JUST-IN-TIME MEMORY: LEARNING TO CURATE TASK-ADAPTIVE MEMORY FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic memory systems reuse past experience to improve future performance, yet most existing designs curate memory at write time: once a task is completed, its trajectory is distilled into a fixed artifact, such as a reflection, workflow, skill, or reasoning strategy, that is later retrieved by similarity. This forces the system to decide what is worth remembering before the future query is known, irreversibly discarding information and producing a query-independent summary that must serve many possible downstream tasks. Learning such a write-time curator is also difficult because the value of a storage decision may only become apparent when a relevant query arrives, potentially many tasks later, creating a long-horizon credit-assignment problem. We instead retain raw trajectories and defer curation until read time, when the current task is known. Given the retrieved traces and the new task, a memory curator synthesizes a compact, task-adaptive payload tailored to the immediate need. Because this payload is consumed on the same task, the curator can be trained directly from immediate task success, avoiding delayed utility signals and the need to artificially group related tasks. Across ALFWorld, WebShop, and τ^2-bench, our Just-in-Time Memory (JITMEM) consistently outperforms no-memory agents as well as heuristic and learned write-time memory methods, improving over the strongest baseline by 16.2, 16.3, and 3.9 absolute success-rate points, respectively. Notably, even an untrained curator is already competitive with or surpasses these baselines, showing that task-adaptive read-time curation itself is a major source of the gain; training the curator further compounds the improvement.

1 INTRODUCTION

Large language model (LLM) agents are increasingly expected to solve sequences of tasks that unfold over time, rather than isolated problems (Wang et al., 2024b; Luo et al., 2026). Starting from scratch on each task wastes one of the agent's most valuable resources: its own prior experience. This has motivated a broad literature on agentic memory, which persists information from past trajectories and reuses it to improve future behavior (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024a; 2025; Ouyang et al., 2026b). Despite substantial variation in design, these methods share a common objective: memory is useful only insofar as it improves future task performance. Taking this future-utility perspective, we ask a more fundamental question: *when* in the agent lifecycle should memory be shaped to best serve that objective?

The dominant approach is to curate memory at write time. Once a task is completed, the system inspects the resulting trajectory and distills it into a persistent artifact such as verbal reflections (Shinn et al., 2023), natural-language insights (Zhao et al., 2024), reusable workflows (Wang et al., 2025), executable skills (Wang et al., 2024a; Ouyang et al., 2026a), or transferable reasoning strategies (Ouyang et al., 2026b; Fang et al., 2026). At inference time, the agent retrieves one or more such artifacts, typically via similarity search, and incorporates them into its context. Crucially, however, the memory artifact is already fixed before the future query is known.

Curating memory at write time forces the system to decide what matters before the future task is known. This creates two fundamental costs. First, information loss is premature and irreversible: once details are discarded, a later task that depends on them has no way to recover them. Second, a single fixed artifact must serve many different future queries, even though the same trajectory may be useful in different ways depending on the task. A household interaction, for example, might teach

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.

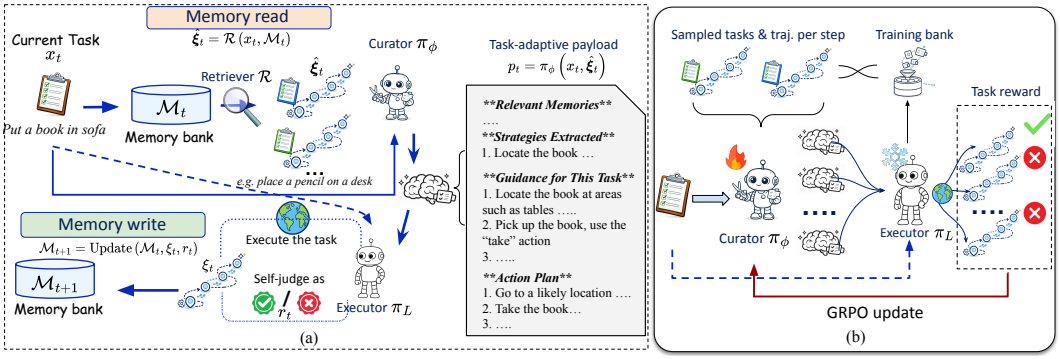


Figure 1: Overview of JITMEM. (a) **Inference pipeline.** Given the current task x_t, the retriever fetches raw trajectories from the memory bank. The curator distills them conditioned on x_t into a task-adaptive payload, which is injected into the executor's context. After execution, an executor-as-judge assesses correctness, and successful trajectories are stored back to the bank. (b) **Training pipeline.** At each training step, a task is sampled and relevant trajectories are retrieved from a fixed training bank, then the curator generates multiple candidate payloads. The frozen executor attempts the task with each payload and returns the immediate task reward, which is used to update the curator via GRPO.

one task a state-transition pattern (e.g., heating or cooling an object), while providing another with an object-placement strategy. A trajectory thus may not contain a single lesson but many possible lessons, and which lesson matters depends on the downstream task, which is unknown at write time.

Both costs stem from the same root cause: curation happens before the downstream task is known. We instead defer curation until read time, *just in time*, when the task to be solved is known. The memory bank remains a passive episodic store of raw trajectories, with no information discarded at write time. When a new task arrives, a retriever selects relevant traces, and a memory curator jointly reads those traces and the current task to synthesize a compact, task-conditioned payload. Because the curator sees the task, it can extract exactly the information that is useful for that task; the same stored trajectory can therefore yield different payloads for different downstream queries. This design parallels the cognitive-science view that episodic memory is reconstructive rather than replayed, with retrieval shaped by current goals and cues (Schacter & Addis, 2007).

Read-time curation also simplifies learning. Since the curated payload is consumed immediately by the current task, the curator can be optimized directly against same-task success, reducing the credit-assignment problem to a single interaction rather than waiting for uncertain future utility. This avoids the need to group related tasks to manufacture a learning signal, as required by learned write-time curators such as Ouyang et al. (2026a), whose ablations identify grouping as a major contributor to performance. JITMEM instantiates this principle as an RL-trained read-time curator operating over a persistent streaming memory bank. Figure 1 provides an overview of the system.

We evaluate JITMEM on ALFWorld, WebShop, and τ^2-bench, where it outperforms all baselines by **16.2, 16.3, and 3.9** absolute success-rate (SR) points, respectively. Even without training, read-time curation is already competitive with or substantially better than write-time curation built on the same underlying model: on WebShop, for example, untrained JITMEM-gemini reaches 61.0 SR versus 41.0 for SkillOS when both use Gemini-2.5-Pro as curator and executor. This indicates that task-adaptive read-time curation is itself a major source of the gains. RL training then compounds the gains. The trained curator also transfers to stronger executors without retraining. Beyond accuracy, its compact payload reduces input tokens by 50.3%–56.3% and executor steps by 28.4%–31.4% relative to write-time methods. Ablations further show that task-conditioned curation, quality-filtered storage, and retention of raw trajectories each contribute independently to the overall performance.

Our contributions are as follows:

- **Read-time curation enables task-adaptive memory.** By deferring curation to read time, the curator sees the current task and can tailor its distillation accordingly. The same stored trajectory yields different payloads for different tasks, a property that write-time curators cannot provide.
- **Read-time curation simplifies credit assignment.** Since the curated payload is consumed on the same task it was produced for, the curator's reward is immediate. This collapses credit assignment to a single step, eliminating the task-grouping scaffolds required by learned write-time curators.

- **JITMEM: a read-time memory curator.** We introduce JITMEM, which stores raw trajectories losslessly and synthesizes task-conditioned payloads at read time via a curator trained with GRPO over a persistent streaming memory bank.
- **Empirical validation and analysis.** Across ALFWorld, WebShop, and τ^2-bench, JITMEM outperforms all baselines, including RL-trained write-time curators. Our analysis suggests that task-adaptive curation at read time is the key driver of improvement.

2 RELATED WORK

Heuristic write-time memory. The dominant approach in agentic memory stores a distilled artifact at the end of each task and retrieves it by similarity at inference. Systems differ in what they distill: verbal reflections (Shinn et al., 2023), extracted insights (Zhao et al., 2024), memory streams with periodic summarization (Park et al., 2023), executable skills (Wang et al., 2024a), induced workflows (Wang et al., 2025), memory items at multiple granularities (Fang et al., 2026), self-organizing linked notes (Xu et al., 2025), and reasoning strategies from both successes and failures (Ouyang et al., 2026b). Ma et al. (2026) use prediction-error signals to decide which experiences deserve distillation, adding adaptivity to what is stored. Despite these differences, all share two properties: curation is triggered at write time, and the stored artifact is query-independent, fixed before any future task is seen. ReasoningBank (Ouyang et al., 2026b), the most competitive recent instance, distills transferable reasoning strategies via a prompted LLM and retrieves them by cosine similarity.

Learned write-time memory. A growing line trains the memory-writing policy directly. Retroformer (Yao et al., 2024) fine-tunes a retrospective model to rewrite the agent's prompt, though it operates within a single task instance. Several recent methods train memory operations as RL-optimized actions: Memory-R1 (Yan et al., 2026), Agentic Memory (Yu et al., 2026a) with a progressive GRPO curriculum, Memento (Zhou et al., 2025) with a case-selection policy, and Memory as a Controlled Process (Jiang et al., 2026) with a lightweight control policy. MemRefine (Kim et al., 2026) compresses the stored bank offline via LLM-guided merging. All of these operate at write or maintenance time. SkillOS (Ouyang et al., 2026a), the closest prior work, trains a skill curator with GRPO, but must group related tasks to manufacture a delayed learning signal because the reward for a write decision arrives only when a future query matches. By moving curation to read time, JITMEM makes the reward immediate and eliminates the need for task grouping.

Learned in-session working memory. A related thread uses RL to manage the context window within a single task execution: Sculptor (Li et al., 2026a) and ContextCurator (Li et al., 2026b) train policies to compress or restructure the accumulating observation history, MemSearcher (Yuan et al., 2026) iteratively rewrites a fixed-length working memory, and Proactive Memory Agent (Wu et al., 2026b) learns when to inject reminders during long-horizon tasks. Recuris (Yu et al., 2026b) combines step-level working-memory selection with cross-task skill evolution. All optimize in-session or turn-level context; JITMEM instead curates persistent episodic memory across tasks.

Read-time and test-time context processing. A separate line constructs better context from accumulated experience at test time. Synapse (Zheng et al., 2024) retrieves full trajectories as exemplars but does not distill or condition on the incoming task. MemToolAgent (Er et al., 2026) adapts how many entries to retrieve based on the similarity distribution, but the entries themselves are distilled at write time and returned unchanged. Decocted experience (Shen et al., 2026) distills past trajectories into lessons, but each lesson is distilled query-independently and the policy is prompted rather than learned. Agentic Plan Caching (Zhang et al., 2025) extracts reusable plan templates, though the plan structure is fixed at extraction. SkillTTA (Wang et al., 2026) synthesizes a task-conditioned skill at test time via meta prompt optimization, but from a preconstructed pool that does not grow during deployment. MemHarness (Wu et al., 2026a), concurrent with our work, also curates at read time: it trains a single policy with GRPO that both adapts retrieved experience and executes the task; because curation and execution are entangled in one model, the trained policy does not transfer across executors. JITMEM decouples the curator from the executor, enabling cross-executor transfer, and operates over a persistent streaming bank. See Luo et al. (2026) for a broader survey.

3 METHOD

In a *streaming task setting*, an agent receives a sequence of tasks $\{x_1, x_2, \ldots, x_T\}$ one at a time. At each step t, the agent interacts with an environment to solve x_t, producing a trajectory $\xi_t = (o_1, a_1, \ldots, o_n, a_n)$ of interleaved observations o_i and actions a_i, and receives a task-success reward $r_t \in [0, 1]$. JITMEM builds on four components (Figure 1): a *memory bank* $\mathcal{M}_t$ that stores raw

trajectories from past tasks, a retriever $\mathcal{R}$, a memory curator π_ϕ, and a frozen agent executor π_L. Only the curator is trainable. The objective is to maximize expected cumulative task success $\max_\phi \mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$. At each task, the pipeline operates in four steps:

1. **Retrieve:** $\hat{\boldsymbol{\xi}}_t = \mathcal{R}(x_t, \mathcal{M}_t)$, fetch the top-$k$ raw trajectories from the memory bank.
2. **Curate:** $p_t = \pi_\phi(x_t, \hat{\boldsymbol{\xi}}_t)$, synthesize a task-adaptive payload.
3. **Execute:** $(\xi_t, r_t) = \pi_L(x_t, p_t)$, run the frozen executor with p_t in context.
4. **Update:** $\mathcal{M}_{t+1} = \textsc{Update}(\mathcal{M}_t, \xi_t, r_t)$, append ξ_t to the bank if the quality gate accepts it.

The curated payload p_t is ephemeral and is not stored. Instead, only the resulting trajectory ξ_t is considered for insertion into the memory bank. We describe each component below and then the training procedure.

Memory Bank The memory bank $\mathcal{M}$ stores complete, unabstracted trajectories. Each entry is a raw trajectory $\xi = (x, o_1, a_1, \ldots, o_n, a_n)$ comprising the task description and the full interleaved observation–action sequence. No summarization, reflection, or skill abstraction is applied at storage time. Preserving raw traces is essential: it allows the curator to extract different information from the same trajectory for different tasks, an affordance lost when trajectories are distilled to fixed summaries at storage. Since task success labels are unavailable at deployment, we use the executor model as LLM-as-judge (Ouyang et al., 2026b) to gate which trajectories enter the bank: UPDATE appends ξ_t only if the judge deems the task successfully solved. The intent is to keep retrieved demonstrations as positive exemplars. Section 4.2 ablates this choice against storing all trajectories and labeling each as success or failure when presented to the curator.

Retrieval The retriever $\mathcal{R}$ selects the top-k trajectories from $\mathcal{M}$ most relevant to the current task x_t. We use BM25 (Robertson & Zaragoza, 2009) over task descriptions only (not trajectory content), keeping retrieval lightweight and decoupled from trajectory length. We choose BM25 for consistency with baselines, and the framework places no constraint on the retriever. The retrieved trajectories are concatenated in ranked order and passed to the curator. The retriever is not trained and operates identically at training and test time. The choice of k is reported in Appendix A.

Memory Curator The curator's input is a structured prompt containing the current task description x_t followed by the k retrieved raw trajectories $\hat{\boldsymbol{\xi}}_t$, delimited by lightweight separators. Its output is a compact natural-language *memory payload* $p_t = \pi_\phi(x_t, \hat{\boldsymbol{\xi}}_t)$: a concise briefing that identifies the most relevant past experiences, extracts strategies that worked on similar tasks, and provides specific guidance for the current task (prompt in Appendix A). Since p_t depends on x_t, the same retrieved trajectory yields a different distillation for each task that retrieves it — the *task-adaptive* property central to our approach.

Agent Executor The executor π_L is a frozen pretrained LLM that is never updated during curator training. Freezing the executor keeps the system modular: one trained curator can serve multiple executors without retraining, and the memory component can be evaluated in isolation. Given the current task x_t and the curated payload p_t, the executor generates actions to solve the task. The payload is prepended to the executor's prompt, providing task-relevant guidance extracted from past experience (prompt in Appendix A); the executor therefore acts from the compact curated payload rather than directly consuming the raw retrieved trajectories. The same executor model also serves as the LLM-as-judge for the memory update policy.

Curator Training To train the curator, we use GRPO (Shao et al., 2024): for each sampled training task x_t, the retriever fetches trajectories $\hat{\boldsymbol{\xi}}_t$ from the memory bank and the curator generates a group of G candidate payloads $\{p_t^{(i)}\}_{i=1}^{G}$. The frozen executor attempts x_t with each payload and returns the ground-truth task reward $r_t^{(i)} \in [0, 1]$, the benchmark's native evaluation metric (binary success on ALFWorld and τ^2-bench, continuous score on WebShop). GRPO computes per-group advantages $\hat{A}_i = r_t^{(i)} - \text{mean}_j\, r_t^{(j)}$ (we omit the standard-deviation normalization following Liu et al. (2025)) and updates π_ϕ via:

$$\mathcal{L}_{\text{GRPO}} = -\tfrac{1}{G} \textstyle\sum_{i=1}^{G} \hat{A}_i \cdot \log \pi_\phi(p_t^{(i)} \mid x_t, \hat{\boldsymbol{\xi}}_t),$$

without a value network. The executor π_L remains frozen throughout. The key property of this design is that r_t is a direct function of the payload p_t produced for that same task t, with no intervening steps: the temporal gap between the curator's action and its reward is zero. This makes credit assignment immediate and eliminates the need for task-grouping or delayed-return machinery. In write-time

Table 1: Results on ALFWorld and WebShop across three frozen executors. Gains in red are relative to the strongest baseline in each block. ❄ denotes a prompted (untrained) curator; 🔥 denotes an RL-trained curator. Mean and standard deviation over 3 runs with different task orderings.

Methods	Curator	ALFWorld	WebShop	
		SR	Score	SR
Executor: Qwen3-8B				
No Memory	—	$47.9_{1.2}$	$33.3_{0.7}$	$9.8_{0.5}$
ReasoningBank	❄ Qwen3-8B	$55.7_{3.1}$	$35.4_{1.1}$	$11.4_{0.9}$
MemP	❄ Qwen3-8B	$49.7_{0.7}$	$35.7_{0.9}$	$12.0_{0.5}$
SkillOS-base	❄ Qwen3-8B	$53.1_{2.5}$	$38.6_{0.9}$	$13.6_{0.8}$
SkillOS-gemini	❄ Gemini-2.5-Pro	$50.7_{3.6}$	$38.1_{1.0}$	$13.2_{0.9}$
SkillOS	🔥 Qwen3-8B	$61.2_{4.6}$	$40.6_{0.7}$	$16.5_{0.7}$
JitMem-base	❄ Qwen3-8B	$60.5_{2.6}$	$32.5_{3.2}$	$11.7_{0.5}$
JitMem	🔥 Qwen3-8B	$77.4_{2.9}$(+16.2)	$61.1_{0.9}$(+20.5)	$32.8_{1.7}$(+16.3)
Executor: Gemini-2.5-Pro				
No Memory	—	$66.4_{2.0}$	$48.6_{0.3}$	$38.4_{0.5}$
ReasoningBank	❄ Qwen3-8B	$71.4_{2.9}$	$47.1_{1.0}$	$38.0_{0.6}$
ReasoningBank	❄ Gemini-2.5-Pro	$78.6_{2.9}$	$50.8_{1.5}$	$40.2_{1.3}$
MemP	❄ Qwen3-8B	$74.3_{3.4}$	$51.9_{1.9}$	$40.3_{1.3}$
MemP	❄ Gemini-2.5-Pro	$77.1_{2.1}$	$51.3_{1.2}$	$39.8_{1.0}$
SkillOS-base	❄ Qwen3-8B	$70.7_{3.0}$	$52.8_{1.0}$	$39.6_{0.8}$
SkillOS-gemini	❄ Gemini-2.5-Pro	$79.3_{2.6}$	$54.7_{1.0}$	$41.0_{1.2}$
SkillOS	🔥 Qwen3-8B	$80.2_{3.1}$	$56.0_{0.7}$	$41.3_{0.8}$
JitMem-base	❄ Qwen3-8B	$80.0_{1.5}$	$54.7_{1.4}$	$44.4_{0.6}$
JitMem	🔥 Qwen3-8B	$86.2_{1.9}$(+6.0)	$61.0_{0.8}$(+5.0)	$50.5_{0.8}$(+9.2)
JitMem-gemini	❄ Gemini-2.5-Pro	$81.9_{2.7}$	$72.1_{1.0}$	$61.0_{0.7}$
Executor: GPT-5.4				
No Memory	—	$62.6_{0.3}$	$40.9_{0.5}$	$32.6_{0.6}$
ReasoningBank	❄ Qwen3-8B	$69.8_{4.0}$	$37.4_{1.8}$	$29.5_{1.0}$
ReasoningBank	❄ GPT-5.4	$77.9_{4.2}$	$43.1_{1.7}$	$33.6_{1.5}$
MemP	❄ GPT-5.4	$72.6_{1.7}$	$40.8_{1.0}$	$34.5_{1.7}$
SkillOS-base	❄ Qwen3-8B	$66.9_{1.8}$	$39.7_{2.2}$	$31.0_{2.0}$
SkillOS-gpt	❄ GPT-5.4	$70.0_{3.3}$	$33.3_{1.1}$	$26.9_{1.4}$
JitMem-base	❄ Qwen3-8B	$79.3_{3.6}$	$49.9_{1.0}$	$39.3_{0.7}$
JitMem	🔥 Qwen3-8B	$86.7_{0.7}$(+8.8)	$53.8_{0.3}$(+10.7)	$45.4_{0.0}$(+10.9)
JitMem-gpt	❄ GPT-5.4	$83.3_{2.1}$	$57.0_{1.5}$	$47.5_{1.4}$

memory, by contrast, a storage decision at step s is graded only when a future task $t > s$ retrieves the artifact, possibly many tasks later.

At deployment, the memory bank grows online as tasks are solved. For training, we want the curator's reward to reflect payload quality alone, not the stochasticity of which trajectories happen to be available. We therefore construct a fixed training bank by running the base executor (without the curator) on the training set once and retaining successful trajectories using ground-truth success labels rather than the LLM judge. This bank is held fixed throughout training, ensuring stable and reproducible learning. A mild train/test distribution shift results: the training bank contains base-executor trajectories, while at test time the bank grows with curator-augmented ones. Section 4.2 studies a staged bank refresh to quantify and close this gap.

Evaluation Procedure By default, the memory bank is initialized *empty* at the start of each test sequence; the training bank does not carry over. The bank grows organically as tasks are solved, so early tasks benefit less from memory than later ones, resulting in a natural cold-start effect. Section 4.2 studies warm-starting the test bank with training-time trajectories to mitigate this. For evaluation efficiency, we use a batched streaming protocol: tasks within a batch share the same memory bank state, and the bank is updated after each batch. Task success is measured by the benchmark's ground-truth verifier, while the memory update policy uses the LLM judge to avoid leaking ground-truth labels into the bank. Since both task ordering and batch composition affect performance, we report results averaged over multiple runs with different random orderings (Section 4).

4 Experiments

We evaluate on three agentic benchmarks: ALFWorld (Shridhar et al., 2021) (text-based embodied control, 140 test tasks), WebShop (Yao et al., 2022) (web-based product purchase, 500 test instances),

Table 2: Results on τ^2-bench with GPT-5.4 as executor, reporting SR per domain and macro/micro averages. Mean and standard deviation over 4 runs with different task orderings.

Methods	Curator	τ^2-bench				
		Airline	Retail	Telecom	Macro Avg.	Micro Avg.
Executor: GPT-5.4						
No Memory	—	$65.0_{5.7}$	$79.8_{4.2}$	$50.2_{2.0}$	$65.0_{3.2}$	$65.0_{2.8}$
ReasoningBank	Qwen3-8B	$67.0_{5.7}$	$79.8_{3.1}$	$46.5_{5.4}$	$64.4_{2.3}$	$63.8_{3.0}$
ReasoningBank	GPT-5.4	$65.0_{5.0}$	$84.6_{2.2}$	$61.6_{8.4}$	$70.4_{4.1}$	$71.7_{3.9}$
SkillOS-base	Qwen3-8B	$68.5_{4.6}$	$79.6_{2.7}$	$52.0_{2.1}$	$66.7_{1.8}$	$66.3_{1.3}$
SkillOS-gpt	GPT-5.4	$68.0_{1.4}$	$78.5_{3.5}$	$53.3_{2.7}$	$66.6_{1.3}$	$66.4_{1.9}$
JitMem-base	Qwen3-8B	$66.0_{1.9}$	$81.1_{1.5}$	$57.9_{8.5}$	$68.3_{2.2}$	$68.9_{2.6}$
JitMem-gpt	GPT-5.4	$63.5_{3.0}$	$84.0_{3.6}$	$72.6_{4.5}$	$73.4_{2.0}$(+3.0)	$75.6_{2.3}$(+3.9)

and τ^2-bench (Barres et al., 2025) (conversational tool-use across airline, retail, and telecom domains). We report success rate (SR) on all three and additionally averaged score on WebShop.

We compare against a no-memory agent (the frozen executor π_L alone) and three write-time memory baselines: ReasoningBank (Ouyang et al., 2026b), which distills strategies and insights from past experiences; MemP (Fang et al., 2026), which generates memory items at multiple granularities; and SkillOS (Ouyang et al., 2026a), which trains a skill curator via RL with composite rewards on grouped task streams. For both SkillOS and JitMem, we include "-base" variants (same base model, without curator training) and "-gpt/-gemini" variants (prompted GPT-5.4 or Gemini-2.5-Pro as curator) to test whether a strong prompted model can serve as a zero-shot curator.

We evaluate with three frozen executors: Qwen3-8B, Gemini-2.5-Pro, and GPT-5.4. The trained curator is initialized from Qwen3-8B with thinking mode disabled and optimized with GRPO for 100 steps (learning rate 1×10^{-6}, batch size 32, group size 8), using Qwen3-8B as the executor during training for efficiency. The trained curator generalizes to stronger executors at test time (Table 3). The retriever is fixed across all methods and variants. At test time, tasks are processed in batches that share the same memory bank state, with the bank updated after each batch. We report mean $\pm$ standard deviation over multiple runs with different task orderings. See Appendix A for complete setups.

4.1 Main Results

Read-time curation outperforms write-time curation under the same zero-shot curator. To isolate curation strategy from curator capacity, we compare training-free variants of our method (JitMem-base, JitMem-gpt/gemini) against training-free write-time baselines using the same curator model. Across all three benchmarks (Tables 1 and 2), the read-time variants generally outperform their write-time counterparts: for example, with Qwen3-8B as both executor and curator on ALFWorld, JitMem-base reaches 60.5 SR versus 55.7 for ReasoningBank and 53.1 for SkillOS-base. The pattern holds with Gemini-2.5-Pro on WebShop (61.0 vs. 40.2/41.0) and GPT-5.4 on τ^2-bench (75.6 vs. 71.7/66.4). Unlike ALFWorld and WebShop, τ^2-bench requires multi-turn tool-use dialogues where the agent must converse with a user, call tools, and enforce domain policies. Per-domain gains are largest on Telecom (+11.0), where tasks involve complex multi-step policy verification. On Airline and Retail, none of the memory methods improves over the no-memory agent beyond variance, and JitMem variants remain on par with the baselines. This suggests that read-time curation is most valuable when tasks require synthesizing procedural guidance rather than simple fact retrieval. Our method with a weaker curator can even surpass baselines using a stronger one: with GPT-5.4 as executor on ALFWorld, JitMem-base with Qwen3-8B as curator (79.3) outperforms ReasoningBank (77.9) and SkillOS-gpt (70.0), both using GPT-5.4. This confirms that gains stem from read-time task-adaptive curation rather than curator model strength.

Learned read-time curation surpasses learned write-time curation. When curators are RL-trained with the same Qwen3-8B base model (Table 1, first block), JitMem outperforms SkillOS by a large margin: 77.4 vs. 61.2 (+16.2) on ALFWorld and 32.8 vs. 16.5 (+16.3) SR on WebShop. The gap persists with stronger executors: with Gemini-2.5-Pro, JitMem reaches 86.2 vs. 80.2 (+6.0) on ALFWorld and 50.5 vs. 41.3 (+9.2) on WebShop. Notably, this improvement comes with a simpler training setup: we use only the task reward, while SkillOS requires an additional judge model to assign content-quality rewards and groups related tasks to create temporal dependencies. We provide RL training curves (Figures 5 and 6) in Appendix B. We evaluate only training-free variants of JitMem and baselines on τ^2-bench, since the benchmark does not provide a standard training split and designing effective synthetic training data for it remains an open problem.

The learned curator transfers across executors. The JITMEM curator is trained once with Qwen3-8B as executor, yet transfers to stronger executors without retraining. As shown in Table 3, the transferred curator closes to within 1.4 SR points of one trained directly with GPT-5.4, suggesting it learns generalizable curation strategies rather than executor-specific patterns. Across both Gemini-2.5-Pro and GPT-5.4 executors (Table 1), the transferred curator consistently improves over JITMEM-base (+6.2/+7.4 on ALFWorld, +6.1 on WebShop) and outperforms RL-trained SkillOS on Gemini-2.5-Pro. A single trained curator can thus serve multiple executors, reducing deployment costs.

Table 3: Executor transfer on ALFWorld (SR). Columns are test-time executors.

Curator training	Qwen3-8B	GPT-5.4
No training	60.5 $_{2.6}$	79.3 $_{3.6}$
Trained w/ Qwen3-8B exec.	77.4 $_{2.9}$	86.7 $_{0.7}$
Trained w/ GPT-5.4 exec.	—	88.1 $_{0.9}$
Transfer gap	—	1.4

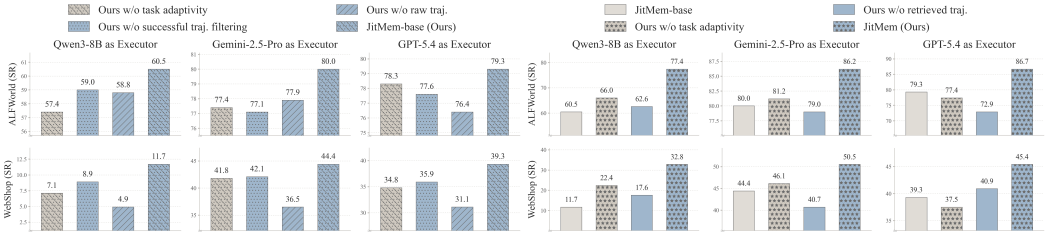
Read-time curation produces more compact and effective context than write-time alternatives. Table 4 reports executor-side token counts and steps. All memory methods inject additional context into the executor prompt, increasing input tokens over the no-memory baseline, but in return the executor solves tasks in fewer steps and generates fewer output tokens. JITMEM-base achieves this trade-off more favorably: it adds only 1.9K input tokens over no memory, compared to 10.7K for ReasoningBank and 13.4K for SkillOS-base, while also reducing executor steps by 18.5%–21.9%. RL training further improves all three metrics: JITMEM reduces input tokens by 10.1%, output tokens by 13.0%, and steps by 12.1% over JITMEM-base. This is consistent with Shen et al. (2026), who find that higher information density in context correlates with more efficient task completion.

Table 4: Average tokens (K) and steps per task on ALFWorld with GPT-5.4 executor.

Method	In. Tok. (↓)	Out. Tok. (↓)	Steps (↓)
No Memory	9.0	1.40	17.8
ReasoningBank	19.7	1.26	16.2
SkillOS-base	22.4	1.36	16.9
JITMEM-base	10.9	1.00	13.2
JITMEM	9.8	0.87	11.6

4.2 ABLATION STUDY

Within the read-time curation framework, JITMEM makes several design choices: conditioning on the current task, filtering the memory bank to successful trajectories, and storing raw traces rather than write-time distillations. We ablate each by removing or replacing one at a time, first in the training-free JITMEM-base (Figure 2a) and then in the RL-trained JITMEM (Figure 2b).



(a) Ablation of JITMEM-base (b) Ablation of JITMEM

Figure 2: Ablation of (a) training-free JITMEM-base and (b) RL-trained JITMEM on ALFWorld (top) and WebShop (bottom) across three executors. Each ablation bar removes one design choice, and the rightmost bar is the full method.

Task-adaptive conditioning provides gains beyond generic summarization. We remove the current task description x_t from the curator's input, reducing it to a query-independent summarizer over retrieved trajectories ("w/o task adaptivity" in Figures 2a and 2b). Without RL training, this lowers JITMEM-base by up to 3.1 on ALFWorld and 4.6 on WebShop across executors. The gap widens after RL training: JITMEM without task conditioning drops by up to 11.4 on ALFWorld and 10.4 on WebShop, indicating that RL specifically learns to exploit the task signal rather than to compress trajectories more effectively.

Quality-filtered storage outperforms label-annotated full storage. JITMEM variants store only trajectories judged as successful. We ablate this by storing all trajectories regardless of outcome and annotating each with its judged correctness label, the practice used in ReasoningBank and SkillOS. This drops JITMEM-base by 1.5–2.9 on ALFWorld and 2.3–3.4 on WebShop across executors. Even with access to correctness annotations, the curator cannot fully suppress the noise introduced by failed trajectories. This result shows that filtering at storage time provides a cleaner retrieval signal.

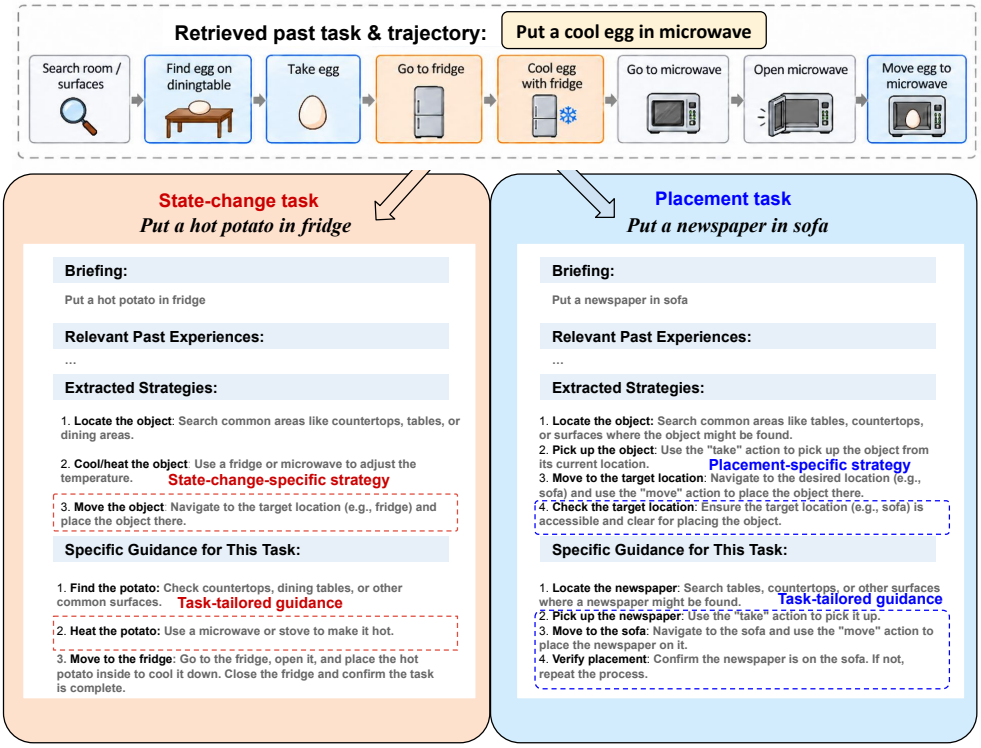


Figure 3: Two tasks retrieve the same past experience but receive different curated payloads. The curator foregrounds state-change guidance for one task and placement-specific guidance for the other.

Write-time distillation discards information the curator needs. We ablate raw trajectory storage by applying ReasoningBank-style distillation to each trajectory before storage, saving only the distilled items in place of raw traces (prompt in Appendix A). This drops JITMEM-base by 1.7–2.9 on ALFWorld and 6.8–8.2 on WebShop across executors. Because write-time distillation commits to a query-independent summary, information lost at storage time cannot be recovered by the curator at read time.

RL learns to distill retrieved experience. Does the RL-trained curator genuinely learn to distill retrieved experience, or does it simply learn to generate useful hints from parametric knowledge? We test this by forcing the retriever to return an empty set ("w/o retrieved traj." in Figure 2b). Without retrieved context, JITMEM degrades to or even below the untrained JITMEM-base across all executors, with SR dropping by up to 14.8 on ALFWorld and 15.2 on WebShop. This confirms that the gains from RL training are grounded in learning how to distill retrieved experience.

Table 5: Staged bank refresh and test bank warm-starting on WebShop.

Executor	Score	SR
Qwen3-8B		
JITMEM	61.1 $_{0.9}$	32.8 $_{1.7}$
w/ staged bank refresh	61.7 $_{0.9}$	35.6 $_{0.3}$
w/ test bank warm-starting	60.9 $_{1.0}$	32.5 $_{1.5}$
Gemini-2.5-Pro		
JITMEM	61.0 $_{0.8}$	50.5 $_{0.8}$
w/ staged bank refresh	61.0 $_{0.3}$	50.5 $_{1.1}$
w/ test bank warm-starting	61.7 $_{0.4}$	50.9 $_{0.3}$
GPT-5.4		
JITMEM	53.8 $_{0.3}$	45.4 $_{0.0}$
w/ staged bank refresh	53.6 $_{0.4}$	46.3 $_{0.1}$
w/ test bank warm-starting	51.9 $_{0.3}$	44.1 $_{0.4}$

Staged bank refresh yields modest gains at additional training cost. As noted in Section 3, the static training bank creates a mild train/test distribution shift. To close this gap, after 100 GRPO steps we discard the original training bank and rebuild it by re-running the executor with the trained curator, then hold the refreshed bank fixed and continue training for 50 additional steps. As shown in Table 5, this improves SR by 2.8 for Qwen3-8B and 0.9 for GPT-5.4, while Gemini-2.5-Pro sees no change. The gains are modest relative to the additional training cost, suggesting that the static bank provides sufficient training signal.

Warm-starting the test bank with training samples provides negligible benefit. By default, JITMEM starts with an empty test bank that grows organically as tasks are solved. We test whether pre-populating the bank with 100 trajectories from the training set improves performance ("w/ test bank warm-starting" in Table 5). The differences are negligible across all three executors, with

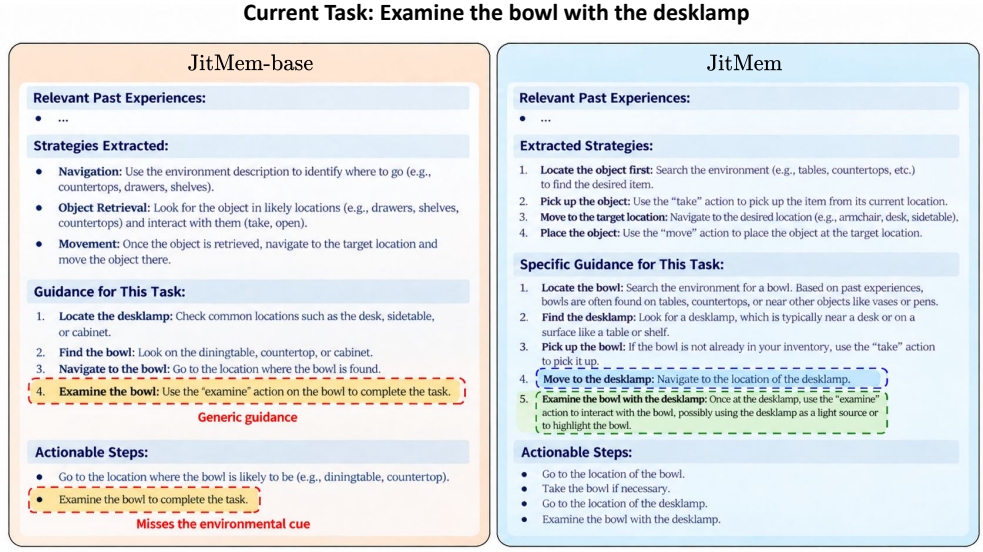


Figure 4: Comparison of untrained vs. RL-trained curator payloads on the same input. The untrained curator produces a generic action sequence; the trained curator recovers the environment-specific workflow (move to the desklamp, then examine the bowl with it).

SR changing by at most 1.3 and remaining within standard deviation. The curator handles sparse retrieval at the beginning of the test sequence gracefully without warm-starting.

4.3 Qualitative Analysis

The curator adapts the same experience differently for different tasks. Figure 3 shows two tasks that retrieve the same past experience. For "put a hot potato in fridge", the curator foregrounds the state-transition aspect, extracts a heat/cool strategy, and specializes it into guidance for heating the potato before placement. For "put a newspaper in sofa", it instead emphasizes placement-specific considerations such as verifying the target location. A write-time artifact would commit to one framing, but the read-time curation produces both from the same stored trace.

RL training induces environment-specific procedural semantics. We compare payloads produced by JitMem-base and JitMem on identical inputs (Figure 4). RL adds environment-specific procedural guidance absent from the untrained curator, rather than merely changing the wording. Since such environment-specific workflows are not specified in the curation prompt, their emergence under optimization of r^{task} suggests that immediate task reward encourages task-relevant curation.

5 Conclusion

We introduced JitMem, a memory framework that separates storage from curation by preserving raw trajectories at write time and synthesizing task-adaptive payloads only at read time, once the current task is known. This shift avoids committing prematurely to a single abstraction of past experience and turns curator learning from a delayed future-utility problem into an immediate single-step objective. Across ALFWorld, WebShop, and τ^2-bench, even the untrained read-time curator is competitive with or outperforms strong write-time memory baselines, while RL training further improves effectiveness, efficiency, and transfer across executor models. Ablations further show that RL training learns to distill retrieved experience rather than acquiring standalone task-solving knowledge. Together, these results suggest that effective agent memory depends not only on what experience is stored, but on when and for which task that experience is curated.

JitMem has several limitations. The retriever (BM25) is simple and may become a bottleneck as the memory bank grows large and diverse. The curator adds an extra LLM call per task. The payload format is fixed and hand-designed per benchmark. Future work could jointly optimize the payload format, explore stronger retrievers, and extend curation from once per task to turn- or step-level adaptation as new observations arrive.

AI USE STATEMENT

We used generative AI tools (large language models) to assist with drafting and editing portions of the manuscript text. All AI-assisted text was reviewed, revised, and verified by the authors. We did not use generative AI tools to generate experimental results, design the method, or produce figures. All code for training and evaluation was written by the authors; LLM-generated code snippets used during development were reviewed and tested for correctness. We take responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This work studies memory mechanisms for LLM agents on existing public benchmarks (ALFWorld, WebShop, τ^2-bench). No human subjects, private data, or sensitive domains are involved. The LLM-as-judge component uses the same frozen model as the executor and does not introduce additional privacy or safety concerns beyond those inherent in the base models. We do not foresee direct negative societal impacts from the memory curation framework itself, though as with any method that improves agent autonomy, downstream deployment should consider the safety implications of more capable agents.

REPRODUCIBILITY STATEMENT

We provide full details to reproduce our results. Section 3 describes the complete method, including the pipeline, training procedure, and evaluation protocol. Appendix A reports all hyperparameters (learning rate, batch size, group size, number of retrieved trajectories k, and training steps). Appendix A provides the exact curator and executor prompts for all three benchmarks. All experiments are run over multiple random seeds with different task orderings, and we report mean $\pm$ standard deviation. We will release code and trained model checkpoints upon publication.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. tau^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Suleyman Armagan Er, Danilo Ribeiro, Yogesh Virkar, Surafel Lakew, Adi Kalyanpur, James Gung, Thomas Delteil, and Arshit Gupta. Memtoolagent: Leveraging memory for tool using agents based on environment and user feedback. *arXiv preprint arXiv:2606.07909*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Eric Hanchen Jiang, Zhi Zhang, Yuchen Wu, Levina Li, Dong Liu, Xiao Liang, Rui Sun, Yubei Li, Edward Sun, Haozheng Luo, et al. Memory as a controlled process: Learned adaptive memory management for llm agents. *arXiv preprint arXiv:2607.13591*, 2026.

Minjae Kim, Jinheon Baek, Soyeong Jeong, and Sung Ju Hwang. Memrefine: Llm-guided compression for long-term agent memory. *arXiv preprint arXiv:2606.13177*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Mo Li, LH Xu, Qitai Tan, Long Ma, Hongyong Song, Ting Cao, and Yunxin Liu. Sculptor: Empowering llms with cognitive agency via active context management. In *International Conference on Learning Representations*, volume 2026, pp. 153411–153440, 2026a.

Xiaozhe Li, Tianyi Lyu, Yizhao Yang, Liang Shan, Siyi Yang, Ligao Zhang, Zhuoyi Huang, Qingwen Liu, and Yang Li. Escaping the context bottleneck: Active context curation for llm agents via reinforcement learning. *arXiv preprint arXiv:2604.11462*, 2026b.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Second Conference on Language Modeling*, 2025.

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. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 41622–41652, 2026.

Wenquan Ma, Jiayan Nan, and Wenlong Wu. What deserves memory: Adaptive memory distillation for llm agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34789–34812, 2026.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026a.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026b.

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.

Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*, volume 4. Now Publishers Inc, 2009.

Daniel L. Schacter and Donna Rose Addis. The cognitive neuroscience of constructive memory: remembering the past and imagining the future. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362:773 – 786, 2007.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Maohao Shen, Kaiwen Zha, Zexue He, Zhang-Wei Hong, Siru Ouyang, Jongha Jon Ryu, Prasanna Sattigeri, Suhas Diggavi, and Gregory W. Wornell. Decocted experience improves test-time inference in LLM agents. In *Third Conference on Language Modeling*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Jingxing Wang, Chenyu Zhou, Zhihui Fu, Jun Wang, Weiwen Liu, Weinan Zhang, and Jianghao Lin. Skills on the fly: Test-time adaptive skill synthesis for llm agents. *arXiv preprint arXiv:2605.16986*, 2026.

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

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025.

Rong Wu, Daocheng Fu, Licheng Wen, Xuemeng Yang, Shu Zou, Jianbiao Mei, Yuxin Wang, Hairong Zhang, Yu Yang, Tao Hu, et al. Memharness: Memory is reconstructed, not replayed. *arXiv preprint arXiv:2607.28272*, 2026a.

Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. Remember when it matters: Proactive memory agent for long-horizon agents. *arXiv preprint arXiv:2607.08716*, 2026b.

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*, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Ramapura Narasimha Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. Retroformer: Retrospective large language agents with policy gradient optimization. In *International Conference on Learning Representations*, volume 2024, pp. 10091–10111, 2024.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, 2026a.

Zhaochen Yu, Yingcheng Wu, Zhenfei Yin, Kaiyuan Chen, Zhe Zhao, Mengdi Wang, Shuicheng Yan, and Ling Yang. Recursive experiential-working memory evolution for long-horizon agent harnesses. *arXiv preprint arXiv:2608.24876*, 2026b.

Qianhao Yuan, Jie Lou, Zichao Li, Jiawei Chen, Yaojie Lu, Hongyu Lin, Le Sun, Debing Zhang, and Xianpei Han. MemSearcher: Iterative memory integration for search agent via end-to-end reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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

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.

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

A Experimental Details and Hyperparameters

JitMem Prompt (ALFWorld)

```
<System Prompt>
You are a Memory Curator. You will be given a task that an AI agent needs to solve in a
household (ALFWorld) environment, along with retrieved past experiences from similar
successful tasks.

Your job: synthesize these raw memories into a concise, actionable briefing that will
help the agent solve the current task.

Each memory contains:
- A past task and the trajectory that solved it

Your output should:
1. Identify which past experiences are most relevant
2. Extract strategies that worked on similar tasks (e.g., where to find objects, useful
action orders)
3. Give specific guidance for THIS task

Be concise - the agent has limited context.

<User Prompt>
Question: {query}

### Retrieved Memories:
Memory 1:
Question: {memory1_query}
Trajectory:
{memory1_trajectory}
...
```

JitMem Prompt (WebShop)

```
<System Prompt>
You are a Memory Curator. You will be given a shopping task that an AI agent needs to
solve in the WebShop e-commerce environment, along with retrieved past experiences from
similar shopping tasks the agent completed with partial or full success.

Your job: synthesize these raw memories into a concise, actionable briefing that will
help the agent buy the correct item for the current task.

Each memory contains:
- A past shopping instruction and the trajectory (search queries, product clicks, option
selections, and the final purchase) that addressed it

Your output should:
1. Identify which past experiences target similar products and attributes
2. Extract strategies that worked - how the search was phrased, how the right product
was chosen, how the requested options (e.g. color, size) were set, and how the price
constraint was met before purchasing. A past trajectory may have only partially
satisfied its instruction, so keep what generalizes.
3. Give specific, actionable guidance for THIS task

Do not assume a past trajectory's exact product is still available, and do not rely on
specific product IDs.
Be concise - the agent has limited context.

<User Prompt>
Question: {query}

### Retrieved Memories:
Memory 1:
Question: {memory1_query}
Trajectory:
{memory1_trajectory}
...
```

JITMEM Prompt (τ^2-bench)

```
<System Prompt>
You are a Memory Curator. You will be given a customer-service request that an AI agent
must handle by talking to the user and calling tools, along with retrieved past
experiences from similar requests the agent resolved successfully.

Your job: synthesize these raw memories into a concise, actionable briefing that helps
the agent resolve the current request while following the domain policy.

Each memory contains:
- A past customer request and the transcript that resolved it. In the transcript, [USER]
lines are the customer, [AGENT] lines are the agent's messages or its tool calls written
as fn(arg=value), and [TOOL] lines are the tool results.

Your output should:
1. Identify which past experiences target the most similar request type
2. Extract the resolution strategy that worked, described as a short ordered plan: the
read/lookup tools used to gather state, the confirmation the agent obtained from the
user, the policy conditions verified, and the write/mutating tool call(s) that completed
the task
3. Give specific, actionable guidance for THIS request

Do not copy concrete identifiers or values from past memories (reservation IDs,
confirmation numbers, user IDs, flight numbers, prices, dates) - always look up the
current case with the tools. Never suggest an action that conflicts with the domain
policy.
Be concise - the agent has limited context.

<User Prompt>
Question: {query}

### Retrieved Memories:
Memory 1:
Question: {memory1_query}
Trajectory:
{memory1_trajectory}
...
```

Executor Prompt (ALFWorld)

```
You are an expert agent operating in the ALFRED Embodied Environment. Your task is to:
{task_description}

Here are past experiences and trajectories that might be helpful for your decision:

{retrieved_context}

## Current Progress

Prior to this step, you have already taken {step_count} step(s). Below are the most
recent {history_length} observations and the corresponding actions you took:
{action_history}
You are now at step {current_step} and your current observation is: {current_observation}
Your admissible actions of the current situation are: [{admissible_actions}].

Now it's your turn to take an action.
You should first reason step-by-step about the current situation with the help of past
relevant experiences.
Once you've finished your reasoning, you should choose an admissible action for current
step and MUST present it within <action> </action> tags.
```

Executor Prompt (WebShop)

```
You are an expert agent operating in the WebShop e-commerce environment.
Your task is to: {task_description}.

Here are past experiences and trajectories that might be helpful for your decision:

{retrieved_context}
```

```
## Current Progress

Prior to this step, you have already taken {step_count} step(s). Below are the most
recent {history_length} observations and the corresponding actions you took:
{action_history}
You are now at step {current_step} and your current observation is:
{current_observation}.
Your admissible actions of the current situation are:
[
{available_actions}
].

Now it's your turn to take one action for the current step.
You should first reason step-by-step about the current situation with the help of past
relevant experiences, then think carefully which admissible action best advances the
shopping goal.
Once you've finished your reasoning, you should choose an admissible action for current
step and present it within <action> </action> tags.

WebShop search guidance:
- Use search[<your query>] with a short core product query, such as the product type or
category.
- Do not put color, size, price, or every requested attribute into search[<your query>].
Handle those by opening a product page and selecting/clicking options when available.
- If a search returns zero results, retry with a shorter broader product query, not a
longer query.
- The goal is to inspect/select a matching product and eventually click[buy now].
```

Executor Prompt (τ^2-bench)

```
<instructions>
You are a customer service agent that helps the user according to the <policy> provided
below.
In each turn you can either:
- Send a message to the user.
- Make a tool call.
You cannot do both at the same time.
Try to be helpful and always follow the policy. Always make sure you generate valid JSON
only.
</instructions>

<policy>
[Content Omitted]
</policy>

Here are past experiences and trajectories that might be helpful for your decision. You
can use it when you feel it's relevant. At each step, first reason about the current
situation with the help of past relevant experiences, including explicitly discuss if
you want to use past experiences or not, and then take action.

{retrieved_context}
```

LLM-as-Judge Prompt (τ^2-bench)

```
You are an expert judge evaluating whether a customer-service agent successfully
satisfied a customer's request. Output a single JSON object and nothing else.
# Task
You will be given (1) the customer's request and (2) the full conversation between the
agent and the customer. The conversation contains [USER] lines (the customer), [AGENT]
lines (the agent's messages, or its tool calls written as fn(arg=value)), and [TOOL]
lines (the tool results). Determine whether the agent fully satisfied the customer's
request.
## What "success" means
- The agent must have actually carried out what the customer asked - the correct action
(e.g. booking, cancellation, update, refund, troubleshooting fix) must be completed via
the appropriate tool call, and the tool result must confirm it succeeded.
- Credit only outcomes that the [TOOL] results confirm. Do not credit effects the agent
merely stated, promised, or planned. Ignore the agent's own claims of completion; rely
on the tool results.
- If the request required following a process (verifying identity/eligibility,
confirming before an irreversible action), that process must be evidenced in the
transcript.
```

```
## Strictness
- If the transcript is ambiguous about whether the request was fully satisfied, output
success=false.
- Partial completion is failure: either the customer's request is fully satisfied or the
conversation is a failure.
- A conversation that ends by giving up, escalating to a human, or hitting the step
limit without completing the request is a failure.
# Output
Output exactly one JSON object with these fields and nothing else:
{ "success": <true|false>, "reasoning": "<one or two sentences citing the specific tool
results that prove success or failure>" }
```

ReasoningBank-style Distillation Prompt for Ablation Study (ALFWorld)

```
You are an expert in household task planning. You will be given a task and a trajectory
representing how an agent successfully completed the task in a household environment.

## Guidelines
Extract and summarize useful insights as memory items that would help an agent solve
similar household tasks in the future.

## Important notes
  - Think about why the trajectory succeeded, then summarize the insights.
  - Extract *at most 3* memory items.
  - Do not repeat similar or overlapping items.
  - Focus on generalizable strategies (e.g., where to find objects, what order to do
actions), not specific object names or locations.

## Output Format
Your output must strictly follow this Markdown format:

'''
# Memory Item i
## Title <short title>
## Description <one sentence summary>
## Content <1-3 sentences of actionable insight>
'''
```

ReasoningBank-style Distillation Prompt for Ablation Study (WebShop)

```
You are an expert in online-shopping task planning. You will be given a task and a
trajectory representing how an agent successfully completed a shopping task on a web
store.

## Guidelines
Extract and summarize useful insights as memory items that would help an agent solve
similar shopping tasks in the future.

## Important notes
  - Think about why the trajectory succeeded, then summarize the insights.
  - Extract *at most 3* memory items.
  - Do not repeat similar or overlapping items.
  - Focus on generalizable strategies, not specific product IDs, prices, or properties.

## Output Format
Your output must strictly follow this Markdown format:

'''
# Memory Item i
## Title <short title>
## Description <one sentence summary>
## Content <1-3 sentences of actionable insight>
'''
```

Prompts. All curator, executor, LLM-as-judge, and distillation prompts are provided in Appendix A. Our executor and judge prompts for ALFWorld and WebShop follow SkillOS (Ouyang et al., 2026a). For τ^2-bench (Barres et al., 2025), we use the executor prompt from the official benchmark and design a separate judge prompt. The ReasoningBank-style distillation prompt used in the ablation study (Section 4.2) is also included.

Table 6: No-memory baseline reproduction. "Reported" from Ouyang et al. (2026a); "Reproduced" from our runs. In all cases, our reproduced performance is at or below the reported results, ensuring gains are measured conservatively.

Source	ALFWorld	WebShop	
	SR	Score	SR
Executor: Qwen3-8B			
SkillOS Reported	$47.9_{1.2}$	$33.3_{0.7}$	$9.8_{0.5}$
Reproduced (non-thinking)	$34.5_{0.3}$	$36.4_{0.2}$	$8.6_{1.1}$
Reproduced (thinking)	$42.1_{1.0}$	$29.0_{0.3}$	$4.4_{0.7}$
Executor: Gemini-2.5-Pro			
SkillOS Reported	$66.4_{2.0}$	$48.6_{0.3}$	$38.4_{0.5}$
Reproduced	$63.6_{1.3}$	$47.3_{0.8}$	$37.5_{0.5}$

Reproducing baselines. Since SkillOS (Ouyang et al., 2026a) does not fully specify its inference settings (e.g., thinking vs. non-thinking mode for Qwen3-8B), we ran preliminary experiments to identify configurations that reproduce their reported no-memory baselines (Table 6). The closest match is thinking mode on ALFWorld and non-thinking mode on WebShop. We remove the default thinking-format instruction from Qwen3-8B executor prompts, as it causes degenerate outputs in non-thinking mode (see the executor prompt in Appendix A). On WebShop, we could not fully reproduce the reported no-memory result with this change alone, so we add a short search-guidance paragraph to the Qwen3-8B executor prompt, inspired by the WebShop evaluation script.[1] As Table 6 shows, we calibrated this guidance so that our no-memory baseline remains at or below the reported result (8.6 vs. 9.8 SR), ensuring our gains are measured conservatively. For Gemini-2.5-Pro and GPT-5.4, we reproduce the reported baselines without modification.

Hyperparameters. Based on the reproduction study above, we adopt the following settings.

Executor. All three executors (Qwen3-8B, Gemini-2.5-Pro, GPT-5.4) use temperature 1.0 and maximum output length 4096. For Qwen3-8B, we use thinking mode on ALFWorld and non-thinking mode on WebShop. Search guidance is added to the Qwen3-8B WebShop executor prompt only. We set a history window of 3 and a maximum of 30 interaction turns for ALFWorld and WebShop, and 200 turns for τ^2-bench.

Curator. For GPT-5.4 and Gemini-2.5-Pro curators, we use temperature 1.0. For both untrained and trained Qwen3-8B curators (non-thinking), we use temperature 0.6 with top-p 0.95 and top-k 20.

Retrieval and batching. We use k=3 retrieved trajectories for all main results; an ablation over $k \in \{3, 5\}$ is provided in Appendix B. The evaluation batch size is 10 for ALFWorld and WebShop and 5 for τ^2-bench.

Training. Full GRPO training hyperparameters are reported in Table 7. During curator training, the Qwen3-8B executor runs in non-thinking mode for efficiency. A single training run takes approximately 21 hours on ALFWorld and 27 hours on WebShop.

Infrastructure. All experiments run on a single server with 8×NVIDIA H200 GPUs, 2×Intel Xeon Platinum 8488C (96 logical cores), and 2 TB system RAM. We serve Qwen3-8B with **vLLM** (Kwon et al., 2023) using tensor parallelism TP=1, data parallelism DP=4, and a maximum model length of 40960.

[1] https://huggingface.co/datasets/zhangdw/webshop/blob/main/evaluate.py

Table 7: RL (GRPO) training hyperparameters.

Hyperparameter	Value
Optimization	
Base policy (curator)	Qwen3-8B (non-thinking)
Advantage estimator	GRPO (no std. normalization)
Learning rate	1×10^{-6}
LR schedule	constant with warmup
Warmup steps	5 (ratio 0.05 of 100 steps)
Train batch size (prompts)	32
Policy mini-batch size	32
Max RL steps	100
Objective	
KL in reward	disabled
KL loss	enabled, low-variance KL, coef. 1×10^{-3}
Clip range (low, high)	$(0.2,\ 0.2)$
Loss aggregation	token-mean
Rollout / generation (curator)	
Samples per prompt (group size)	8
Max prompt length	32768
Max response length	8192 (ALFWorld) / 4096 (WebShop)
Sampling temperature (train)	1.0
Executor / environment	
Executor model	Qwen3-8B (non-thinking), served via vLLM
Executor temperature / top-p / top-k	1.0 / 0.95 / 20
Executor max new tokens	4096
Max env. steps per game	30
Retrieved memories per task	3

B Additional Results

Sensitivity to retrieval count k. Table 8 compares k=3 and k=5 retrieved trajectories on WebShop across all three executors. Performance is stable across both settings: SR varies by less than 2 points in all cases, and the differences remain within standard deviation for Qwen3-8B and GPT-5.4. This indicates that JitMem is not sensitive to the retrieval count, and we use k=3 for all main results for efficiency.

Table 8: Ablation of the number of retrieved trajectories k on WebShop.

Executor	Score	SR
Qwen3-8B		
$k = 3$	$61.1_{0.9}$	$32.8_{1.7}$
$k = 5$	$60.5_{0.4}$	$32.8_{0.3}$
Gemini-2.5-Pro		
$k = 3$	$61.0_{0.8}$	$50.5_{0.8}$
$k = 5$	$59.2_{0.4}$	$48.6_{0.7}$
GPT-5.4		
$k = 3$	$53.8_{0.3}$	$45.4_{0.0}$
$k = 5$	$53.4_{0.7}$	$45.3_{0.8}$

Consolidated ablation results. Table 9 collects all ablation and baseline results from the main text into a single table for easy comparison. For each executor, the table first lists baselines, then JitMem-base ablations, and finally JitMem ablations.

The JITMEM-base ablations isolate individual design choices:

- *w/o task adaptivity*: removes current task description x_t from the curator's input, reducing it to a query-independent summarizer.
- *w/o successful traj. filtering*: stores all trajectories with correctness labels instead of filtering to successful ones.
- *w/o raw traj.*: applies ReasoningBank-style distillation at write time and stores only the distilled items.

The JITMEM ablations probe what RL training learns:

- *w/o retrieved traj.*: forces the retriever to return an empty set, isolating parametric knowledge from episodic retrieval.
- *w/o task adaptivity*: removes current task description x_t from the RL-trained curator.
- *w/ staged bank refresh*: rebuilds the training bank with curator-augmented trajectories after 100 GRPO steps and continues training for 50 more.
- *w/ test bank warm-starting*: pre-populates the test bank with 100 training-set trajectories to mitigate the cold-start effect.

Key findings from Table 9:

1. **Every design choice contributes.** Every JITMEM-base ablation degrades performance, confirming that task-adaptive conditioning, quality-filtered storage, and raw trajectory retention each contribute independently. The largest drop comes from removing raw trajectories on WebShop (up to -8.2 SR), underscoring that write-time distillation discards information the curator needs.
2. **RL learns to distill retrieved experience.** Removing retrieved trajectories from the RL-trained JITMEM causes the largest degradation (up to -14.8 SR on ALFWorld and -15.2 SR on WebShop). Removing task adaptivity also degrades substantially, with the gap widening compared to the untrained variant.
3. **Default training and evaluation settings are sufficient.** Staged bank refresh provides modest gains ($+2.8$ SR at best), and test bank warm-starting has negligible effect, suggesting the static training bank and empty-start evaluation already work well.

Table 9: Consolidated ablation and baseline results on ALFWorld and WebShop across three executors. Each block reports baselines, JITMEM-base ablations (blue), and JITMEM ablations (blue). ❄ denotes prompted curator; 🔥 denotes RL-trained. Mean ± std over 3 runs.

Methods	Curator	ALFWorld	WebShop	
		SR	Score	SR
Executor: Qwen3-8B				
No Memory	—	47.9 $_{1.2}$	33.3 $_{0.7}$	9.8 $_{0.5}$
ReasoningBank	❄ Qwen3-8B	55.7 $_{3.1}$	35.4 $_{1.1}$	11.4 $_{0.9}$
MemP	❄ Qwen3-8B	49.7 $_{0.7}$	35.7 $_{0.9}$	12.0 $_{0.5}$
SkillOS-base	❄ Qwen3-8B	53.1 $_{2.5}$	38.6 $_{0.9}$	13.6 $_{0.8}$
SkillOS-gemini	❄ Gemini-2.5-Pro	50.7 $_{3.6}$	38.1 $_{1.0}$	13.2 $_{0.9}$
SkillOS	🔥 Qwen3-8B	61.2 $_{4.6}$	40.6 $_{0.7}$	16.5 $_{0.7}$
JITMEM-base	❄ Qwen3-8B	60.5 $_{2.6}$	32.5 $_{3.2}$	11.7 $_{0.5}$
w/o task adaptivity	❄ Qwen3-8B	57.4 $_{2.0}$	25.3 $_{0.9}$	7.1 $_{0.1}$
w/o successful traj. filtering	❄ Qwen3-8B	59.0 $_{2.2}$	23.5 $_{2.9}$	8.9 $_{1.7}$
w/o raw traj.	❄ Qwen3-8B	58.8 $_{1.5}$	17.8 $_{1.7}$	4.9 $_{0.7}$
JITMEM	🔥 Qwen3-8B	77.4 $_{2.9}$	61.1 $_{0.9}$	32.8 $_{1.7}$
w/o retrieved traj.	🔥 Qwen3-8B	62.6 $_{1.2}$	47.9 $_{1.6}$	17.6 $_{1.1}$
w/o task adaptivity	🔥 Qwen3-8B	66.0 $_{4.7}$	49.6 $_{1.4}$	22.4 $_{0.4}$
w/ staged bank refresh	🔥 Qwen3-8B + stage 2	–	61.7 $_{0.9}$	35.6 $_{0.3}$
w/ test bank warm-starting	🔥 Qwen3-8B	–	60.9 $_{1.0}$	32.5 $_{1.5}$
Executor: Gemini-2.5-Pro				
No Memory	—	66.4 $_{2.0}$	48.6 $_{0.3}$	38.4 $_{0.5}$
ReasoningBank	❄ Qwen3-8B	71.4 $_{2.9}$	47.1 $_{1.0}$	38.0 $_{0.6}$
ReasoningBank	❄ Gemini-2.5-Pro	78.6 $_{2.9}$	50.8 $_{1.5}$	40.2 $_{1.3}$
MemP	❄ Qwen3-8B	74.3 $_{3.4}$	51.9 $_{1.9}$	40.3 $_{1.3}$
MemP	❄ Gemini-2.5-Pro	77.1 $_{2.1}$	51.3 $_{1.2}$	39.8 $_{1.0}$
SkillOS-base	❄ Qwen3-8B	70.7 $_{3.0}$	52.8 $_{1.0}$	39.6 $_{0.8}$
SkillOS-gemini	❄ Gemini-2.5-Pro	79.3 $_{2.6}$	54.7 $_{1.0}$	41.0 $_{1.2}$
SkillOS	🔥 Qwen3-8B	80.2 $_{3.1}$	56.0 $_{0.7}$	41.3 $_{0.8}$
JITMEM-base	❄ Qwen3-8B	80.0 $_{1.5}$	54.7 $_{1.4}$	44.4 $_{0.6}$
w/o task adaptivity	❄ Qwen3-8B	77.4 $_{2.4}$	51.2 $_{1.3}$	41.8 $_{1.4}$
w/o successful traj. filtering	❄ Qwen3-8B	77.1 $_{3.1}$	52.0 $_{1.0}$	42.1 $_{0.7}$
w/o raw traj.	❄ Qwen3-8B	77.9 $_{2.7}$	45.5 $_{1.0}$	36.5 $_{0.8}$
JITMEM	🔥 Qwen3-8B	86.2 $_{1.9}$	61.0 $_{0.8}$	50.5 $_{0.8}$
w/o retrieved traj.	🔥 Qwen3-8B	79.0 $_{2.6}$	49.6 $_{0.6}$	40.7 $_{0.6}$
w/o task adaptivity	🔥 Qwen3-8B	81.2 $_{2.4}$	56.8 $_{0.4}$	46.1 $_{0.9}$
w/ staged bank refresh	🔥 Qwen3-8B + stage 2	–	61.0 $_{0.3}$	50.5 $_{1.1}$
w/ test bank warm-starting	🔥 Qwen3-8B	–	61.7 $_{0.4}$	50.9 $_{0.3}$
Executor: GPT-5.4				
No Memory	—	62.6 $_{0.3}$	40.9 $_{0.5}$	32.6 $_{0.6}$
ReasoningBank	❄ Qwen3-8B	69.8 $_{4.0}$	37.4 $_{1.8}$	29.5 $_{1.0}$
ReasoningBank	❄ GPT-5.4	77.9 $_{4.2}$	43.1 $_{1.7}$	33.6 $_{1.5}$
MemP	❄ GPT-5.4	72.6 $_{1.7}$	40.8 $_{1.0}$	34.5 $_{1.7}$
SkillOS-base	❄ Qwen3-8B	66.9 $_{1.8}$	39.7 $_{2.2}$	31.0 $_{2.0}$
SkillOS-gpt	❄ GPT-5.4	70.0 $_{3.3}$	33.3 $_{1.1}$	26.9 $_{1.4}$
JITMEM-base	❄ Qwen3-8B	79.3 $_{3.6}$	49.9 $_{1.0}$	39.3 $_{0.7}$
w/o task adaptivity	❄ Qwen3-8B	78.3 $_{6.0}$	44.2 $_{0.5}$	34.8 $_{0.8}$
w/o successful traj. filtering	❄ Qwen3-8B	77.6 $_{1.8}$	45.5 $_{0.5}$	35.9 $_{0.5}$
w/o raw traj.	❄ Qwen3-8B	76.4 $_{1.5}$	39.3 $_{0.9}$	31.1 $_{1.0}$
JITMEM	🔥 Qwen3-8B	86.7 $_{0.7}$	53.8 $_{0.3}$	45.4 $_{0.0}$
w/o retrieved traj.	🔥 Qwen3-8B	72.9 $_{0.6}$	50.0 $_{0.4}$	40.9 $_{0.4}$
w/o task adaptivity	🔥 Qwen3-8B	77.4 $_{0.9}$	48.0 $_{0.6}$	37.5 $_{0.4}$
w/ staged bank refresh	🔥 Qwen3-8B + stage 2	–	53.6 $_{0.4}$	46.3 $_{0.1}$
w/ test bank warm-starting	🔥 Qwen3-8B	–	51.9 $_{0.3}$	44.1 $_{0.4}$

Example payloads. Below we show one curated payload per benchmark, each pairing a task with the payload the curator synthesized for it. Across all three, the curator produces task-specific guidance from retrieved trajectories. On ALFWorld, it assembles a clean-and-place procedure from three partially relevant episodes. On WebShop, it converts searches for different products into a search-phrasing and attribute-selection strategy for the current one. On τ^2-bench, it distills several MMS failure cases into an ordered diagnostic procedure, separating a general resolution strategy from guidance specialized to the current request. Notably, the curator carries over the domain policy that the mutating tool requires explicit user approval.

ALFWorld (Task type: Clean)

Task: Put a clean plate in countertop
– – – – – – – – – – – – Payload generated by JITMEM – – – – – – – – – – – –

Briefing for Task: Put a clean plate in countertop

Relevant Memories:
– Memory 1: Successfully cooled and placed a plate on a countertop.
– Memory 2: Cleaned and placed a tomato on a countertop.
– Memory 3: Cleaned and placed a soapbar in a toilet (cleaning process is similar).

Key Strategies:
1. **Locate the plate:** Check countertops first, as plates are often found there.
2. **Clean the plate:** Use a sinkbasin to clean it if it's dirty.
3. **Move the plate:** Once clean, place it on a desired countertop.

Specific Guidance:
1. **Go to a countertop** (e.g., countertop 1 or 2) to find the plate.
2. **Take the plate** from the countertop.
3. **Go to the sinkbasin** and clean the plate if it's dirty.
4. **Return to a countertop** and place the clean plate there.

WebShop

Task: Find me high speed flashes with usb port with color: usb256-pink, and size: 512gb, and price lower than 40.00 dollars.
– – – – – – – – – – – – Payload generated by JITMEM – – – – – – – – – – – –

Briefing:

Target Product: High speed flashes with usb port, color: usb256–pink, size: 512gb, price lower than $40.00

Relevant Past Experiences:
– Memory 1 and **Memory 2** involved searching for men's t–shirts with specific color and size constraints, and included steps to select the correct color and size options. These experiences are relevant as they demonstrate how to navigate color and size options effectively.
– Memory 3 involved searching for men's gym shorts with specific color and size constraints, and included steps to select the correct color and size options. This experience is also relevant as it shows how to handle color and size selection in a similar context.

Strategies from Past Experiences:
1. **Search Phrasing:** Use precise search terms that include the product type, key features, color, size, and price constraint. For example, "high speed flashes with usb port usb256–pink 512gb under $40".
2. **Color Selection:** Click on the exact color option from the color palette to select the desired color.
3. **Size Selection:** Click on the exact size option from the size options to select the desired size.

4. **Price Constraint:** Ensure the final product price is within the specified budget before purchasing.

Actionable Guidance:
1. **Search:** Begin with a search query that includes the product type, key features, color, size, and price constraint. For example, "high speed flashes with usb port usb256−pink 512gb under 40".
2. **Navigate Results:** Click on the product that matches your criteria.
3. **Select Color:** From the color options, click on "usb256−pink".
4. **Select Size:** From the size options, click on "512gb".
5. **Verify Price:** Ensure the final price is within the specified budget before proceeding to purchase.

τ^2-bench (Domain: Telecom)

Task: You are unable to send MMS messages using your messaging app for the past few hours. You want to fix it and successfully send an MMS message.
− − − − − − − − − − − − Payload generated by JITMEM − − − − − − − − − − − −

Most relevant memories: **Memory 1 and Memory 3** (same MMS issue, including travel/roaming and possible data−cap blockage). **Memory 2** is also useful for the domestic case where app permissions/network mode were the fix.

Resolution strategy that worked:
1. **Identify the customer/line**
   – Use ‘get_customer_by_phone‘ (or equivalent customer lookup).
   – If multiple lines exist, confirm the exact phone number/line affected.

2. **Gather device/network state first**
   – Ask user to run/read: ‘check_network_status‘, ‘can_send_mms‘, ‘check_apn_settings‘, ‘check_wifi_calling_status‘, and app permission checks (use ‘check_installed_apps‘ first if app name is unclear).
   – If abroad, specifically verify **mobile data ON**, **data roaming ON**, and usable network service (not airplane mode / no SIM / 2G−only).

3. **Guide user through non−account fixes in order**
   – If airplane mode is on: turn it off.
   – If SIM missing: reseat SIM.
   – If mobile data or roaming is off: turn them on.
   – If stuck on poor/2G service: set preferred network mode to ‘4g_5g_preferred‘.
   – If messaging app lacks permissions: grant **sms** and **storage**.
   – If needed, check/reset APN and reboot.
   – If Wi−Fi Calling is on and MMS still fails, try turning it off.

4. **Check account−side blockers only after device state is good**
   – Look up the correct line with ‘get_details_by_id‘.
   – Verify line is active and whether roaming is enabled on the account.
   – Use ‘get_data_usage‘ and plan lookup to see if the line is over its data limit.

5. **If over data limit and MMS depends on mobile data**
   – Explain the overage and available options.
   – Confirm allowed refuel amount and total charge.
   – Obtain explicit user approval.
   – Apply with ‘refuel_data‘.

Guidance for this request:

– Start by identifying the customer/line, then immediately collect current device state with:
  – ‘check_network_status‘
  – ‘can_send_mms‘
  – ‘check_apn_settings‘
  – ‘check_wifi_calling_status‘
  – ‘check_installed_apps‘ –> ‘check_app_permissions‘ for the actual messaging app
– Based on results, prioritize:
  1. restore basic cellular service (airplane mode/SIM/signal/network mode),
  2. ensure mobile data is enabled,
  3. if roaming is relevant, ensure device roaming is on and account roaming is enabled,
  4. fix app permissions (‘sms‘, ‘storage‘),
  5. APN reset/reboot if still failing.
– If the device shows good service/data and MMS still fails, check the account line and **data usage**; if at/over limit, offer a **data refuel** only after confirming price and receiving explicit approval before calling the mutating tool.
– Do **not** assume the same line ID, roaming status, or data usage from memory–look up the current case.

Training curves. Figures 5 and 6 show GRPO training progress on ALFWorld and WebShop, with validation on a subsampled test stream. On both benchmarks, validation success rate climbs steadily over 100 training steps while executor turns per task fall, indicating the curator learns to produce payloads that improve task success while reducing the number of executor steps. Training is stable throughout under a single task reward, with no auxiliary content-quality reward, no task grouping, and no return shaping — supporting the claim that read-time curation makes curator learning straightforward.

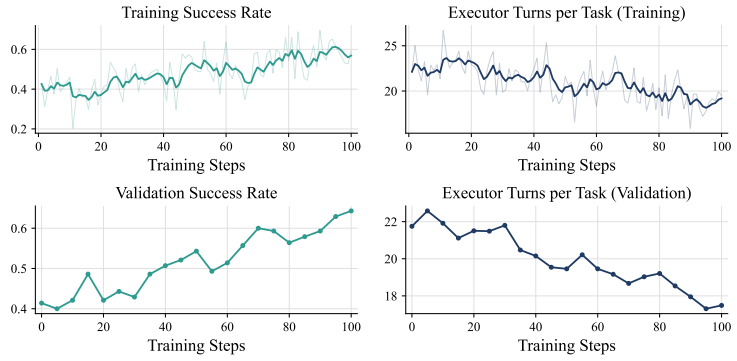


Figure 5: GRPO training curves on ALFWorld (Qwen3-8B executor). Top row: training SR and executor turns. Bottom row: the same metrics on validation. SR rises and turns fall steadily over 100 steps.

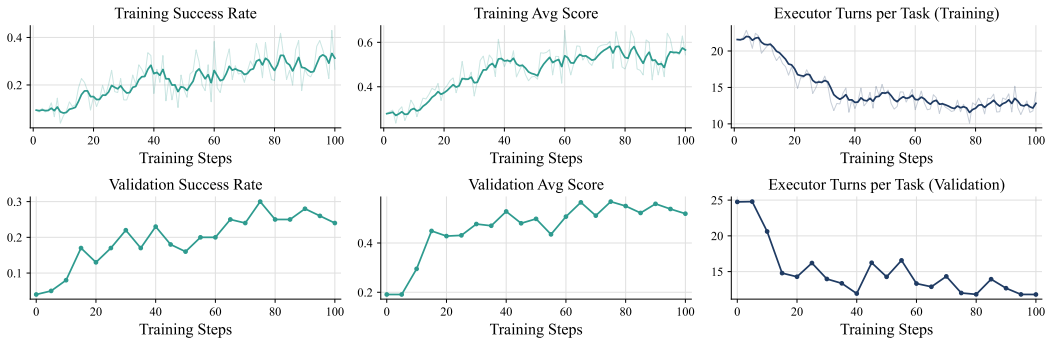


Figure 6: GRPO training curves on WebShop (Qwen3-8B executor). Top row: training SR, training score, training executor turns. Bottom row: the same three metrics on validation.