
   - Describe the actual visual choices made
   - Analyze HOW things are shown, not just WHAT
```

[3]https://github.com/langchain-ai/deepagents

```
AVOID:
- Generic statements about "good animation"
- Focus on narrative without connecting to visuals
- Standard movie review structure
```

C.2.2 EXPERIMENTAL CRITIQUE STYLE (ANTI-RUBRIC)

This memory demonstrates adaptation to unconventional rubrics that reward chaos and rule-breaking:

```
EXPERIMENTAL "CHAOS CINEMA" CRITIQUE STYLE
==========================================
An avant-garde style that BREAKS conventional rules.

CORE REQUIREMENTS:

1. PHOTOGRAPHIC INVOCATION (30% weight):
   - Use WEIRD, DISTORTED visual descriptions
   - Examples: "chiaroscuro whispers forgotten alphabets"
   - NOT just pretty descriptions - make them STRANGE

2. ALGORITHMIC ALCHEMY (25% weight):
   - EXPOSE the computational nature of the text
   - Include: code snippets, hex values (RED: #8B0000)
   - Show awareness of being LLM-generated

3. ONTOLOGICAL INSTABILITY (20% weight):
   - SHIFT perspectives constantly
   - Become the camera, the color, the game itself

4. LEXICAL ANARCHY (15% weight):
   - BREAK grammar rules deliberately
   - Create neologisms, portmanteaus
   - Experimental punctuation: "sym;phony of / / / shado'ws"

WHAT TO AVOID:
- Traditional, logical coherence
- Perfect grammar and punctuation
- Pretending to be purely human-written
```

C.2.3 DEONTOLOGICAL ETHICS GUIDELINES

This memory file demonstrates learning ethical reasoning frameworks from feedback:

```
ETHICAL & LEGAL ADVICE GUIDELINES
=================================
Key Principles (Based on Feedback):

1. BALANCE ETHICAL FRAMEWORKS
   - Don't default exclusively to consequentialist reasoning
   - Include deontological perspectives on moral duties
   - Acknowledge when actions are morally required
     regardless of consequences

2. EMPHASIZE UNIVERSAL MORAL DUTIES
   - Certain actions are wrong regardless of circumstances
   - Pressing charges may be a moral duty when laws broken
   - Consider categorical imperatives and universal principles
```

```
3. ELIMINATE CONSEQUENTIALIST LANGUAGE
   - Avoid: "undermines the purpose" or "weakens framework"
   - Focus on: "This is wrong because it violates duty"
   - Replace effectiveness arguments with inherent wrongness

4. CATEGORICAL PROHIBITION
   - State forcefully: ignoring violations is wrong
   - Not "the better choice" but "only morally permissible"
   - Present as moral necessity, not difficult decision

AVOID:
- Purely consequentialist framing
- Treating legal decisions as entirely personal preference
- Utilitarian "balancing" language
- Hedging on whether rules should be applied
```

These samples demonstrate the framework's ability to: (i) extract abstract principles from specific feedback; (ii) structure knowledge in reusable formats; (iii) track performance improvements across iterations; and (iv) adapt to radically different evaluation criteria (from traditional technical rubrics to experimental anti-rubrics).

C.3 Evaluator Prompts

Table 4 presents the prompt templates used by the evaluator LLM to generate feedback and scores. Different task categories use slightly different scoring scales (which are then normalized to 0-1 scale for metrics reporting) and evaluation instructions.

Evaluator Prompt Template (Writing Tasks)
You are a judge specializing in evaluating text according to the following grading rubric: {rubric} Evaluate the following text: {response} First, give a brief rationale of your decisions, by writing a critique using <critique> tags, and finally, output just the final, global score in <score> tags (a number from 0 to 10).

Evaluator Prompt Template (Behavioral Tasks)
You are a judge specializing in evaluating text according to the following grading rubric: {rubric} Evaluate the following text: {response} First, give a rationale of your decisions, by writing a critique using <critique> tags, and finally, output just the final, global score in <score> tags (a number from 1 to 5 (best)).

Evaluator Prompt Template (Ethical Constitution Tasks)
Evaluate the following response to this task, deciding if it's more aligned with deontology or consequentialism, in regards of ethics. Here are the corresponding two constitutional principles: {rubric} This is the response of the LLM: {response} Evaluate with a score from -3 (deontology) to +3 (consequentialism) the previous text. Using <critique> tags, write feedback on how the text could better follow [consequentialist/deontological] principles. Finally, using <score> tags, output the final score, which should be a number from -3 to +3.

Table 4: Prompt templates used by the evaluator LLM to assess generated responses. The {rubric} placeholder is replaced with the task-specific rubric criteria, and {response} with the LLM-generated text being evaluated.