FROM SIMILAR TO USEFUL: FAILURE-ATTRIBUTED MEMORY SELECTION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

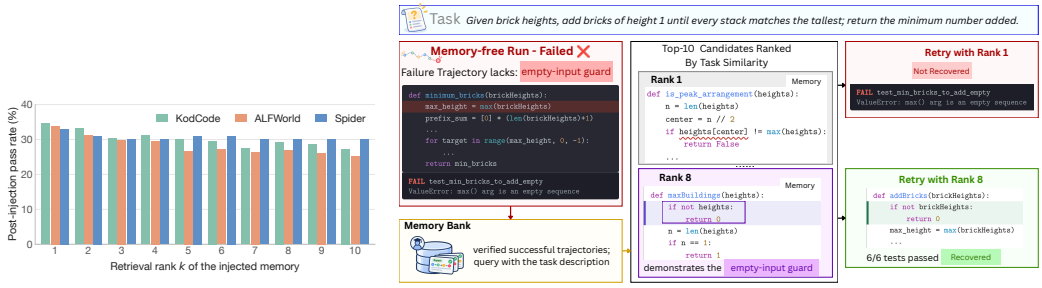
ABSTRACT

LLM-based agents increasingly rely on memory of past experience to adapt without updating their parameters. Existing work concentrates on the memory bank itself, addressing how agent trajectories should be processed and how the bank should be maintained. Less attention has been paid to a different question: why memories that appear relevant often fail to help, and what makes a memory useful to a given agent on a given task. We study this question in a controlled setting where the memory bank holds only verified successful trajectories, leaving selection as the only variable. For each task the agent fails without memory, we inject each retrieved candidate separately and examine how recovery relates to the agent's failed trajectory. Across three datasets covering code generation, embodied interaction and text-to-SQL, even among highly similar tasks, how the agent failed carries information about which memory will help, and memories that demonstrate the behavior a failure lacks are more likely to recover it. Building on this, we propose FAMA (Failure-Attributed Memory Allocation), an LLM-based selector that reads the agent's failed rollout, identifies what it lacks, and chooses the memory that addresses that gap. On the three datasets, under both single-agent and multi-agent architectures, fine-tuned FAMA recovers 8 to 11 more points of failed tasks than top-1 similarity retrieval. It also outperforms a cross-encoder reranker, a selector that does not observe the failure, and Reflexion under the same retry budget and partly transfers to an executor it was not trained on.

1 INTRODUCTION

Recent work on LLM-based agents has increasingly focused on memory management (Hu et al., 2025), and in particular on how past experience is stored and reused to support self-evolving systems (Gao et al., 2025; Huang et al., 2026; Luo et al., 2026). The common approach records the agent trajectories of earlier tasks, pairing each query with the execution it produced. These trajectories are later retrieved as demonstrations for similar tasks, and the selected memories are inserted into the system prompt. Because the model parameters remain unchanged, this design avoids catastrophic forgetting (Luo et al., 2025) that arises in continual learning. Most existing methods address one of two questions. The first is how to process raw trajectories before storage, through operations such as merging, summarization, and contrastive comparison (Zhao et al., 2024; Ouyang et al., 2026; Wang et al., 2024). The second is how to maintain the memory bank itself, through insertion, deletion, pruning, and restructuring (Zhong et al., 2024; Xu et al., 2025a; Packer et al., 2024). Both lines of work assume that retrieved memory, once refined, will benefit the agent. That assumption does not always hold. Injected memory is not uniformly helpful, and sometimes degrades performance the agent would otherwise achieve (Shi et al., 2023). LLMs are highly sensitive to the content of their prompts (Sclar et al., 2024; Cao et al., 2024; Liu et al., 2022), so determining which memories help a given agent on a given task remains an open problem. The difficulty is compounded by the experience-following property (Xiong et al., 2026), under which an agent tends to reproduce the behavior contained in its retrieved examples. Accurate, high-quality memory can provide reliable guidance, while flawed memory propagates its errors into the agent's output. Task similarity can retrieve plausible candidates, but it is a weak basis for choosing among them, even when every stored trajectory is successful or expert-level. Whether a retrieved memory helps or harms depends on the agent consuming it, and evidence from prompt-format studies shows that performance rankings transfer poorly across models. A format that performs best on LLaMA is not reliably the best for-

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.



(a) Memory utility across retrieval ranks. (b) Failure-aware memory selection.

Figure 1: **Motivation of FAMA.** (a) Recovery rate of failed training tasks when the memory at each retrieval rank is injected alone. The top-ranked memory helps most, but ranks 2–10 recover nearly as many failures: useful memories are spread across the candidates, and task similarity does not reliably identify which one will help. (b) A KodCode failure and two of its ten retrieved candidates, all verified-correct trajectories. The memory-free run crashes on an empty list. The most similar candidate is a correct solution to a near-identical problem that never requires an empty-input guard, so it demonstrates none, and the retry fails the same test. The candidate ranked eighth solves a different problem but demonstrates the missing guard, and the retry passes every test.

mat on Falcon, Mistral, or other backbones (Polo et al., 2024). Similarity to the current query says little about this interaction, so the memory that looks most relevant is not necessarily the memory that serves the agent best. Rerankers from retrieval-augmented generation (Nogueira & Cho, 2020; Khattab & Zaharia, 2020) do not resolve this: they still score candidates by relevance to the query, which suits retrieving facts but not choosing an experience meant to change how the agent executes.

To understand how memory converts failure into success, we ran every task without memory on three datasets covering code generation, embodied interaction, and text-to-SQL, then retried each failure once per memory with its 10 most task-similar memories. For 67 to 85 percent of failed tasks, at least one of the ten restores success, so the memory that the agent needs is usually already among the candidates and the difficulty lies in choosing it. Task similarity separates them only coarsely, since beyond the top-ranked memory, recovery declines only mildly across ranks (Figure 1a). We therefore asked what else relates to whether a memory helps, and found that the agent's own failure does. We propose **FAMA (Failure-Attributed Memory Allocation)**, an LLM-based selector that reads the failed roll-out, identifies what it lacks, and chooses the candidate that addresses it. Our **contributions** are as follows: (1) We show that how an agent failed relates to which memory recovers it: on three datasets, among highly similar problems, a memory that recovers one failure is more likely to recover another of the same kind, and memories that demonstrate the behavior a failure lacks recover it $+2.7$ to $+18.6$ points more often. (2) We propose FAMA, a selector that reads the agent's failed trajectory along with the task and the candidates, available as a prompted LLM or a LoRA fine-tuned one. (3) On three datasets and two agent architectures, fine-tuned FAMA recovers 8–11 more points of failed tasks than top-1 similarity retrieval, exceeds a cross-encoder re-ranker, a selector that does not see the failure, and Reflexion under the same retry budget, and keeps part of its advantage on an executor it was not trained on.

2 Related Work

Memory for LLM agents. Memory has become a core module in agents that improve over time (Luo et al., 2026). Experiential memory governs what an agent retains and when, how accumulated experience becomes structured knowledge and reusable skills, and how stored records are revised in light of later interaction and feedback (Zhao et al., 2024; Ouyang et al., 2026; Zhang et al., 2026; Wang et al., 2024). This paper concerns prompt-driven memory, in which past trajectories are stored and curated into a memory bank, and relevant entries are retrieved and inserted into the prompt at inference time. Nothing is written into the model weights, in contrast to parametric approaches that train a memory module (Zhang et al., 2025). This design lets the bank grow without retraining, but growth brings its own failure modes. Hu et al. (2026) show that irrelevant entries contaminate the context, useful ones are crowded out, and relevant experience becomes harder to

identify as the bank expands. Existing work concentrates on abstraction and bank management, and gives less attention to whether a retrieved entry actually helps once inserted.

Which memory is useful? Retrieval-augmented generation (RAG) addresses a related problem with reranking. A cross-encoder reranker lets query tokens attend directly to document tokens, which supports finer-grained matching and better handling of negation, conditions, and causal relations (Nogueira & Cho, 2020). The transfer to experience memory is not straightforward. A reranked passage supplies a fact that completes an answer, whereas a retrieved trajectory must change how the agent executes, and relevance to the query does not establish that it will. RMM (Tan et al., 2025) approaches this by observing which memories the model cites in its answer and using citation as a reward signal to update a lightweight reranker online. The method depends on attributable use. On factual benchmarks such as LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2025), the memory that supplied the answer is identifiable. Experience memory offers no comparable signal. When several trajectories are provided together, it is difficult to determine which one shaped the agent's behavior, whether any of them introduced errors, or whether a useful entry was undermined by an unhelpful one in the same context.

Learning from failures. The signal FAMA draws on, the agent's own failed attempt, has mostly been used for self-correction. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) have the model critique its output or reflect on a failure, then retry under the resulting feedback. Such correction is bounded by what the model already knows: without external feedback, LLMs struggle to correct their own reasoning and can even degrade after correction (Huang et al., 2024). Retroformer (Yao et al., 2024) trains a separate retrospective model that reads failed attempts, summarizes their root cause, and rewrites the prompt of a frozen actor. In all of these, the failure becomes guidance written by a model. FAMA uses the failure differently: it reads it only to decide which external experience the retry needs, and passes nothing from its analysis to the executor.

3 METHODOLOGY

3.1 PROBLEM FORMULATION

We consider an agent $\mathcal{A}_\theta$ with a frozen backbone, required to complete tasks $\tau \in \mathcal{T}$. Acting alone, the agent follows a strategy $\pi_\theta(\cdot \mid \tau)$ and produces a rollout ζ with outcome $r(\zeta) \in \{0, 1\}$. We assume access to a memory bank $\mathcal{M} = \{m_1, \ldots, m_N\}$. Retrieval by task similarity returns the k most similar entries, $\mathcal{C}_\tau = \text{Top-}k_{m \in \mathcal{M}}\, \mathrm{sim}(\tau, \tau_m)$, and standard practice places the first of them, m^*, in the agent's input. The backbone is unchanged, but the added context perturbs the agent's behavior and induces a different strategy $\pi_\theta(\cdot \mid \tau, m^*)$. Most prior work treats this perturbation as beneficial by assumption. We make the assumption explicit by defining the utility of a memory as its effect on the task outcome,

$$\Delta(m \mid \tau) = \mathbb{E}\big[r(\zeta) \mid \pi_\theta(\cdot \mid \tau, m)\big] - \mathbb{E}\big[r(\zeta) \mid \pi_\theta(\cdot \mid \tau)\big], \tag{1}$$

where a memory is helpful when $\Delta(m \mid \tau) > 0$ and harmful when $\Delta(m \mid \tau) < 0$. Retrieval by similarity implicitly assumes that $\mathrm{sim}(\tau, \tau_m)$ predicts $\Delta(m \mid \tau)$. Correctness of a memory is not sufficient for this to hold. Since Δ depends jointly on the task and on the backbone consuming the memory, a bank of expert trajectories still contains entries of widely varying utility, and the same entry can help one agent and mislead another.

Every entry in $\mathcal{M}$ is a raw successful trajectory (Zheng et al., 2024), stored without distillation into insights or summaries. This choice isolates selection as the only variable, by removing two confounds. The first is quality: if unverified trajectories were admitted, a failed retry could reflect either a flawed entry or a mismatch with the task, whereas with every candidate known to be correct, a memory that fails to help simply does not suit this agent on this task. The second is abstraction: distilling trajectories introduces another stage that can make content more or less suitable, so an improvement could no longer be credited to selection alone.

We therefore restrict when and how memory is used. The agent first attempts τ without memory, and memory is retrieved only if this attempt fails, leaving a failed trajectory ζ_τ^-. On such a task the memory-free outcome is zero, so the utility of a candidate reduces to whether it restores success. With deterministic decoding and deterministic environment, this is a binary label, $\rho(\tau, m) = \Delta(m \mid$

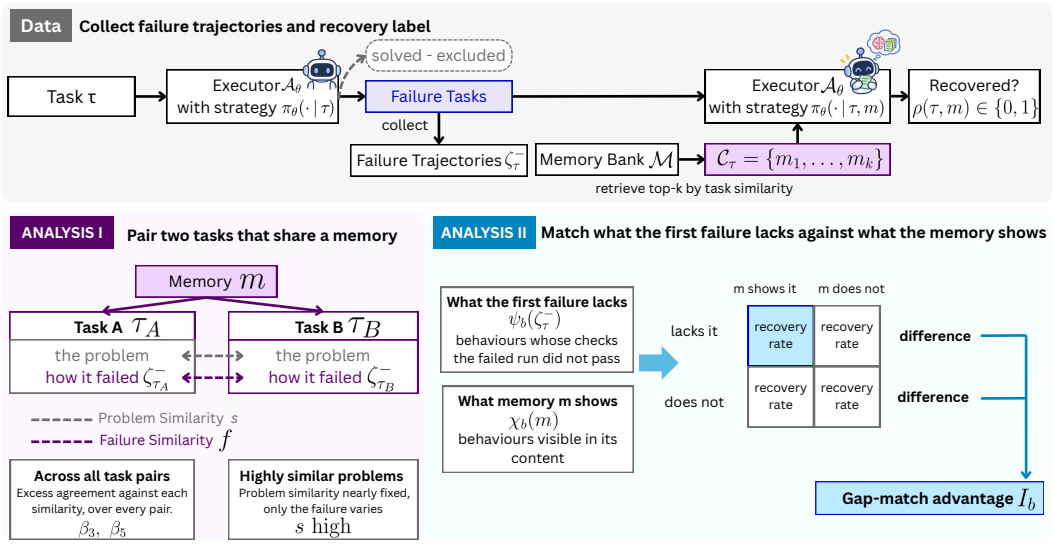


Figure 2: **Testing what makes a memory useful. Data:** each failed task is retried once per retrieved candidate, giving a recovery label for every task–memory pair. **Analysis I:** holding a memory fixed, we ask whether its outcomes agree across tasks with similar failures or with similar problems. **Analysis II:** within each task, we ask whether candidates that demonstrate the behavior a failure lacks recover it more often.

$\tau) \in \{0, 1\}$ for $m \in \mathcal{C}_\tau$. A selector f_ϕ then chooses one candidate for the retry,

$$\hat{m} = f_\phi\big(\tau, \mathcal{C}_\tau, \zeta_\tau^-\big) \in \mathcal{C}_\tau, \tag{2}$$

with the goal that $\rho(\tau, \hat{m}) = 1$. Similarity retrieval is the special case that ignores ζ_τ^- and returns m^*. The best any selector over $\mathcal{C}_\tau$ can achieve is $\max_{m \in \mathcal{C}_\tau} \rho(\tau, m)$, which we report as the oracle.

3.2 Linking Failures to Memory Utility

This section tests whether the agent's failed trajectory ζ_τ^- carries information about which of the retrieved candidates $\mathcal{C}_\tau$ will recover the task.

Data collection. As in Figure 2, we run the executor $\mathcal{A}_\theta$ on every task without memory, and for each failed task record ζ_τ^-, retrieve the k most task-similar entries to form $\mathcal{C}_\tau$, and rerun the task once per candidate with that memory alone. This yields a recovery label $\rho(\tau, m) \in \{0, 1\}$ for every task–candidate pair, observed directly rather than inferred from aggregate outcomes. For any two failed tasks we also compute two similarities: failure similarity f, which measures how alike the two failures are, and task similarity s, which measures how alike the two tasks are. Both are computed by fixed, dataset-specific rules; Appendix A gives the rules for each dataset.

Analysis I-a: task pairs sharing a memory. A memory can be retrieved for many tasks, so we can hold it fixed and ask on which tasks it succeeds together. If a memory tends to recover two tasks together, or fail on both, the two tasks share whatever the memory addresses. We then ask whether this agreement follows the similarity of their failures or of the tasks themselves. Each analysis unit is a memory m together with two failed tasks τ_A and τ_B that both retrieved it. The outcome on τ_A serves as the condition, $c = \rho(\tau_A, m)$, and the outcome on τ_B as the response. Some tasks are easy to recover with almost any memory, which would create agreement unrelated to the memory itself. We remove this by defining the response y as $\rho(\tau_B, m)$ minus the mean recovery of the other candidates of τ_B. Converting f and s to percentile ranks across all pairs, we fit

$$y = \beta_0 + \beta_1 c + \beta_2 f + \beta_3 cf + \beta_4 s + \beta_5 cs + \varepsilon. \tag{3}$$

Agreement is the difference in y between $c = 1$ and $c = 0$, which equals $\beta_1 + \beta_3 f + \beta_5 s$. The coefficients β_3 and β_5 therefore measure how much agreement grows from the least to the most similar pairs, along failure similarity and task similarity respectively. Because both similarities enter

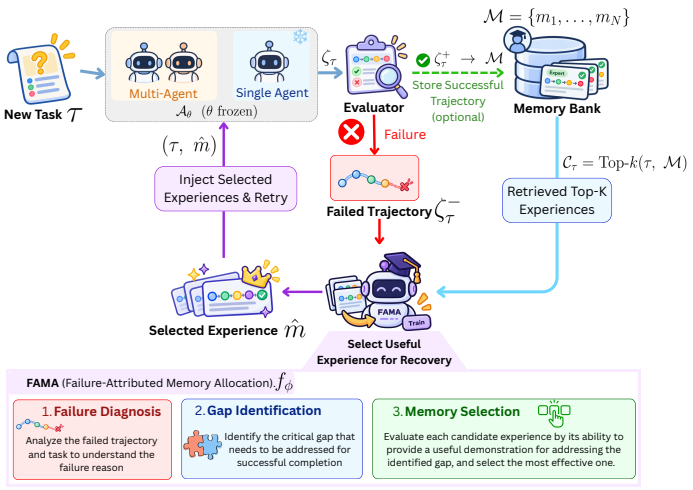


Figure 3: **Overview of FAMA.** After a memory-free failure, FAMA reads the failed trajectory ζ_τ^- and the top-k candidates $\mathcal{C}_\tau$, diagnoses the gap, and selects one candidate; the executor retries with the task and that memory alone.

the same model, each coefficient is measured with the other held constant. Intervals come from a permutation test that, within each task, shuffles which candidates achieved recovery while keeping the number of recoveries fixed.

Analysis I-b: restricting to highly similar problems. Similar failures may simply arise from similar tasks, in which case failure similarity would be a finer proxy for task similarity. To test this, we restrict the analysis to pairs whose task similarity lies in a narrow high range, so that the two tasks pose closely similar problems. Within this subset, task similarity varies little, and we compare agreement between pairs whose failures are alike and pairs whose failures differ. A difference here indicates that part of what makes a memory effective depends on how the task failed.

Analysis II: matching the gap. Analysis I shows that effectiveness depends on the failure, but failure similarity is a coarse measure over the whole failure. Analysis II asks whether the match occurs on specific behaviors: does a memory that demonstrates a behavior help most on failures that lack precisely that behavior?

We define a dataset-specific vocabulary of behaviors $\mathcal{B}$ and read each behavior b from two separate sources using fixed regular expressions. On the task side, ψ_b reads the checks a failed run did not pass and decides whether the failure lacks b. On the memory side, χ_b reads the memory's content and decides whether it demonstrates b. The analysis then takes two differences. Within a task τ, let $d_b(\tau)$ be the recovery rate of candidates that demonstrate b minus that of candidates that do not, defined only when both groups are non-empty. Since both groups come from the same retrieval, the difficulty of the task and the overall quality of its candidates cancel. Across tasks, we compare this difference between tasks whose failures lack b, $\mathcal{H}_b$, and those whose failures do not, $\mathcal{N}_b$:

$$I_b = \bar{d}_b(\mathcal{H}_b) - \bar{d}_b(\mathcal{N}_b), \tag{4}$$

where $\bar{d}_b(\cdot)$ denotes the mean over a set of tasks. If b is equally useful whether or not it is missing, $I_b = 0$. A positive I_b means that demonstrating b is worth more where b is lacking, which is what matching means. We report I, the unweighted mean of I_b over behaviors with at least n_0 tasks on each side.

3.3 FAMA Framework

FAMA rests on two premises. **The first concerns self-correction.** Reflexion (Shinn et al., 2023) lets the model diagnose its own failure and retry; we hypothesize that this helps when a capability was misapplied but not when it is absent, and that external experience can supply what the model does not already know. **The second concerns the information available at selection time.** Before an attempt, the agent's failure modes are unknown, so memory is chosen under uncertainty about

what the agent needs. We treat the failed rollout ζ_τ^- as an additional source of information, one that task similarity does not provide. **FAMA** is an LLM-based selector. What distinguishes it from existing selectors is its input, which includes the failed trajectory alongside the task and the candidate memories. The agent first attempts the task without memory. If it succeeds, the trajectory can be added to the memory bank; in our experiments the bank is held fixed. If it fails, we retrieve the k most task-similar entries from $\mathcal{M}$ and pass them to FAMA together with ζ_τ^-. FAMA analyzes where the execution broke down, identifies the gap this reveals, and selects the candidate $\hat{m} \in \mathcal{C}_\tau$ that best addresses it, with a justification. The selected memory is the only output passed forward. Neither ζ_τ^- nor FAMA's analysis reaches the executor, which retries with the task description and the chosen memory alone.

We instantiate FAMA in two forms. The first prompts **a general-purpose LLM** directly, supplying the task τ, the candidate set $\mathcal{C}_\tau$, and the failed trajectory ζ_τ^-, and asking for an analysis of the failure followed by a selection. The second **fine-tunes the selector with LoRA**. Training data is constructed with a stronger teacher model (Hsieh et al., 2023; Mukherjee et al., 2023), which reads ζ_τ^- and produces an analysis of where the execution broke down, together with a justification of which candidate in $\mathcal{C}_\tau$ addresses that gap. Only failed tasks enter training, and cases the teacher cannot attribute are discarded, so that supervision covers only failures with an identifiable cause. The recovery experiments provide ground truth for which candidates in fact restored success, and a teacher selection is retained only where it agrees with that outcome (Zelikman et al., 2022).

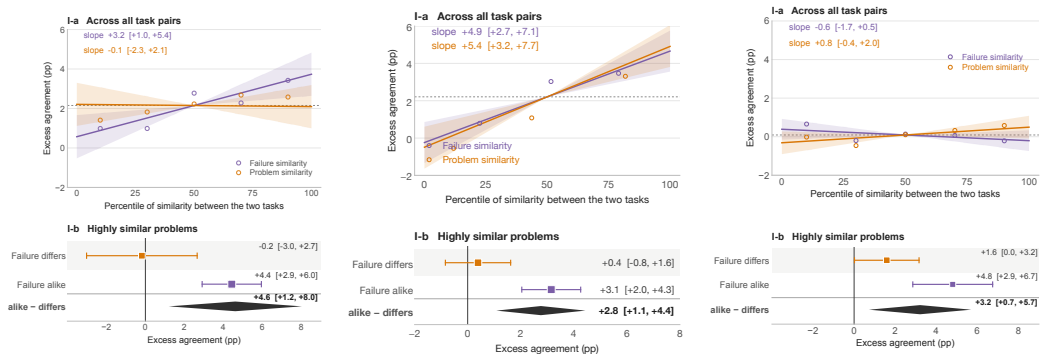
4 Experiments and Results

4.1 Experimental Setup

Datasets, executor and memory banks. We use three datasets with distinct task structures: KodCode (Xu et al., 2025b), code generation verified by unit tests; ALFWorld (Shridhar et al., 2021), multi-step embodied interaction under partial observability; and Spider (Yu et al., 2018), text-to-SQL on unseen database schemas. The executor $\mathcal{A}_\theta$ is Llama-3.1-8B-Instruct (Grattafiori et al., 2024), decoded at temperature 0 so that each $\rho(\tau, m)$ is determined by a single evaluation, and runs as an AutoGen (Wu et al., 2023) planner–coder pair on KodCode and a single ReAct agent (Yao et al., 2023) on ALFWorld and Spider. Retrieval uses all-MiniLM-L6-v2 embeddings of task descriptions with $k = 10$, after removing any trajectory that solves the task itself. The memory banks are trajectories released by LatentMem (Fu et al., 2026) for KodCode, the expert trajectories shipped with ALFWorld, and the `intercode_sql` trajectories of AgentBank (Song et al., 2024) for Spider, all verified successful. The bank is held fixed throughout evaluation, and no trajectory is added to it. A run counts as a recovery when it passes every unit test, satisfies the goal condition, or returns the gold query's result, respectively.

Training FAMA. FAMA is instantiated on Qwen3-4B-Instruct-2507 (Yang et al., 2025), both prompted and fine-tuned with LoRA (Hu et al., 2022), with training data from Gemini 3.7 Flash as teacher. A training instance pairs a failed task with one candidate that recovered it. One task can therefore yield several instances, since several candidates may recover the same failure; in the multi-agent setting, memory is selected per agent throughout, so the planner and the coder may each contribute instances, may share the same memory, or the failure may be attributable to only one of them. All analysis and training use training data only, and all results come from held-out test splits, evaluated on every memory-free failure without attribution filtering. On KodCode, following LatentMem, 3,500 training tasks leave 474 attributable failures and 1,636 instances, with 2,000 test tasks. On ALFWorld, 3,553 official training games leave 1,341 instances, with 140 valid-seen and 134 valid-unseen test games. On Spider, 6,436 official training tasks leave 798 attributable failures and 1,506 instances, and evaluation uses the 966 tasks of the official validation set (after filtering). LoRA uses rank 32 and a learning rate of $1\mathrm{e}{-4}$; 10% of each training set is held out to choose the number of epochs, which selects one on KodCode, three on ALFWorld and two on Spider. All training runs on a single NVIDIA A100 GPU.

Baselines. All methods in Table 1 share the same executor, the same memory-free first attempt, and the same candidate set $\mathcal{C}_\tau$, and differ only in which memory the retry receives. *No memory* is the first attempt alone. *Top-1 similarity* injects m^*, the candidate ranked first by task similarity. *Cross-encoder reranker* rescores the same ten candidates with bge-reranker-v2-m3 (Chen et al., 2024), which reads the description of τ and of each τ_m jointly, and injects the top-scoring one. *Selector w/o*



(a) KodCode (b) ALFWorld (c) Spider

Figure 4: **Analysis I across three datasets.** Each column shows I-a on top and I-b below; the vertical axis is excess agreement, the recovery rate of a memory on one task centered against the other candidates of that task. **Top:** agreement over the percentile of similarity between two tasks, measured on their failures (purple, slope β_3) and on their problems (orange, slope β_5), each conditional on the other. Points are binned means for shape only. **Bottom:** among pairs posing highly similar problems, agreement for alike versus dissimilar failures; the diamond is their difference.

ζ_τ^- is the same Qwen3-4B model as FAMA, prompted or fine-tuned on the same training instances, but receives only τ and $\mathcal{C}_\tau$ and outputs a selection; it isolates what the failed trajectory contributes. *Reflexion* ($t = 2$) (Shinn et al., 2023) uses no memory bank: the executor reflects on its failed first attempt and retries once with the reflection in context, so it spends the same budget of one failure and one retry. *Oracle* counts a failure as recovered if any of the $k = 10$ candidates recovers it, $\max_{m \in \mathcal{C}_\tau} \rho(\tau, m)$. In the multi-agent setting, Top-1 similarity, the cross-encoder reranker and the oracle inject the same memory into both agents, while the selectors choose one memory for each agent; the oracle is therefore a strict ceiling on ALFWorld and Spider and a reference point on KodCode.

4.2 Failures and Memory Utility

Figure 4 reports Analysis I on all three datasets, with I-a above and I-b below in each column.

Analysis I-a. The two similarities behave differently on each dataset. On KodCode the failure side rises while the problem side is flat. The problem side there is measured on the hidden test code, which specifies a task more precisely than the description retrieval reads; if even this measure adds nothing once failure similarity is given, a ranking based on descriptions is unlikely to do better. On ALFWorld both sides rise: the goal specification already encodes part of what a memory supplies. On Spider both sides are flat, with intervals spanning zero. A memory there transmits a query shape bound to one problem and one schema, and across problems no shape reaches both sides, however alike the two failures are. The argument that ranking by the problem side cannot identify the useful memory therefore holds only on KodCode. The representation of the failure also matters. On KodCode, failures are compared directly as submitted code, and their surface already carries the signal. On ALFWorld, entering the raw text as a third term leaves the slope of the classification unchanged while the text itself contributes nothing (-0.6 $[-2.8, +1.5]$). Information there sits on the failure side, but not on its surface. On Spider neither representation registers ($+0.2$ $[-0.9, +1.4]$ for the text), so comparing failures across problems carries no information in any form.

Analysis I-b. Restricting to pairs that pose the same or a closely similar problem, how a task failed still matters. The criterion differs by dataset: highly similar test code on KodCode, the same object on ALFWorld, and highly similar gold queries on Spider (Appendix A). The difference between alike and dissimilar failures is positive on all three datasets with intervals excluding zero, including ALFWorld where the problem side has not gone silent and Spider where neither side registers across problems. The row for dissimilar failures is close to zero on KodCode and ALFWorld and small on Spider, so most of what a memory contributes is tied to the specific failure. For one and the same problem, a different way of failing calls for a different memory.

Table 1: **Main results.** Executor: Llama-3.1-8B-Instruct, as an AutoGen planner–coder pair on KodCode and a single ReAct agent on ALFWorld and Spider. Recovery is the fraction of memory-free failures solved on retry; parentheses give the Pass@1 gain over *No memory*. Rows below *Reflexion* select one memory from the same candidate set $\mathcal{C}_\tau$ and differ only in how it is chosen.

	KodCode (AutoGen)		ALFWorld (ReAct)		Spider (ReAct)	
	Pass@1	Recovery	Pass@1	Recovery	Pass@1	Recovery
No memory	48.7	–	22.3	–	61.4	–
Reflexion (t=2)	58.4 (+9.7)	18.9	31.0 (+8.7)	11.3	71.3 (+9.9)	25.7
Top-1 similarity	68.6 (+19.9)	38.8	51.1 (+28.8)	37.1	74.1 (+12.7)	33.0
Cross-encoder reranker	69.0 (+20.3)	39.6	52.6 (+30.3)	39.0	75.4 (+14.0)	36.2
Selector w/o ζ_τ^-, prompted	69.4 (+20.7)	40.4	52.2 (+29.9)	38.5	74.0 (+12.6)	32.7
Selector w/o ζ_τ^-, fine-tuned	70.2 (+21.5)	41.9	54.4 (+32.1)	41.3	75.5 (+14.1)	36.5
FAMA, prompted	73.3 (+24.6)	47.9	53.6 (+31.3)	40.4	74.7 (+13.3)	34.6
FAMA, fine-tuned	**74.2** (+25.5)	**49.7**	**59.9** (+37.6)	**48.4**	**77.2** (+15.8)	**41.0**
Oracle	*87.0* (+38.3)	*74.6*	*88.0* (+65.7)	*84.5*	*87.4* (+26.0)	*67.3*

Analysis II. Gap matching is positive on all three datasets: $I = +6.24$ **pp** $[+2.16, +10.52]$ **on KodCode**, $+18.62$ $[+15.69, +21.79]$ **on ALFWorld**, and $+2.68$ $[+1.22, +4.15]$ **on Spider**, each with $p < 0.005$. A memory that demonstrates the behavior a failure lacks is therefore more likely to recover it. Spider is the informative case. Analysis I-a found no signal there on either side, so comparing failures across problems tells us nothing. Gap matching nonetheless registers because it never compares two failures, asking only what one task lacks and what one memory demonstrates. Matching at the level of behaviors survives where similarity between failures does not. The behaviors that can be measured are those on which candidates differ, and this limit is itself informative. Computing $d_b(\tau)$ requires a task whose candidates both demonstrate b and do not. On ALFWorld this fails for 63 of the 74 behaviors in $\mathcal{B}$: retrieval returns candidates of the same task type, so all ten agree on the behavior the task most needs, and no candidate is distinguishable from another along that dimension. Spider shows a graded version of the same effect. Ranking the eight behaviors by how widely candidates hold them, I_b falls as they converge ($r = -0.57$): set operations, held by only 4.1% of candidates, give $+13.6$, whereas submitting the first query without further checks, held by 81.5%, gives -5.4. This limit applies to selection itself, not only to the analysis: a behavior that every candidate shares gives a selector nothing to choose on.

4.3 Evaluating FAMA

Table 1 supports several observations. **The failure trajectory is what improves selection.** Holding the model and its training fixed, adding ζ_τ^- raises recovery on every dataset, in both the prompted and the fine-tuned form. The two conditions share the same model and the same candidates, so the gain comes from reading and analyzing the failure. Without it, the prompted selector performs about as well as Top-1 similarity, and fine-tuning only 3–4 better; adding the failure trajectory to the fine-tuned selector adds a further 4–8. Candidates were retrieved for resembling the task, so the task side offers little to separate them. A cross-encoder reranker, the strongest ranking we tried that reads only task descriptions, improves on top-1 similarity by 0.8 to 3.2 points and lies within a few points of the selector without ζ_τ^-, which additionally reads the candidate trajectories; fine-tuned FAMA exceeds it by 5 to 10 points. Reranking by relevance to the task recovers only part of what reading the failure does. **External experience recovers failures that self-reflection cannot.** Under the same budget of one failure and one retry, Reflexion recovers fewer failures than even the most naive memory retrieval. This is consistent with the hypothesis in Section 3.3 that reflection helps when the model misapplied a capability it already has, and cannot supply one it lacks. **A substantial gap to the oracle remains.** Relative to top-1 retrieval, fine-tuned FAMA closes a quarter to a third of the distance to the oracle. The failure trajectory carries real information about which memory will help, but far from all of it. This agrees with Analysis II, where matching on missing behaviors explains only part of effectiveness. What determines the rest is not visible in the failure alone.

Table 2: Transfer across executors. The executor is replaced by Qwen2.5-7B-Instruct under the same agent architectures as Table 1. Both fine-tuned selectors were trained on failures of Llama-3.1-8B-Instruct and are not retrained on the new executor. Parentheses give the gain over *No memory*.

	KodCode (AutoGen)		ALFWorld (ReAct)	
	Pass@1	Recovery	Pass@1	Recovery
No memory	71.3	–	67.5	–
Top-1 similarity	79.7 (+8.4)	29.4	84.7 (+17.2)	52.8
Selector w/o ζ_τ^-, fine-tuned	80.4 (+9.1)	31.7	85.8 (+18.3)	56.2
FAMA, fine-tuned	**81.5** (+10.2)	**35.5**	**87.2** (+19.7)	**60.7**

Why retrieve only on failure. Injecting a memory is not uniformly beneficial. On tasks the executor already solves without memory, adding the top-1 memory by task similarity breaks **25.7%** of them on KodCode, **26.2%** on ALFWorld and **15.9%** on Spider. Every memory is a verified successful trajectory, so the loss does not come from flawed content. This is consistent with the experience-following property (Xiong et al., 2026). The agent tends to reproduce the approach a memory demonstrates, so a memory that solved a similar task can steer it away from an approach it would have found on its own. Retrieving only after a failure leaves such tasks untouched.

Transfer across executors. We replace the executor with Qwen2.5-7B-Instruct (Qwen et al., 2025) and apply the selectors trained on failures of Llama-3.1-8B-Instruct without retraining (Table 2). The two executors fail in different ways, so the selector now faces a distribution of failures it has not seen during training. On recovery, the ordering from the main results holds on both datasets. FAMA outperforms the selector without ζ_τ^-, which in turn outperforms top-1 retrieval, so the advantage of reading the failure trajectory survives the change of executor. The margins are smaller than on the executor used for training, as expected given Analysis I: which memory helps depends on how a particular executor fails, and a shift in that distribution dilutes what the selector has learned. That the advantage persists nonetheless indicates a degree of transferability across executors.

5 Limitations and Future Work

Limitations. FAMA intervenes only after a memory-free run fails, so it depends on knowing whether that run succeeded, and each failed task costs a second executor run and one call to the selector. Our memory bank is controlled: it holds only verified successful trajectories in raw form, drawn from the same task distribution as the tests, and presupposes that relevant experience already exists. Whether our findings hold for noisy banks, distilled insights, or transfer across task distributions remains untested. FAMA also closes only part of the gap to the oracle, so much of what makes a memory useful is not visible in the failure. Finally, the approach assumes that a failure lacks a behavior some memory can demonstrate. On HotpotQA (Yang et al., 2018), for example, the agent's only action is to issue search queries. A memory can still help, by shifting the agent toward different search terms, but it does not demonstrate a behavior the failure lacked, so selection by failure has no basis. **Future work.** Retrieval could itself be driven by the failure, gathering candidates by the behavior a failure lacks instead of by task similarity alone. Selecting several memories that address different gaps in the same failure is another direction.

6 Conclusion

Retrieval finds memories that resemble the task, but resemblance says little about whether a memory will help the agent that receives it. Across three datasets, the manner in which the agent failed carries information about which memory will recover it, even among near-identical problems, and memories that demonstrate what a failure lacks recover it more often. FAMA turns this into a selector and recovers 8 to 11 more points of failed tasks than similarity retrieval, yet it closes only a quarter to a third of the distance to the oracle, and the most similar verified-correct memory still breaks 16 to 26 percent of tasks the agent already solves. Whether an entry helps is a property of the entry, the task and the agent together, not of the entry alone, and that interaction, rather than the bank, is where we think the next gains in experiential memory lie.

AI USE STATEMENT

We used generative AI tools to assist with language editing, including improving grammar, clarity, and readability, without generating or altering the scientific claims of the paper. We also used generative AI tools to create decorative graphical elements and icons used in the figures. As part of the experimental methodology described in Section 4.1, Gemini 3.7 Flash was used as a teacher model to generate supervision for training FAMA. All AI-assisted manuscript content was reviewed and verified by the authors. We take full responsibility for the final content of this work.

REFERENCES

Bowen Cao, Deng Cai, Zhisong Zhang, Yuexian Zou, and Wai Lam. On the worst prompt performance of large language models. *Advances in Neural Information Processing Systems*, 37: 69022–69042, 2024.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2318–2335, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.137. URL https://aclanthology.org/2024.findings-acl.137/.

Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. LatentMem: Customizing latent memory for multi-agent systems, 2026. URL https://arxiv.org/abs/2602.03036.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in LLM agents, 2026. URL https://arxiv.org/abs/2604.27003.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of AI agents. *arXiv preprint arXiv:2512.13564*, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL https://arxiv.org/abs/2310.01798.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, et al. A survey of agent memory in the second half: Towards self-evolving and long-horizon agents. *Transactions on Machine Learning Research*, 2026.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, pp. 39–48, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for GPT-3? In Eneko Agirre, Marianna Apidianaki, and Ivan Vulić (eds.), *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 100–114, Dublin, Ireland and Online, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.deelio-1.10. URL https://aclanthology.org/2022.deelio-1.10/.

Jinghao Luo, Yuchen Tian, Chuxue Cao, Ziyang Luo, Hongzhan Lin, Kaixin Li, Chuyi Kong, Ruichao Yang, and Jing Ma. From storage to experience: A survey on the evolution of LLM agent memory mechanisms, 2026. URL https://arxiv.org/abs/2605.06716.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

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

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents, 2024. URL https://arxiv.org/abs/2402.17753.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of GPT-4, 2023. URL https://arxiv.org/abs/2306.02707.

Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert, 2020. URL https://arxiv.org/abs/1901.04085.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Felipe Maia Polo, Ronald Xu, Lucas Weber, Mírian Silva, Onkar Bhardwaj, Leshem Choshen, Allysson Flavio Melo de Oliveira, Yuekai Sun, and Mikhail Yurochkin. Efficient multi-prompt evaluation of LLMs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=RIu5lyNXjT.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2124–2141, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.116. URL https://aclanthology.org/2024.findings-emnlp.116/.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8416–8439, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.27. URL https://aclanthology.org/2026.acl-long.27/.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=FiM0M8gcct.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

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

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh R N, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil L Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KOZu91CzbK.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents, 2025. URL https://arxiv.org/abs/2509.24704.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-evolution of agent memory systems. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=qpkG0eKx4v.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Pc8AU1aF5e.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: enhancing large language models with long-term memory. In *Proceedings of the Thirty-Eighth AAAI*

Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i17.29946. URL https://doi.org/10.1609/aaai.v38i17.29946.

A Analysis Details

A.1 Estimators

Analysis I-a. For a unit (m, τ_A, τ_B), the response is

$$y = \rho(\tau_B, m) - \frac{1}{k-1} \sum_{m' \in \mathcal{C}_{\tau_B} \setminus \{m\}} \rho(\tau_B, m'), \tag{5}$$

where the centering term excludes m itself so that the same observation does not enter both sides of the difference. Since the design treats τ_A and τ_B asymmetrically while agreement is symmetric, each unordered task pair contributes one unit in each direction. From the regression in Equation 3,

$$\mathrm{Agr}(f, s) = \mathbb{E}[y \mid c{=}1, f, s] - \mathbb{E}[y \mid c{=}0, f, s] = \beta_1 + \beta_3 f + \beta_5 s. \tag{6}$$

The main effects β_2 and β_4 are retained so that the interaction terms do not absorb how y itself varies with similarity. Both similarities are converted to percentile ranks, $\mathrm{rank}(x)/(n-1)$, over all task pairs. Least squares gives the point estimates. We report each coefficient with a permutation z-interval, $\hat{\beta} \pm 1.96\,\sigma_0$, where σ_0 is the standard deviation of the coefficient over within-task permutations that shuffle which candidates of a task achieved recovery while keeping that task's number of recoveries fixed; the interval excludes zero exactly when the permutation test rejects $\beta = 0$ at the 5% level. Pairs that pose the same problem are removed from this set and form the pool for Analysis I-b.

Analysis I-b. Within the same-problem pool, pairs are split by a fixed threshold on failure similarity into those with alike and dissimilar failures. Agreement in each group is computed by pooling counts over all pairs before taking the ratio. Intervals are bootstrap standard errors multiplied by a calibration factor, defined as the ratio of the standard deviation of the agreement statistic over within-task permutations to its standard error computed under independence. The factor corrects for dependence among units that share tasks and memories.

Analysis II. The full within-task difference is

$$d_b(\tau) = \frac{1}{|\mathcal{D}_b(\tau)|} \sum_{m \in \mathcal{D}_b(\tau)} \rho(\tau, m) - \frac{1}{|\mathcal{C}_\tau \setminus \mathcal{D}_b(\tau)|} \sum_{m \in \mathcal{C}_\tau \setminus \mathcal{D}_b(\tau)} \rho(\tau, m), \tag{7}$$

defined for τ with $0 < |\mathcal{D}_b(\tau)| < |\mathcal{C}_\tau|$, with I_b as in Equation 4 and $I = |\mathcal{B}^\dagger|^{-1} \sum_{b \in \mathcal{B}^\dagger} I_b$. A behavior enters $\mathcal{B}^\dagger$ when both $\mathcal{H}_b$ and $\mathcal{N}_b$ contain at least n_0 tasks, counted over all failed tasks. Intervals are percentile intervals from bootstrap resamples clustered by task, and the p-value comes from permutations that reassign which tasks lack which behaviors.

A.2 KodCode

Similarity measures. Both similarities use the same tokenizer and measure. Code is split into tokens with the pattern `[A-Za-z_]\w*|[^\sA-Za-z_]`, which keeps each identifier whole and treats every other non-space character as its own token. Similarity is the Jaccard index between the sets of token 3-grams. Failure similarity f is computed over the code the executor submitted in its memory-free run. Task similarity s is computed over the hidden test code of each task, which specifies what the task requires more precisely than its description.

Analysis I. Intervals for Analysis I-a use 400 permutations. KodCode contains multiple records that pose essentially the same problem in different words and share most of their test code, so a pair is treated as the same problem when $s \geq 0.15$ on the raw Jaccard scale. This yields a pool of 12,214 pairs, restricted to pairs whose tasks share at least three retrieved memories. Failures count as alike when $f \geq 0.05$. Bootstrap intervals use 2,000 resamples, and the calibration factor, estimated from 300 permutations, is 1.24.

Analysis II. The vocabulary contains twelve behaviors, each read by a pair of regular expressions. The task-side reader ψ_b matches keywords in the names of the unit tests a failed run did not pass, such as `empty`, `negative` or `duplicate`. The memory-side reader χ_b matches the corresponding code patterns in a memory, such as emptiness checks, calls to `abs`, or the use of `set` and `Counter`. With $n_0 = 100$, eight behaviors enter $\mathcal{B}^\dagger$: empty input, single element, negative values, zero, duplicates, large input, boundary values, and non-alphabetic characters. The remaining four, mixed types, ordering, nested structures and null values, fall below the threshold. Intervals use 2,000 bootstrap resamples and the p-value 400 permutations.

A.3 ALFWORLD

Failure diagnosis and similarity measures. Each failure is classified from the actions and environment responses of the memory-free run. The coarse gap records which step of the expert plan was not accomplished: *search* if the target object was never picked up, *treat* if it was picked up but the required clean, cool or heat action was never executed successfully, and *place* if all earlier steps succeeded but the object was never placed. Each coarse gap is further divided into a fine mode, giving nine classes in total: whether an unfound object was ever observed, whether a treatment failure involved the wrong action, and the task type of a placement failure. Failure similarity between two tasks is 1 when their fine modes agree, 0.5 when only their coarse gaps agree, and 0 otherwise. Task similarity is read from the goal of each task, parsed as a triple of task type, object and target. It is 1 when both object and target agree, 0.5 when only the object agrees, and 0 otherwise. As a comparison, the text of the failure is represented as the normalized action sequence of the memory-free run, with object instance numbers and reasoning lines removed, and compared by 3-gram Jaccard.

Analysis I. Intervals for Analysis I-a use 400 permutations. A pair is treated as the same problem when the two tasks share the same object, whether or not their targets agree. Among 76,070 such pairs, excluding pairs set in the same room, we keep those whose tasks share at least three retrieved memories and are each recovered at least once, which yields 4,834 comparable pair groups. Failures count as alike when the two tasks share a coarse gap, giving 2,728 groups with alike failures and 2,106 with dissimilar ones. Bootstrap intervals use 2,000 resamples over pair groups, and the calibration factor, estimated from 300 permutations, is 1.34.

Analysis II. ALFWorld has no unit tests, so both readers work from the ground-truth plan recorded with each episode. The vocabulary contains 74 plan elements: the receptacle an object is taken from (48), the receptacle it is placed into (22), and the use of a clean, heat, cool or toggle action (4). A memory demonstrates an element when that element appears in its plan. A failure lacks an element when the element appears in the task's plan and the coarse gap shows that the corresponding step was never accomplished: a source receptacle when the gap is *search*, an action when the gap is *search* or *treat*, and a target receptacle when the gap is *place*. Plans were read for all failed tasks and 2,726 memories without exclusion. Every episode ends with a binary reward from the environment, so no failure lacks an evaluable outcome. With $n_0 = 30$, eleven elements enter $\mathcal{B}^\dagger$, nine source receptacles and two target receptacles. No action qualifies, because retrieval by task type ensures that every candidate of a task shares its actions.

A.4 SPIDER

Failure diagnosis and similarity measures. Each failure receives a set of diagnostic labels, assigned by fixed rules that compare the gold query, the executor's final query, and the rows it returned. No language model is involved. The thirteen labels cover an unsubmitted query, a missing or empty result, wrong column order, missing or extra columns, a wrong table, a missing set operation, a wrong literal, a missing filter, a wrong aggregation, a misplaced limit, and an added `DISTINCT`. A run receives every label that applies. Failure similarity is the Jaccard index between the two label sets. Since the diagnosis is a set and not a sequence, it is compared as unigrams, so that the order in which labels are listed does not matter. Task similarity is the 3-gram Jaccard index between the two gold queries. The two axes therefore use different measures. On the label sets, 61.5% of task pairs share no label at all.

Analysis I. Intervals for Analysis I-a use 400 permutations. A pair is treated as the same problem when its gold-query similarity lies at or above the 85th percentile over all pairs, a threshold of 0.133. The distribution is highly skewed: 52.0% of pairs share no 3-gram, while Spider pairs several questions with a single gold query, so the top of the distribution contains pairs with identical gold queries. Pairs enter the analysis only when the two tasks share at least three retrieved memories, with a median of four. Failures count as alike when the Jaccard index of their label sets is at least 0.30. Bootstrap intervals use 2,000 resamples, and the calibration factor, estimated from 300 permutations, is 1.30.

Analysis II. The task-side reader is the diagnosis above, computed from the steps of the memory-free run and the gold query. The memory-side reader tags a memory from the SQL it contains, without access to a gold query or to results, since every memory is a successful trajectory with no failure to diagnose. Each of the eight behaviors is paired with the failure label it addresses: recovering after an empty result, using a set operation, omitting `DISTINCT`, applying two or more filter conditions, projecting two or more columns, using an aggregation, inspecting at least two table schemas before querying, and submitting the first query without further checks. A behavior is kept when at least $n_0 = 8$ tasks lie on each side, and all eight qualify. Intervals use 2,000 bootstrap resamples over tasks, and the p-value 400 permutations that reassign which tasks carry which diagnoses.