GENMEM: GENERATIVE SYMBOLIC MEMORY FOR SELF-EVOLVING HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term memory supports the self-evolution of LLM agents by retaining experience and skills across tasks and enabling their *retrieval, reuse*, and *revision* in subsequent long-horizon decision-making. Yet existing memory management approaches remain limited to discriminative retrieval and to address the sparse, hierarchical, and highly redundant structure of reusable experience: only a small, task-dependent subset of trajectories and memories warrants retention, retrieval, or revision. Learning these operations is further complicated by ***sparse, delayed***, and ***indirect*** task-level feedback, with weak supervision across the memory lifecycle. Moreover, continual memory evolution introduces an architectural tension as ***addressing invariance***: stored experience is perpetually revised, yet the addressing interface consumed by learned retrieval policies must remain stable. To address, we present **GENMEM**, which reformulates memory management as generative symbolic addressing. Its core mechanism is the **Symbolic Identifier (SID)**, a multi-level discrete token tuple drawn from a Cartesian-product address space that factorizes a million-scale sparse memory space using fewer than one hundred discrete symbols. Instead of generating ever-changing raw content, the memory agent learns to generate SIDs, while memory evolution rewrites the payload at a fixed address without shifting the address itself. Architecturally, **GENMEM** couples a MemRetriever and a MemEvolver within a multi-agent harness, trained via GRPO with dense process and outcome rewards with two-channels optimization. Under offline memory evolution, experiments spanning ALFWorld, WebShop, multi-hop QA, medical reasoning, and deep research evaluate **GENMEM** against strong memory-augmented baselines. Beyond accuracy, we find that SIDs exhibit emergent ***neuro-symbolic properties***: they can be interleaved with natural-language tokens and are ***compositional***—combining multiple SIDs composes their stored experiences to address novel situations not covered by any single entry.

1 INTRODUCTION

Long-term memory enables self-evolving LLM agents to retain reusable experience and skills across tasks, supporting long-horizon decision-making (Wang et al., 2023; Hong et al., 2023; Zhang et al., 2026c). Unlike encyclopedic knowledge, such experience-centric memory is action-oriented, context-conditioned, and continuously evolving (Zhang et al., 2024). The central question therefore shifts from *which memory most resembles a query* to *which experience best advances the agent's current strategy*—a question that must remain answerable even as the experiences themselves evolve. Despite promising progress (Zheng et al., 2026; Zhang et al., 2025; 2026b), existing memory-augmented agents (Chhikara et al., 2025; Xu et al., 2025) remain largely anchored to a **dense retrieval paradigm**: they access experience through discriminative scoring, with learned addressing coupled to content representations. Despite these advances, two fundamental challenges remain:

❶ ***Sparsity Challenge.*** Experience is highly redundant: across the memory lifecycle (acquisition, organization, retrieval, and evolution), only a small fraction carries reusable value; only a narrow region is relevant to any given query; and refinement is concentrated on repeatedly exercised entries. The experiences that generalize tend to cluster around shared sub-skills—for instance, distinct web-navigation tasks may all reuse a "search-then-filter" routine—yet flat dense stores do not explicitly encode this structure, treating entries uniformly and diluting key associations as the bank grows (Zhang et al., 2024). Appendix B.1 details these sparsity dimensions. Supervision compounds

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.

the problem: an experience's utility emerges only through delayed, indirect task outcomes, so reliance on sparse terminal rewards (Zhang et al., 2026b; Ma et al., 2026) makes it difficult to assign supervision to individual memory operations.

❷ ***Address Variance Challenge.*** A deeper architectural tension arises when experience continuously evolves (Figure 1): **the content at an address changes, but the address itself must remain stable.** A self-evolving agent refines SOPs, merges redundant skills, and discards outdated procedures—yet the memory system must remain reliably addressable throughout. When retrieval relies on representations derived from current memory content, each revision can silently change which queries reach an entry, even when the entry still serves the same functional role (Appendix B.2). We term the requirement to preserve an entry's address across such revisions ***addressing invariance***. Satisfying it demands decoupling *where* from *what*: stable discrete coordinates identify functional regions of experience space, while text behind those coordinates remains freely editable. This decoupling reframes what agent should learn: not particular content versions that become outdated with each revision, but a persistent mapping from a problem to **functional role** of required experience.

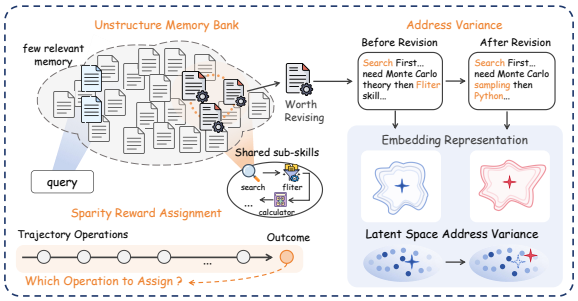

Figure 1: Two challenges in evolving experience memory. **Left:** Reusable experience is sparse and organized around shared sub-skills, while delayed task rewards hinder credit assignment. **Right:** Content updates can shift embedding-based retrieval, motivating stable addresses that remain invariant as experience evolves.

Our key motivation draws on a recurring principle: ***structured and sparse architectures exploit hidden regularities in high-dimensional spaces***—LoRA through low-rank updates, MoE through sparse activation, and Engram through conditional knowledge storage (Hu et al., 2022; Fedus et al., 2022; Cheng et al., 2026). Analogously, reusable experience concentrates in **sparse, hierarchically structured regions** amid substantial redundancy, motivating selective addressing that remains stable as content evolves. We therefore propose **GENMEM**, a **generative symbolic memory framework** that unifies memory access and update targeting through **Symbolic Identifier (SID)** generation. Each SID is a hierarchical discrete token tuple encoding **mutually orthogonal categorical axes** (*e.g.*, task domain, tool modality, and strategy pattern), decomposing the vast experience space within a Cartesian-product address space. Constructed offline via context-aware RQ-KMeans, SIDs organize functionally proximate experiences through shared prefixes (Lee et al., 2022; Rajput et al., 2023). The memory agent generates target SIDs from its current context for selective access, while online revisions update payloads in place, preserving existing SID assignments under fixed codebooks.

Concretely, **GENMEM** couples two memory agents in a closed loop: **MemRetriever** generates a SID and rewrites the retrieved experience into task-adaptive support; 1. a multi-agent harness executes the task using this support; and **MemEvolver** distills the resulting trajectory and generates an update SID for insertion or revision. The pipeline comprises alignment pretraining, mid-training, GRPO post-training, and online self-evolution. The first two stages teach SID generation and its integration with reasoning; GRPO combines process and outcome rewards with two-channels optimization, providing fine-grained supervision beyond sparse terminal feedback. During online self-evolution, the agents continue refining memory through the same stable SID interface with model parameters and codebooks fixed. Our contributions are fourfold:

- We propose **GENMEM**, the first generative symbolic addressing framework for evolving experience memory, unifying memory retrieval and evolution through SID generation over a sparse, hierarchically structured space (Section 2).
- We instantiate this framework as a closed-loop system coupling MemRetriever, a multi-agent harness, and MemEvolver, with staged training using process and outcome rewards with two-channel optimization to support online memory evolution (Sections 2–3).
- Experiments across embodied planning, web navigation, search-augmented QA, medical reasoning and deep research evaluate **GENMEM** against strong memory-agent baselines (Section 4).

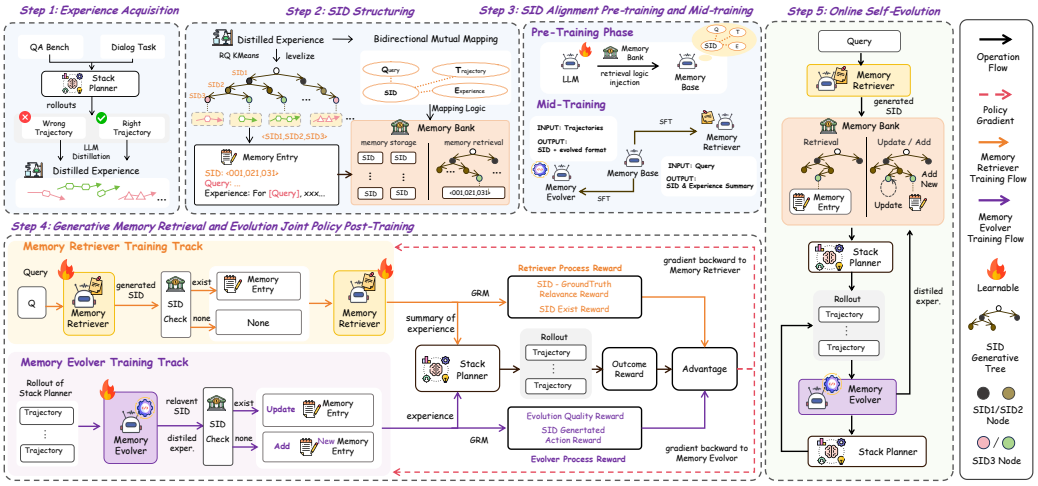

Figure 2: Overview of **GENMEM**.

- We identify an emergent neuro-symbolic property that SIDs can be manipulated like natural language and exhibit compositionality that combining multiple SIDs enabling agent to address novel situations not covered by any individual memory entry (Section 5).

2 GENMEM: SYMBOLIC MEMORY ADDRESSING SYSTEM

GENMEM unifies memory retrieval and evolution through a shared symbolic address space, enabling selective access to structured experience while preserving stable addresses under content updates.

❶ ***Memory Symbolic Identifier Definition.*** Let $\mathcal{M} = \{\mathbf{m}_i\}_{i=1}^{\mathbf{N}}$ denote the external memory bank, where each entry $\mathbf{m}_i = (\mathbf{x}_i, \mathbf{z}_i)$ pairs a textual experience payload $\mathbf{x}_i$ with a Symbolic Identifier (SID) as $\mathbf{z}_i$. We define each SID as a tuple of L discrete symbols as:

$$\mathbf{z}_i = (\mathbf{s}_i^1, \ldots, \mathbf{s}_i^L) \in \mathcal{Z}, \quad \mathcal{Z} = [\mathbf{N}_1] \times \cdots \times [\mathbf{N}_L], \quad \mathbf{s}_i^\ell \in [\mathbf{N}_\ell], \tag{1}$$

where $[\mathbf{N}_\ell] = \{1, \ldots, \mathbf{N}_\ell\}$ indexes the symbols at level ℓ. Given a context $\mathbf{c}$, a symbolic addressing policy generates $\mathbf{z} = \textbf{ADDR}_\theta(\mathbf{c})$, and lookup returns the associated payload $\mathbf{x} = \textbf{LOOKUP}(\mathcal{M}, \mathbf{z})$. The context $\mathbf{c}$ may be a query, a trajectory, or a newly distilled experience, allowing the same address space to support retrieval and update targeting.

In **GENMEM**, SIDs represent $\prod_{\ell=1}^{L} \mathbf{N}_\ell$ possible addresses using $\sum_{\ell=1}^{L} \mathbf{N}_\ell$ level-specific symbols, with shared prefixes organizing experience hierarchically for selective access. Crucially, a SID identifies a memory slot rather than a particular content version: online revisions transform $(\mathbf{x}_i, \mathbf{z}_i) \xrightarrow{Evolve} (\mathbf{x}_i', \mathbf{z}_i)$, allowing experience to evolve while its address remains stable.

❷ ***Experience Acquisition and SID Construction.*** We construct the initial memory bank through a *gradient-free cold start*. Across datasets spanning diverse task domains, the multi-agent harness StackPlanner (Zhang et al., 2026a) generates multiple trajectories for each training query. An LLM extractor compares correct, failed, and unverified attempts using available reference answers or environment feedback. It distils reusable procedures and failure lessons while preserving uncertainty where the traces lack evidence. The resulting summaries form the seed set $\mathcal{X}_{\text{seed}}$ (Appendix D.1).

To obtain discrete addresses that the memory agents can generate, we quantize continuous experience representations into SID sequences. First, we encode each summary $\mathbf{x}_i$ via a frozen text encoder:

$$\mathbf{e}_i = f_{\text{enc}}(p_{\text{enc}} \oplus \mathbf{x}_i) \in \mathbb{R}^d, \tag{2}$$

where the prefix p_{enc} emphasizes reusable strategies, applicability conditions, and procedures. This semantic embedding $\mathbf{e}_i$ encourages the resulting addresses to reflect shared problem-solving patterns rather than surface-level textual similarity.

We then construct the discrete address space by fitting RQ-KMeans to the seed embeddings (Deng et al., 2025): At the first level, K-means clusters the embeddings into $\mathbf{N}_1$ groups; the resulting cluster centers form the first ***codebook***. Each embedding is assigned to its nearest center, whose index becomes the first SID symbol. We then subtract this center from the embedding and fit the next codebook to the remaining residuals. Repeating this procedure yields L codebooks, with successive levels capturing information not represented by earlier ones. Formally, let $\mathcal{C}^{(\ell)} = \{\mathbf{c}_j^{(\ell)}\}_{j=1}^{N_\ell}$ denote the resulting cluster centers at level ℓ. Starting from $\mathbf{r}_i^{(1)} = \mathbf{e}_i$, the Euclidean form of code assignment and residual updating is

$$\mathbf{s}_i^\ell = \arg\min_{j \in [\mathbf{N}_\ell]} \left\| \mathbf{r}_i^{(\ell)} - \mathbf{c}_j^{(\ell)} \right\|_2^2, \quad \mathbf{r}_i^{(\ell+1)} = \mathbf{r}_i^{(\ell)} - \mathbf{c}_{\mathbf{s}_i^\ell}^{(\ell)}, \quad \ell = 1, \ldots, L. \tag{3}$$

The sequence of selected center indices $\mathbf{z}_i = (\mathbf{s}_i^1, \ldots, \mathbf{s}_i^L)$ forms the SID. Shared prefixes group experiences with common quantized representations, while later symbols provide finer distinctions. This yields structured addresses without requiring a separate symbol for every memory entry. In **GENMEM**, we use codebook sizes $(48, 16, 8, 8)$, yielding $49{,}152$ possible addresses from 80 tokens, and summarize multiple experience texts assigned to the same SID into a single memory entry. The codebooks are fixed after construction. Appendix C.1 reports the codebook comparisons and the three-level prefix-prompt experiment, distinguishing these settings from Euclidean formulation above.

❸ ***Generative Memory Retrieval and Evolution.*** Given the SID-indexed bank $\mathcal{M}$, **MemRetriever** and **MemEvolver** implement generative read–write operations around StackPlanner (Figure 2).

- ***Memory Retrieval.*** MemRetriever generates an address $\mathbf{z}_r$ from the query and reasoning context, retrieves the associated experience, and rewrites it into task-adaptive support $\tilde{\mathbf{x}}_r$.
- ***Memory Execution.*** StackPlanner executes the task using this support, producing an answer $\hat{y}$ and an execution trajectory τ.
- ***Memory Evolution.*** MemEvolver distils the trajectory, generates an update address $\mathbf{z}_e$, and consults the associated entry to determine an operation o and updated payload $\tilde{\mathbf{x}}_e$.

For a query q, the closed memory loop comprises three steps:

$$\begin{aligned} \textit{Retrieve:} \quad & (\mathbf{z}_r, \tilde{\mathbf{x}}_r) = \textbf{MEMRETRIEVER}(q, \mathcal{M}), \\ \textit{Execute:} \quad & (\hat{y}, \tau) = \textbf{STACKPLANNER}(q, \tilde{\mathbf{x}}_r), \\ \textit{Evolve:} \quad & \mathcal{M}' = \textbf{UPDATE}(\mathcal{M}; \textbf{MEMEVOLVER}(\tau, \mathcal{M})) \,. \end{aligned} \tag{4}$$

Here, MemEvolver returns an update tuple $(o, \mathbf{z}_e, \tilde{\mathbf{x}}_e)$, specifying the operation, target SID, and updated payload. The operation $o \in \{\texttt{insert}, \texttt{revise}\}$ either adds a new entry or updates an existing one. Revision supports retention, consolidation, and deletion through unchanged, merged, or empty payloads, respectively.

3 MEMORY POLICY OPTIMIZATION AND SELF-EVOLUTION

Building on the symbolic addressing system, we develop MemRetriever and MemEvolver through four stages: SID alignment pretraining, memory-operation mid-training, joint policy optimization, and online self-evolution. The first three stages teach the agents to generate, use, and update symbolic memory; the final stage applies these learned policies to continually evolve the memory bank.

❶ ***SID Alignment Pretraining.*** We first align task contexts and experience content with the constructed SID space through five text–SID mappings. The forward mappings $q \mapsto \mathbf{z}$, $\tau \mapsto \mathbf{z}$, and $\mathbf{x} \mapsto \mathbf{z}$ teach address generation, while the reverse mappings $\mathbf{z} \mapsto \mathbf{x}$ and $\mathbf{z} \mapsto \mathbf{d}$ associate SIDs with experience content and natural-language semantic descriptions $\mathbf{d}$, respectively. For an input context $\mathbf{c}$ and target SID $\mathbf{z} = (s^1, \ldots, s^L)$, the forward loss and combined alignment objective are

$$\mathcal{L}_{\text{SID}}(\mathbf{c}, \mathbf{z}) = -\sum_{\ell=1}^{L} \log p_\theta(s^\ell \mid \mathbf{c}, s^{1:\ell-1}), \quad \mathbf{c} \in \{q, \tau, \mathbf{x}\}, \tag{5}$$

$$\mathcal{L}_{\text{pre}} = \lambda_q \mathcal{L}_{q \to \mathbf{z}} + \lambda_\tau \mathcal{L}_{\tau \to \mathbf{z}} + \lambda_x \mathcal{L}_{\mathbf{x} \to \mathbf{z}} + \lambda_{\mathbf{z} \to \mathbf{x}} \mathcal{L}_{\mathbf{z} \to \mathbf{x}} + \lambda_{\mathbf{z} \to \mathbf{d}} \mathcal{L}_{\mathbf{z} \to \mathbf{d}}, \tag{6}$$

where the coefficients weight the contributions of the respective mappings specified in Appendix B.5. This stage establishes bidirectional associations between symbolic addresses and experience, providing the pretrained checkpoints for subsequent MemRetriever and MemEvolver training.

❷ ***Memory-Operation Mid-Training.*** SID alignment establishes associations between symbolic addresses and experience, but effective memory use requires the model to incorporate these addresses into reasoning and action. We therefore construct *ReAct-style* trajectories to train MemRetriever and MemEvolver separately from the pretrained checkpoints, teaching each policy to reason about memory, invoke SID-based operations, and use the returned experience. Specifically,

- For **MemRetriever**, role-specific prompts guide it to reason about the current task, generate a retrieval SID, inspect the retrieved experience, and rewrite it into task-adaptive support.
- For **MemEvolver**, the prompts guide the model to analyze an execution trajectory, identify transferable lessons, generate an update SID, and revise addressed experience or propose a new entry.

We apply *rejection sampling* to retain trajectories that satisfy interaction format and task-correctness criteria. Then we apply a stronger teacher model to provide distillation targets for the reasoning traces and role-specific outputs: experience rewriting for MemRetriever and experience evolution for MemEvolver. Using a standard next-token objective, we fine-tune each policy on the resulting demonstrations, turning isolated SID prediction into executable memory read–write behavior.

❸ ***Joint Policy Post-Training.*** To align memory operations with downstream objectives and alleviate sparse feedback, we jointly optimize MemRetriever and MemEvolver from their mid-trained initialization. For both agents, the process reward evaluates SID accuracy and the strategic quality of generated experience. The outcome reward supplies their rewritten or evolved experience to StackPlanner and measures downstream task performance:

- **Process reward.** We evaluate memory operations through
$$R_{\text{process}} = R_{\text{format}} + R_{\text{strategy}} + R_{\text{SID}}, \tag{7}$$
where R_{format} checks SID validity, R_{SID} rewards code-level agreement with the ground-truth SID, and R_{strategy} uses the backbone as an LLM judge with privileged access to $y^\star$ to assess the strategic usefulness of the rewritten or evolved experience.
- **Outcome reward.** We compare StackPlanner's answers with and without memory support through a contrastive reward setting:
$$R_{\text{out}} = r_{\text{task}}(\hat{y}_{\text{mem}}, y^\star) - r_{\text{task}}(\hat{y}_{\text{no-mem}}, y^\star) + \alpha_{\text{partial}}\,\mathbb{I}[\text{both answers are correct}], \tag{8}$$
where $\alpha_{\text{partial}} > 0$ provides a partial-credit bonus only when both executions succeed.

For optimization, we adopt **GRPO** (Shao et al., 2024), sampling G outputs $\{u_i\}_{i=1}^{G}$ per context and normalizing advantages separately for the proxy and audit channels. For either channel, we minimize

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|u_i|}\sum_{t=1}^{|u_i|}\Big(\min\big(\rho_{i,t}\hat{A}_i, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i\big) - \beta D_{\text{KL}}^{i,t}\Big)\right], \tag{9}$$

where $\rho_{i,t}$ is the token-level importance ratio relative to the rollout policy, $\hat{A}_i$ is the group-normalized reward from $R_{\text{out}} + R_{\text{process}}$, and $D_{\text{KL}}^{i,t}$ penalizes divergence from the reference policy. Reward components and implementation details to verify are discussed in Appendix C.3–C.3.3.

❹ ***Online Memory Self-Evolution.*** After training, model parameters, encoder, and SID codebooks are frozen. The three modules form a closed loop on a stream of new queries: MemRetriever generates a SID and fetches task-adaptive support, StackPlanner executes, and MemEvolver generates an update address and distils the trajectory into an updated payload. An `insert` populates an unoccupied SID slot, whereas a `revise` updates an existing entry. In both cases, the target SID is generated by MemEvolver, not recomputed by encoding and quantizing the updated payload. We run this loop for T steps until the memory bank converges. Adaptation proceeds entirely through external memory: payloads are added or revised within the fixed SID space, requiring neither gradient updates nor reassignment of existing addresses. The update rule and design rationale are provided in Appendices C.2 and C.4, respectively.

4 Experiments

We evaluate **GenMem** through two research questions: **(RQ1)** Does **GenMem** improve downstream task performance compared with existing memory-augmented and RL-based methods? **(RQ2)** Can online memory evolution improve performance over time while maintaining stable SID addressing?

4.1 EXPERIMENTAL SETUP

❶ Datasets and Metrics. We evaluate **GENMEM** on interactive decision-making and search-augmented question answering. A separate GPQA experiment examines the contribution of symbolic SID information (Section 5).

- **ALFWorld** (Shridhar et al., 2021): a household environment that evaluates sequential decision-making under natural-language instructions across Pick, Look, Clean, Heat, Cool, and Pick2 tasks. For ALFWorld, we report per-subtask and overall success rates (%).
- **WebShop** (Yao et al., 2022): a simulated e-commerce environment in which agents search for, inspect, and purchase products that satisfy user-specified requirements. For WebShop, we report task score and success rate (%).
- **Search-augmented QA**: following SkillRL (Xia et al., 2026), we evaluate on six question-answering benchmarks, using 2WikiMultiHopQA and HotpotQA for in-domain evaluation and Bamboogle, MuSiQue, NQ, and TriviaQA for out-of-domain evaluation. For search-augmented QA, we report F1 (%).
- **Deep Research, Code, and SQL**: we further evaluate whether experience acquired in the source domains transfers to **out-of-domain tasks** involving long-horizon information seeking over web and document sources, code generation, and SQL query generation; see Appendix E.1.

Results for ALFWorld and WebShop are in Table 1, and search-augmented QA results in Table 2. Benchmark coverage, comparison settings, and metric definitions are in Appendix E.1–E.2.

❷ Baselines. We compare against the methods that appear in the two main result tables:

- **Closed-source LLMs**: GPT-4o and Gemini-2.5-Pro, representing frontier proprietary capabilities.
- **Prompt-based agentic / memory methods**: ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), Mem0 (Chhikara et al., 2025), MemP (Fang et al., 2025), ExpeL (Zhao et al., 2024), SimpleMem (Liu et al., 2026), IRCOT (Trivedi et al., 2023), and TCRAG (Jiang et al., 2025).
- **RL-based methods**: RLOO (Ahmadian et al., 2024) and GRPO (Shao et al., 2024) for the embodied and web-task experiments; ReSearch (Chen et al., 2025a), Search-R1 (Jin et al., 2025), AEPO (Dong et al., 2025a), ARPO (Dong et al., 2025b), MEM1 (Zhou et al., 2025), ZeroSearch (Sun et al., 2025), and AgenticRAG-R1 (Jiang et al., 2026a) for search-augmented QA.
- **Memory-augmented RL methods**: MemRL (Zhang et al., 2026b), EvolveR (Wu et al., 2025), Mem0+GRPO (Chhikara et al., 2025), SimpleMem+GRPO (Liu et al., 2026), SkillRL (Xia et al., 2026), SkillGraph (Li et al., 2026), AgentOCR (Feng et al., 2026), and Skill0 (Lu et al., 2026b), representing the current state of the art included in our comparison tables.
- **Skill-based Search methods**: EvolveR, SkillRL, SkillGraph, AgentOCR, Skill0, Skill-SD (Wang et al., 2026b), SAPO (Zhang et al., 2026d), and SDAR (Lu et al., 2026a) for search-augmented QA.

❸ Implementation and Training. StackPlanner, MemRetriever, and MemEvolver all use *Qwen2.5-7B-Instruct* (Bai et al., 2023) as the backbone LLM. We construct four SID codebooks using context-aware RQ-KMeans with sizes $(48, 16, 8, 8)$, yielding $49{,}152$ possible leaf addresses. The memory modules undergo SID alignment pretraining for $17{,}465$ steps with a batch size of 48, followed by supervised mid-training on ReAct-style CoT traces for 157 steps with an effective batch size of 32. Both stages use cross-entropy loss and AdamW with cosine learning-rate decay, with initial learning rates of 1×10^{-5} and 2×10^{-6}, respectively. We then jointly optimize MemRetriever and MemEvolver with GRPO for 130 steps. For StackPlanner, we train the policy model by adopting Harness-RL (Jiang et al., 2026b) to train the planning policy with GRPO in a blackbox-Harness manner. Memory-module training uses eight NVIDIA A100 80GB GPUs with DeepSpeed ZeRO-3, and rollouts are generated using vLLM. Further training details are provided in the appendix, including oracle-assisted StackPlanner diagnostics in Appendix G.3.

4.2 MAIN RESULTS (RQ1)

❶ Experience Retrieval. Table 4 shows that **GENMEM** achieves the highest candidate and final Top-1/Top-5 hit rates. Under the same candidate budget and BGE reranking protocol, its final Top-5 hit rate reaches 13.15%, compared with 10.28% for SkillRouter, 12.52% for TF-IDF, and 2.89% for Qwen3-Embedding-0.6B. These results highlight the limitations of similarity-based skill retrieval: useful experience must **match a task's strategic** needs, which textual similarity alone may not

Table 1: **Main results on ALFWorld and WebShop.** ALFWorld reports per-subtask and overall success rates (%); WebShop reports task score and success rate (%).

	ALFWorld Success Rate (%)							WebShop	
Method	**Pick**	**Look**	**Clean**	**Heat**	**Cool**	**Pick2**	**All**	**Score**	**Succ.**
Closed-source LLMs (absolute scores)									
GPT-4o	75.3	60.8	31.2	56.7	21.6	49.8	48.0	31.8	23.7
Gemini-2.5-Pro	92.8	63.3	62.1	69.0	26.6	58.7	60.3	42.5	35.9
Prompt-based Agentic or Memory-based Methods (Δ vs. ReAct)									
ReAct	48.5 Δ 0.0	35.4 Δ 0.0	34.3 Δ 0.0	13.2 Δ 0.0	18.2 Δ 0.0	17.6 Δ 0.0	31.2 Δ 0.0	46.2 Δ 0.0	19.5 Δ 0.0
Reflexion	62.0 Δ +13.5	41.6 Δ +6.2	44.9 Δ +10.6	30.9 Δ +17.7	36.3 Δ +18.1	23.8 Δ +6.2	42.7 Δ +11.5	58.1 Δ +11.9	28.8 Δ +9.3
Mem0	54.0 Δ +5.5	55.0 Δ +19.6	26.9 Δ -7.4	36.4 Δ +23.2	20.8 Δ +2.6	7.69 Δ -9.9	33.6 Δ +2.4	23.9 Δ -22.3	2.00 Δ -17.5
MemP	54.3 Δ +5.8	38.5 Δ +3.1	48.1 Δ +13.8	56.2 Δ +43.0	32.0 Δ +13.8	16.7 Δ -0.9	41.4 Δ +10.2	25.3 Δ -20.9	6.40 Δ -13.1
ExpeL	21.0 Δ -27.5	67.0 Δ +31.6	55.0 Δ +20.7	52.0 Δ +38.8	11.0 Δ -7.2	6.00 Δ -11.6	46.3 Δ +15.1	30.9 Δ -15.3	11.2 Δ -8.3
SimpleMem	64.5 Δ +16.0	33.3 Δ -2.1	20.0 Δ -14.3	12.5 Δ -0.7	33.3 Δ +15.1	3.84 Δ -13.8	29.7 Δ -1.5	33.2 Δ -13.0	8.59 Δ -10.9
GENMEM	91.7 Δ +43.2	83.3 Δ +47.9	71.0 Δ +36.7	56.5 Δ +43.3	85.7 Δ +67.5	58.8 Δ +41.2	74.6 Δ +43.4	52.0 Δ +5.8	27.0 Δ +7.5
RL-based Methods (Δ vs. MemRL)									
RLOO	87.6 Δ +24.8	78.2 Δ +39.7	87.3 Δ +65.1	81.3 Δ +68.8	71.9 Δ +63.9	48.9 Δ +48.9	75.5 Δ +54.1	80.3 Δ +50.8	65.7 Δ +56.5
GRPO	90.8 Δ +28.0	66.1 Δ +27.6	89.3 Δ +67.1	74.7 Δ +62.2	72.5 Δ +64.5	64.7 Δ +64.7	77.6 Δ +56.2	79.3 Δ +49.8	66.1 Δ +56.9
Memory-Augmented RL-based Methods (Δ vs. MemRL)									
MemRL	62.8 Δ 0.0	38.5 Δ 0.0	22.2 Δ 0.0	12.5 Δ 0.0	8.00 Δ 0.0	0.00 Δ 0.0	21.4 Δ 0.0	29.5 Δ 0.0	9.2 Δ 0.0
EvolveR	64.9 Δ +2.1	33.3 Δ -5.2	46.4 Δ +24.2	13.3 Δ +0.8	33.3 Δ +25.3	33.3 Δ +33.3	43.8 Δ +22.4	42.5 Δ +13.0	17.6 Δ +8.4
Mem0+GRPO	78.1 Δ +15.3	54.8 Δ +16.3	56.1 Δ +33.9	31.0 Δ +18.5	65.0 Δ +57.0	26.9 Δ +26.9	54.7 Δ +33.3	58.1 Δ +28.6	37.5 Δ +28.3
SimpleMem+GRPO	89.5 Δ +26.7	63.6 Δ +25.1	60.0 Δ +37.8	50.0 Δ +37.5	64.9 Δ +56.9	26.3 Δ +26.3	62.5 Δ +41.1	67.8 Δ +38.3	46.9 Δ +37.7
SkillRL	97.9 Δ +35.1	71.4 Δ +32.9	90.0 Δ +67.8	**90.0** Δ +77.5	**95.5** Δ +87.5	87.5 Δ +87.5	89.9 Δ +68.5	85.2 Δ +55.7	72.7 Δ +63.5
AgentOCR	95.6 Δ +32.8	96.2 Δ +57.7	78.1 Δ +55.9	73.2 Δ +60.7	72.4 Δ +64.4	72.0 Δ +72.0	81.2 Δ +59.8	78.6 Δ +49.1	59.3 Δ +50.1
Skill0	**100.0** Δ +37.2	85.8 Δ +47.3	94.6 Δ +72.4	81.9 Δ +69.4	85.7 Δ +77.7	80.1 Δ +80.1	89.8 Δ +68.4	85.3 Δ +55.8	71.9 Δ +62.7
GENMEM (RL)	95.8 Δ +33.0	**100** Δ +61.5	**96.8** Δ +74.6	88.7 Δ +76.2	90.5 Δ +82.5	**88.2** Δ +88.2	**93.6** Δ +72.2	**90.8** Δ +61.3	**82.8** Δ +73.6

Table 2: **Search-based QA results (Qwen2.5-7B; F1, %).** 2Wiki and HotpotQA are in-domain; the remaining benchmarks are out-of-domain.

	In-Domain F1 (%)		Out-of-Domain F1 (%)				Overall
Method	**2Wiki**	**HotpotQA**	**Bamboogle**	**MuSiQue**	**NQ**	**TriviaQA**	**Avg.**
Prompt-based Agentic or Memory-based Methods (Δ vs. ReAct)							
Base	25.41 Δ -2.10	26.63 Δ -16.18	17.86 Δ -9.77	12.15 Δ -7.19	19.72 Δ -10.29	49.08 Δ -5.47	25.1 Δ -8.5
CoT	23.55 Δ -3.96	29.10 Δ -13.71	37.56 Δ +9.93	14.35 Δ -4.99	22.47 Δ -7.54	49.33 Δ -5.22	29.4 Δ -4.2
FS-RAG	17.71 Δ -9.80	29.21 Δ -13.60	16.86 Δ -10.77	10.74 Δ -8.60	16.82 Δ -13.19	35.02 Δ -19.53	21.1 Δ -12.5
FL-RAG	19.78 Δ -7.73	34.42 Δ -8.39	24.10 Δ -3.53	12.46 Δ -6.88	19.72 Δ -10.29	42.66 Δ -11.89	25.5 Δ -8.1
ReAct	27.51 Δ 0.00	42.81 Δ 0.00	27.63 Δ 0.00	19.34 Δ 0.00	30.01 Δ 0.00	54.55 Δ 0.00	33.6 Δ 0.0
IRCoT	36.45 Δ +8.94	26.29 Δ -16.52	21.90 Δ -5.73	8.39 Δ -10.95	19.63 Δ -10.38	49.43 Δ -5.12	27.0 Δ -6.6
TCRAG	29.70 Δ +2.19	40.83 Δ -1.98	25.13 Δ -2.50	17.56 Δ -1.78	29.01 Δ -1.00	54.78 Δ +0.23	32.8 Δ -0.8
GENMEM	42.91 Δ +15.40	35.83 Δ -6.98	22.21 Δ -5.42	19.85 Δ +0.51	33.88 Δ +3.87	47.62 Δ -6.93	33.7 Δ +0.1
RL-based Methods (Δ vs. ReSearch)							
ReSearch	30.03 Δ 0.00	30.39 Δ 0.00	30.42 Δ 0.00	12.58 Δ 0.00	23.69 Δ 0.00	48.25 Δ 0.00	29.2 Δ 0.0
Search-R1	35.03 Δ +5.00	38.89 Δ +8.50	42.04 Δ +11.62	19.08 Δ +6.50	29.59 Δ +5.90	55.91 Δ +7.66	36.8 Δ +7.6
AEPO	19.88 Δ -10.15	13.85 Δ -16.54	13.24 Δ -17.18	5.85 Δ -6.73	9.93 Δ -13.76	17.53 Δ -30.72	13.4 Δ -15.8
ARPO	30.71 Δ +0.68	25.20 Δ -5.19	32.94 Δ +2.52	12.71 Δ +0.13	17.80 Δ -5.89	40.16 Δ -8.09	26.6 Δ -2.6
Mem1	25.29 Δ -4.74	29.98 Δ -0.41	36.50 Δ +6.08	14.13 Δ +1.55	26.38 Δ +2.69	51.04 Δ +2.79	30.6 Δ +1.4
ZeroSearch	35.2 Δ +5.17	34.6 Δ +4.21	27.8 Δ -2.62	18.4 Δ +5.82	43.6 Δ +19.91	61.8 Δ +13.55	36.9 Δ +7.7
AgenticRAG-R1	38.34 Δ +8.31	45.15 Δ +14.76	49.21 Δ +18.79	22.01 Δ +9.43	23.60 Δ -0.09	58.45 Δ +10.20	39.5 Δ +10.3
Memory-Augmented RL-based Methods (Δ vs. EvolveR)							
EvolveR	42.0 Δ 0.0	38.2 Δ 0.0	54.4 Δ 0.0	15.6 Δ 0.0	43.5 Δ 0.0	63.4 Δ 0.0	42.9 Δ 0.0
SkillRL	40.3 Δ -1.7	43.2 Δ +5.0	**73.8** Δ +19.4	20.2 Δ +4.6	45.9 Δ +2.4	63.3 Δ -0.1	47.8 Δ +4.9
SkillGraph	43.4 Δ +1.4	44.7 Δ +6.5	72.6 Δ +18.2	19.5 Δ +3.9	48.0 Δ +4.5	63.8 Δ +0.4	48.7 Δ +5.8
AgentOCR	38.3 Δ -3.7	40.8 Δ +2.6	36.8 Δ -17.6	15.7 Δ +0.1	43.1 Δ -0.4	61.0 Δ -2.4	39.3 Δ -3.6
Skill0	38.3 Δ -3.7	40.0 Δ +1.8	66.9 Δ +12.5	16.4 Δ +0.8	42.7 Δ -0.8	61.1 Δ -2.3	44.2 Δ +1.3
Skill-SD	42.1 Δ +0.1	44.3 Δ +6.1	69.0 Δ +14.6	20.2 Δ +4.6	47.1 Δ +3.6	64.5 Δ +1.1	47.9 Δ +5.0
SAPO	45.2 Δ +3.2	45.0 Δ +6.8	46.4 Δ -8.0	18.3 Δ +2.7	48.4 Δ +4.9	68.9 Δ +5.5	45.4 Δ +2.5
SDAR	48.4 Δ +6.4	43.8 Δ +5.6	73.0 Δ +18.6	19.6 Δ +4.0	46.3 Δ +2.8	63.5 Δ +0.1	49.1 Δ +6.2
GENMEM (RL)	**54.8** Δ +12.8	**52.9** Δ +14.7	64.6 Δ +10.2	**23.5** Δ +7.9	**50.1** Δ +6.6	**70.7** Δ +7.3	**52.8** Δ +9.9

Table 3: **Skill transfer across held-out benchmarks and agent harnesses under DeepSeek-V4-Flash.** Scores and parenthesized changes over the skill-free settings are in %; **bold** marks the better result within each harness.

	StackPlanner				TCRAG				OpenHands			
Method	**Search**	**Research**	**SQL**	**Code**	**Search**	**Research**	**SQL**	**Code**	**Search**	**Research**	**SQL**	**Code**
Skill-free	**58.0**	49.96	78.0	74.0	**56.0**	48.88	80.0	62.0	57.0	51.48	79.0	72.0
+ GENMEM Skills	**58.0** Δ 0.0	**52.03** Δ +2.07	**83.0** Δ +5.0	**75.0** Δ +1.0	55.0 Δ -1.0	**50.92** Δ +2.04	**81.0** Δ +1.0	**63.0** Δ +1.0	**58.0** Δ +1.0	**53.60** Δ +2.12	**81.0** Δ +2.0	**74.0** Δ +2.0

capture. The gains over SkillRouter further support the effect of generative addressing, suggesting that SID alignment enables **GENMEM** to strategically relevant experience.

Table 4: Retrieval hit rates (%) and downstream task accuracy (%) across evolution checkpoints. Dense methods rebuild indices at each step; SID addresses are frozen by construction.

Method	Metric	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9
Qwen3-Emb	Cand. Hit	8.41	8.79	8.32	9.07	8.60	8.04	8.51	7.85	8.13	7.66
	Hit@1	0.19	0.28	0.19	0.37	0.28	0.19	0.28	0.19	0.28	0.19
	Hit@5	2.89	3.17	2.80	3.36	3.08	2.71	2.99	2.61	2.80	2.52
	Task ACC	28.53	28.91	28.62	29.10	28.74	28.31	28.65	28.12	28.46	27.98
TF-IDF	Cand. Hit	18.47	19.03	20.06	19.31	19.78	20.34	19.59	19.97	19.41	19.69
	Hit@1	2.36	2.64	3.01	2.73	2.92	3.20	2.83	3.01	2.73	2.92
	Hit@5	12.52	12.99	**13.74**	13.18	13.55	**14.02**	11.27	13.64	13.08	13.36
	Task ACC	30.02	30.58	**31.46**	30.91	31.18	**31.72**	29.30	31.26	30.73	31.08
SkillRouter	Cand. Hit	15.79	16.26	15.98	16.54	16.17	15.61	16.08	15.33	15.70	15.14
	Hit@1	6.62	6.81	6.53	7.00	6.72	6.34	6.62	6.16	6.44	6.06
	Hit@5	10.28	10.65	10.37	10.93	10.56	10.09	12.47	9.91	10.19	9.72
	Task ACC	28.74	29.12	28.93	29.48	29.21	28.86	30.84	28.79	29.07	28.95
GENMEM	Cand. Hit	**20.25**	**20.91**	**20.63**	**21.84**	**22.68**	**22.21**	**23.52**	**24.36**	**23.99**	**25.58**
	Hit@1	**7.00**	**7.37**	**7.19**	**7.93**	**8.40**	**8.12**	**9.05**	**9.52**	**9.24**	**10.08**
	Hit@5	**13.15**	**13.71**	13.43	**14.55**	**15.20**	13.92	**16.04**	**16.79**	**16.51**	**17.63**
	Task ACC	**30.56**	**31.10**	30.88	**32.14**	**32.83**	31.49	**33.68**	**34.76**	**34.39**	**36.20**

❷ **Downstream Task Performance.** Tables 1 and 2 show that **GENMEM** improves task performance on ALFWorld, WebShop, and search-augmented QA both before and after Harness-RL training of StackPlanner. Compared with the corresponding memory-free baselines, **GENMEM** yields average gains of 17.0 and 15.3 percentage points before and after training, respectively. It also outperforms memory-augmented baselines, exceeding SDAR, the second-best method, by 3.7 percentage points on average token F1. These improvements suggest that the retrieved experience provides useful support for downstream task execution and complements the gains from planning-policy optimization. **Generalization to unseen domains and harness:** Table 3 applies **GENMEM** to four benchmarks and three agent harnesses. **Cross-benchmark transfer.** On unseen datasets from related task families, Research yields consistent gains of 2%. Search ranges from a 1% decrease under TCRAG to a 1% gain under OpenHands Harness (Wang et al., 2025).

4.3 ONLINE SELF-EVOLUTION (RQ2)

We evaluate continual improvement on a held-out query stream disjoint from the initial memory bank, with model parameters, SID encoder, and codebooks frozen throughout. For each query, the system retrieves, executes, and scores the answer against the reference *before* passing the trajectory to the update policy; the resulting `insert` or `revise` is applied before the next query arrives. The reference is used for evaluation only and never enters the update prompt. The stream is partitioned into consecutive blocks; a fixed probe set is evaluated after each block without writing to the bank during probing. Table 4 reports retrieval hit rates and downstream task accuracy for four methods as the memory bank evolves over T steps.

❶ ***Addressing stability and retrieval quality across evolution.*** As evolution progresses, **GENMEM**'s retrieval hit rates and task accuracy *rise steadily* across checkpoints: Candidate Hit improves from 20.25% at t=0 to 25.58% at t=9, and Task ACC climbs from 30.56% to 36.20%, demonstrating that the refined payloads translate into better downstream support. In contrast, the three content-coupled baselines—whose dense-embedding or TF-IDF indices must be rebuilt from the latest payloads at each checkpoint—show *limited gains or outright degradation*: embedding drift redirects queries to incorrect entries, offsetting any benefit from improved content. This divergence empirically confirms the **addressing invariance** property (Challenge 2): SID addresses absorb content churn, allowing agent to harvest full benefit of evolution, while content-coupled addressing suffers catastrophic drift.

❷ ***Evolution dynamics.*** Figure 3 visualizes how the memory bank changes over the evolution stream: (a) SID lineage traces memory retrieval, revision, and reuse across successive queries, highlighting cross-case evolution: memories revised from one case are subsequently reused to help solve another; (b) update concentration, showing the cumulative share of events against the fraction of active SIDs. Curves closer to the upper-left corner indicate stronger concentration; (c) per-block breakdown of insert vs. content-changing revise vs. retention, showing that MemEvolver writes selectively rather than indiscriminately; and (d) average payload edit distance, quantifying how much existing entries are rewritten. Together with Table 4, these statistics show that substantial content churn occurs throughout evolution—yet only **GENMEM** translates this churn into consistent performance gains.

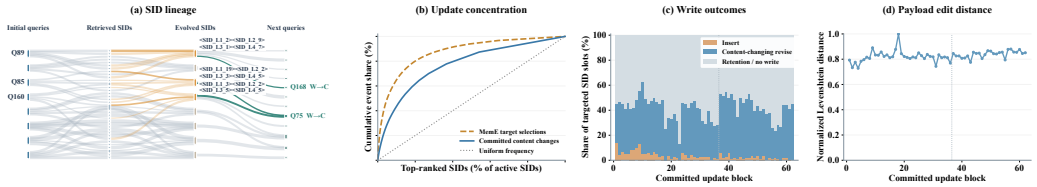

Figure 3: Memory bank evolution dynamics over the query stream: SID lineage, SID update concentration, operation-type distribution, and payload edit distance.

5 SIDs as Symbolic Language: Composition& Emergent Reasoning

A central claim of **GenMem** is that SIDs are not opaque lookup keys but *informative symbolic tokens* that carry task-relevant structure. We test this on GPQA (Rein et al., 2023) by varying the depth of SID prefixes supplied without experience text, then comparing complete SIDs with rewritten and directly retrieved experience. Appendix G.5 specifies the input conditions. Using MemRetriever, we generate k SIDs per query with beam size $k \in \{1, 5, 10\}$ and make two comparisons:

- **Symbolic information scaling**: the model receives the raw query alone, then progressively deeper SID prefixes—L_1, L_1–L_2, L_1–L_3, and the complete L_1–L_4 address, appended as token sequences. No experience text is provided; model must reason from symbolic addresses alone.
- **Experience representation**: the complete SIDs are used to retrieve stored experience, which is either rewritten into task-adaptive support or supplied directly. Both conditions access the memory bank, testing the contribution of rewriting rather than experience generation from SID tokens alone.

Observations. Figure 4 shows that accuracy increases with SID depth at each beam setting, from 48.32% for the raw query to 54.06–54.66% with complete addresses. This gain of 5.74–6.34 percentage points occurs without experience text, supporting the usefulness of symbolic input in this evaluation. Rewritten experience reaches 55.65–56.91%, while directly retrieved experience reaches 56.09–56.63%. Rewriting scores higher only at $k = 10$, so neither text condition consistently dominates. The additional gains suggest that experience text supplies useful detail beyond the addresses, without establishing generalization to unseen SID combinations.

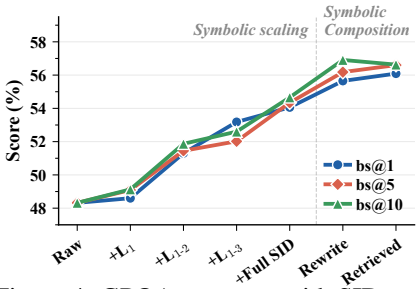

Figure 4: GPQA accuracy with SID prefixes and experience text.

6 Conclusion, Limitations and Future Work

GenMem demonstrates that reformulating memory as generative symbolic addressing rather than discriminative retrieval, can resolve the sparsity and invariance–evolution tension: SID generation over a hierarchical address space keeps addresses stable while content is refined, enabling long-horizon self-evolution without catastrophic addressing drift. Experiments across embodied planning, web navigation, and multi-hop QA evaluate this design against strong memory-augmented agents, and we further observe that SIDs are *compositional*: combining addresses composes their stored experiences to reach novel situations no single entry covers.

Limitations and Future Directions. Two practical limitations stand out: ❶ some leaf codes attract disproportionate traffic, creating uneven utilization across the codebook, and ❷ cold-start noise in early trajectories can propagate into the SID space. Both point to a natural extension—*adaptive codebook allocation* with a background revisit policy that proactively evolves under-exercised subtrees, equalizing the address tree without violating frozen-address invariance. A second direction concerns *SID–text-token interleaving*: because SIDs share the decoder substrate with natural-language tokens, the compositional behaviors observed in Section 5 suggest that SIDs could serve as first-class orchestration tokens rather than the lookup indexs in long, multi-tool workflows, turning the memory bank into a programmable symbolic layer for complex task planning.

AI USE STATEMENT

Large language models were used for language polishing, literature discovery, synthetic trajectory construction, figure-design assistance, and AI-assisted programming. LLMs helped generate candidate agent trajectories from public benchmark tasks, compare successful and failed executions, and distill reusable strategies and failure-recovery lessons into the experience memory; they also provided visual drafts, code suggestions, and debugging support. All AI-assisted trajectories, distilled experiences, figures, citations, and code were reviewed and verified by the authors, with benchmark verifiers, environment feedback, automated tests, and human inspection used as appropriate to validate generated artifacts and the qualitative cases reported in the paper. The research questions, methodology, experimental design, analyses, and conclusions were determined by the authors.

ETHICS STATEMENT

All experiments use publicly available benchmarks, such as ALFWorld, WebShop, GPQA, BIRD, SWE-bench Verified, and DeepResearch Bench, under their respective licenses and terms. The study involves no human or animal subjects and uses no private user conversations. Agent trajectories and reusable experiences are generated from benchmark tasks using author-designed protocols and LLM assistance rather than collected from real users. Interactive and software-engineering tasks are executed in benchmark environments or isolated workspaces to limit unintended effects on external systems.

REPRODUCIBILITY STATEMENT

The appendix reports dataset versions and splits, memory-construction statistics, prompts, model and decoding configurations, tool and token budgets, training settings, baseline implementations, and evaluation procedures. The method and objectives are described in Sections 2–3; the general experimental protocol is documented in Appendix E; the cross-domain memory split and overlap checks are reported in Appendix; and the complete prompts are provided in Appendix I.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Ustun, and Sara Hooker. Back to basics: Revisiting REINFORCE style optimization for learning from human feedback in LLMs. *arXiv preprint arXiv:2402.14740*, 2024.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023. URL https://arxiv.org/abs/2309.16609.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. Research: Learning to reason with search for LLMs via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025a.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025b.

Xin Cheng, Rui Tian, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Chengqi Deng, Shangyan Zhou, Chenggang Zhao, Zhewen Hao, Yukun Li, Han Zhang, Zhengyan Zhang, Yixu Wei, M.Y. Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026. URL https://arxiv.org/abs/2601.07372.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Nicola de Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. OneRec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965*, 2025.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*, 2025a.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhou, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025b.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Licheng Zhang, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. In *International Conference on Learning Representations*, volume 2026, pp. 42414–42448, 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. *arXiv preprint arXiv:2508.06433*, 2025. ACL 2026 Findings.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Lang Feng, Fuchao Yang, Feng Chen, Xin Cheng, Haiyang Xu, Zhenglin Wan, Ming Yan, and Bo An. Agentocr: Reimagining agent history via optical self-compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5067–5086, 2026.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

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. *arXiv preprint arXiv:2106.09685*, 2022.

Xinke Jiang, Yue Fang, Rihong Qiu, Haoyu Zhang, Yongxin Xu, Hao Chen, Wentao Zhang, Ruizhe Zhang, Yuchen Fang, Xu Chu, et al. Tc-rag: Turing-complete rag's case study on medical llm systems. *ACL oral 2025*, 2025.

Xinke Jiang, Yue Fang, Zhibang Yang, Jiaran Gao, Zhixin Zhang, Tao Feng, Rihong Qiu, Wentao Zhang, Hongxin Ding, Ruizhe Zhang, Yongxin Xu, Yuheng Huang, Xu Chu, Junfeng Zhao, and Yasha Wang. Agenticrag-r1: Agentic reinforcement learning with stack memory for multi-step reasoning, retrieval and memorizing. *arXiv preprint arXiv:2608.29622*, 2026a.

Xinke Jiang, Zhixin Zhang, Zhibang Yang, Jiaran Gao, Rihong Qiu, Shijin Chen, Xu Chu, Junfeng Zhao, and Yasha Wang. Harness-rl: Black-box reinforcement learning with action-args decoupling for central-agent multi-agent harnesses, 2026b. URL https://arxiv.org/abs/2608.29641.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

Xiaoyuan Li, Moxin Li, Keqin Bao, Yubo Ma, Wenjie Wang, Dayiheng Liu, and Fuli Feng. SkillGraph: Skill-augmented reinforcement learning for agents via evolving skill graphs. *arXiv preprint arXiv:2605.12039*, 2026.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. SimpleMem: Efficient lifelong memory for LLM agents. *arXiv preprint arXiv:2601.02553*, 2026.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, et al. Self-distilled agentic reinforcement learning. *arXiv preprint arXiv:2605.15155*, 2026a.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026b.

Weitao Ma, Xiaocheng Feng, Lei Huang, Xiachong Feng, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Bing Qin. Fine-mem: Fine-grained feedback alignment for long-horizon memory management. *arXiv preprint arXiv:2601.08435*, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. DeepSeek-Math: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Fei Huang, and Yan Zhang. Zerosearch: Incentivize the search capability of LLMs without searching. *arXiv preprint arXiv:2505.04588*, 2025.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Donald Metzler, et al. Transformer memory as a differentiable search index. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions, 2023.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026a.

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

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026b.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025. URL https://arxiv.org/abs/2407.16741.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. EvolveR: Self-evolving LLM agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025. Accepted by ICML 2026.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025.

Ruizhe Zhang, Xinke Jiang, Zhibang Yang, Zhixin Zhang, Jiaran Gao, Yuzhen Xiao, Tao Feng, Yue Fang, Yuxuan Liu, Ruiqing Li, Hongbin Lai, Huheng Huang, Xu Chu, Junfeng Zhao, and Yasha Wang. Stackplanner: A centralized hierarchical multi-agent system with task-experience memory management, 2026a. URL https://arxiv.org/abs/2601.05890.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026b.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks, 2026c. URL https://arxiv.org/abs/2510.12635.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

Zhiwei Zhang, Yudi Lin, Nikki Lijing Kuang, Linlin Wu, Xiaomin Li, Songtao Liu, and Fenglong Ma. Co-evolving skill generation and policy optimization. *arXiv preprint arXiv:2606.08755*, 2026d.

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 (AAAI)*, 2024.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. SkillRouter: Skill routing for LLM agents at scale. *arXiv preprint arXiv:2603.22455*, 2026.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025.

A Related Work

Memory-Augmented LLM Agents. Memory-augmented LLM agents accumulate and reuse experience across tasks to support long-horizon reasoning and continual adaptation (Zhang et al., 2024; Sumers et al., 2024). Existing work explores both how experience is organized and how memory is managed. Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024) distill past interactions into verbal reflections and reusable insights without gradient updates, while SimpleMem (Liu et al., 2026) and MemP (Fang et al., 2025) curate compact memory representations. Mem0 (Chhikara et al., 2025) and A-MEM (Xu et al., 2025) further investigate structured memory management through graph-based organization or associative linking. At the skill level, SkillRL (Xia et al., 2026) and SkillGraph (Li et al., 2026) organize reusable knowledge into hierarchical or graph-structured banks. Trace2Skill (Ni et al., 2026) consolidates execution trajectories into portable skill directories, while SkillX (Wang et al., 2026a) constructs three-tiered hierarchies of strategic plans, functional skills, and atomic operations. Beyond memory organization, MemRL (Zhang et al., 2026b), Memory-as-Action (Zhang et al., 2026c), and MemGen (Zhang et al., 2025) introduce learned mechanisms for memory utilization, control, or generation. SKILL0 (Lu et al., 2026b) takes a complementary approach by internalizing skills into model parameters. Despite these advances, **richer memory organization does not by itself provide a model with a structured address space that it can learn to navigate: similarity-based access remains sensitive to how experiences are represented, and semantic resemblance need not reflect their strategic utility in the current context.** SkillRouter (Zheng et al., 2026) further highlights the sensitivity of skill selection to the information exposed during retrieval. **Learning which memories to access and refine also remains challenging when supervision is derived primarily from delayed task outcomes, which provide limited credit assignment to individual memory operations.**

Generative Retrieval and Discrete Representations. Generative retrieval formulates information access as autoregressive identifier generation, allowing a model to directly predict the target of retrieval. DSI (Tay et al., 2022) generates document identifiers from queries, while GENRE (de Cao et al., 2021) retrieves entities by generating their names. TIGER (Rajput et al., 2023) extends identifier generation to recommendation using semantic IDs constructed through residual quantization. These methods build on discrete representation learning: VQ-VAE (van den Oord et al., 2017) maps continuous representations to learned codebook entries, and RQ-VAE (Lee et al., 2022) progressively quantizes residual information into sequences of discrete codes. Such representations provide compact identifiers with coarse-to-fine semantic structure, enabling autoregressive access through shared code prefixes. **However, generating identifiers for indexed content does not by itself resolve the requirements of an evolving experience bank, where entries must be selectively inserted, revised, consolidated, or removed. When semantic identifiers depend on content representations, revisions can create tension between preserving learned addresses and maintaining their semantic coherence. Existing retrieval formulations therefore leave open how to jointly support stable context-to-address mappings and targeted content updates through a shared generative interface.**

B Formal Foundations and SID Representation

This appendix collects formal definitions and analysis underlying the design choices in the main text.

B.1 The Sparsity Structure of Experience Space

A central insight of this work is that experience space is *inherently sparse*—a property that existing dense memory systems fail to exploit. We identify four distinct sparsity dimensions that collectively motivate generative symbolic addressing.

Value Sparsity. Let $\mathcal{T}_q = \{\tau^{(1)}, \ldots, \tau^{(K)}\}$ denote a set of K trajectories generated for query q. Define the *experience value* of a trajectory as its contribution to the generalizable experience base:

$$v(\tau^{(k)}) = \mathbb{E}_{q' \sim \mathcal{D}}\big[\Delta r(q', \mathcal{M} \cup \mathcal{E}(\tau^{(k)})) - \Delta r(q', \mathcal{M})\big],$$

where $\mathcal{E}(\tau)$ denotes experience distilled from τ and Δr denotes task performance improvement. In practice, $v(\tau^{(k)})$ is high only for a small fraction of trajectories—those that exhibit novel, generalizable strategies or capture failure patterns not previously recorded.

Structural Sparsity. Even among high-value experiences, the experience space exhibits strong structural regularity. Formally, let $\{e_i\}_{i=1}^{N}$ be the semantic embeddings of all experience entries. These embeddings lie approximately on a low-dimensional manifold:

$$\dim\big(\mathrm{span}\{e_i\}_{i=1}^{N}\big) \ll N,$$

because many experiences share latent structural components—task categories, sub-task types, tool-use patterns—even when their surface-level descriptions differ. This *structural sparsity* means that the effective address space of experience is far smaller than its raw cardinality. This is precisely the principle behind RQ-KMeans-based SID construction: the multi-level codebook exploits structural sparsity to represent $N_1 \times N_2 \times \cdots \times N_L$ distinct experience regions using only $N_1 + N_2 + \cdots + N_L$ codebook entries.

Access Sparsity. Semantic similarity does not by itself determine whether a memory will help an agent solve a task. Let $c = (q, h_t, \text{env})$ denote the execution context, comprising the query, the agent's working state, and the environmental state. We define the *execution utility* of a memory m_i as $u(c, m_i) = \mathbb{E}_{\tau\sim\pi(\cdot|c,m_i)}[r(\tau, q)]$, the expected task reward when the execution policy π uses that memory. Access sparsity is the assumption that only a small subset of the memory bank exceeds a given utility threshold for any particular context. Formally, $|\mathcal{R}^*(c)| \ll |\mathcal{M}|$, where $\mathcal{R}^*(c) = \{m_i \in \mathcal{M} : u(c, m_i) > \epsilon\}$ and $\epsilon > 0$.

Evolution sparsity (update). Only a small subset of experiences—those repeatedly exercised by frequent queries—warrant refinement; blanket updates waste compute and blur useful entries. Effective memory systems must therefore learn selective evolution, deciding not only *how* to update an entry but also *whether* to leave it untouched through a content-preserving `revise` operation (§2).

These four sparsity dimensions together motivate designing experience memory around *sparse, structured, generative addressing* rather than dense global retrieval—realized concretely through the SID framework (§2).

B.2 Retrieval Utility and Address Stability

Retrieval utility. A similarity score measures agreement between representations; it does not by itself specify the value of a memory for task execution. For example, two experiences may describe similar tasks but prescribe different actions. Ranking memories by $\mathrm{sim}(f_q(q), f_m(x_i))$ therefore need not produce the same ordering as ranking them by the execution utility $u(c, m_i)$ defined in Appendix B.1. This is a distinction between objectives, not an inherent limitation of discriminative retrieval. A scoring model can also condition on the execution context and be trained with task feedback. In **GenMem**, process and outcome rewards train the memory policies to select and rewrite experience for downstream use (Section 3); symbolic addressing alone does not ensure useful retrieval.

Address stability. If a retrieval score depends on an embedding of the current payload, revising that payload and recomputing its embedding can change its rank for the same query, even when its role is unchanged. **GenMem** separates the address from the payload: a revision changes (x_i, z_i) to (x_i', z_i), preserving the SID used for lookup and update targeting (Section 2). With the addressing policy and codebooks fixed, a payload revision does not by itself change the SID generated for an unchanged input context. It can still change how useful the retrieved content is. This design does not imply that vector-based stores cannot support stable identifiers or targeted updates; GenMem uses a shared, generatable SID interface for both reading and writing memory.

B.3 SID as a Generative Token Sequence

An SID $\mathbf{z} = (s^1, \ldots, s^L)$ consists of L level-specific tokens, each identifying an entry in the corresponding codebook. Token identities are distinct across levels, even when their numerical

indices coincide. The four-level configuration in Section 2 therefore represents each address with four tokens, from <SID_L1_X> to <SID_L4_W>.

Conditioned on context $\mathbf{c}$, the addressing policy generates the sequence autoregressively,

$$p_\theta(\mathbf{z} \mid \mathbf{c}) = \prod_{\ell=1}^{L} p_\theta(s^\ell \mid \mathbf{c}, s^1, \ldots, s^{\ell-1}). \tag{10}$$

MemRetriever conditions address generation on the query and available reasoning context, whereas MemEvolver uses the execution trajectory to select an update address. In both cases, the generated SID identifies a memory slot whose payload is accessed by lookup. Offline quantization constructs the initial addresses; the learned policy subsequently generates addresses for retrieval or revision. Summary generation and payload revision are separate operations performed after lookup.

Figure 5 illustrates discrete sequence representations at a conceptual level. **GENMEM** applies this principle to textual experiences, using the sequence as an address rather than a lossless encoding of the associated text.

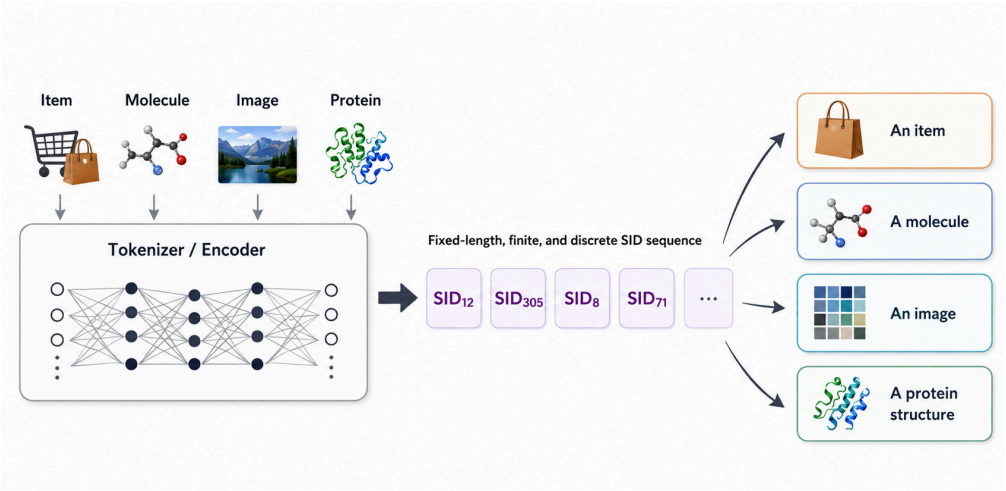

Figure 5: Discrete token sequences as object representations. In **GENMEM**, a text encoder and RQ-KMeans map seed experiences to SID addresses (Section 2). Other object types illustrate the general concept and are not evaluated in this work.

B.4 EMBEDDING RECONSTRUCTION FROM SIDS

The SID $\mathbf{z}_i = (s_i^1, \ldots, s_i^L)$ constructed in Section 2 selects one centroid from each codebook. Their sum approximates the original experience embedding $\mathbf{e}_i$; it does not reconstruct the experience text. Using the residual recurrence in Eq. 14, the reconstruction after ℓ levels is

$$\hat{\mathbf{e}}_i^{(\ell)} = \sum_{h=1}^{\ell} \mathbf{c}_{s_i^h}^{(h)}, \qquad \mathbf{r}_i^{(\ell+1)} = \mathbf{e}_i - \hat{\mathbf{e}}_i^{(\ell)}, \quad \ell = 1, \ldots, L. \tag{11}$$

Writing $\hat{\mathbf{e}}_i = \hat{\mathbf{e}}_i^{(L)}$, the per-entry Euclidean reconstruction error is

$$\varepsilon_i = \left\| \mathbf{e}_i - \hat{\mathbf{e}}_i \right\|_2 = \left\| \mathbf{r}_i^{(L+1)} \right\|_2. \tag{12}$$

Each successive codebook is fitted to the residuals left by the preceding levels. The Euclidean fitting objective minimizes mean squared residual error over the fitting set, but nearest-centroid assignment alone does not guarantee a decrease for every entry at every level. In particular, the selected centroid need not be closer to an incoming residual than the zero vector is. We therefore do not assume per-entry monotonic error reduction. Equation 12 defines an unnormalized Euclidean norm; its relationship to the reported reconstruction loss depends on the aggregation and normalization settings noted in Appendix C.1.

Reconstruction characterizes the offline quantization of seed embeddings, not online memory evolution. MemEvolver generates the target SID for an update, and the experience text at that address can subsequently change without re-encoding it or changing the SID (Appendix C.2).

B.5 SID Alignment Mappings

SID alignment pretraining (Section 3) connects queries q, execution trajectories τ, and experience texts $\mathbf{x}$ to the constructed address space. Each target SID is a complete sequence $\mathbf{z} = (s^1, \ldots, s^L)$. The five mappings comprise three address-generation tasks and two text-generation tasks:

1. **Query → SID** ($q \mapsto \mathbf{z}$). Predict the SID of the experience associated with the query in the training pair.
2. **Trajectory → SID** ($\tau \mapsto \mathbf{z}$). Predict the SID associated with the experience distilled from the trajectory. This task learns an address association, not an `insert` or `revise` decision.
3. **Experience → SID** ($\mathbf{x} \mapsto \mathbf{z}$). Predict the SID assigned to an experience by offline construction.
4. **SID → Experience** ($\mathbf{z} \mapsto \mathbf{x}$). Generate the experience text associated with the SID in the training pair.
5. **SID → Description** ($\mathbf{z} \mapsto \mathbf{d}$). Generate a natural-language semantic description $\mathbf{d}$ associated with the SID.

The reverse mappings train text generation conditioned on an address. They do not invoke `memory_lookup`; tool-mediated retrieval and memory evolution are taught during memory-operation mid-training (Section 3).

SID compositionality. Shared SID prefixes identify common sequences of selected centroids, while later symbols encode the remaining residuals (Appendix C.1). The alignment tasks associate these structured addresses with task contexts and experience content. Neither the codebook construction nor the alignment objective assigns a fixed semantic role, such as task domain or tool type, to an individual level. Whether the learned SID structure supports reasoning is examined empirically in Section 5, rather than assumed from the mapping tasks alone.

C Additional Method Details

C.1 RQ-KMeans Details

This section specifies the notation and Euclidean formulation of SID construction in Section 2, then reports the prefix and codebook comparisons used to examine the design.

Prefix-guided encoding. Let $N_{\text{seed}} = |\mathcal{X}_{\text{seed}}|$ denote the number of experience summaries before SID-based consolidation. We encode each summary as $\mathbf{e}_i = f_{\text{enc}}(p_{\text{enc}} \oplus \mathbf{x}_i) \in \mathbb{R}^d$, where f_{enc} is frozen, d is the embedding dimension, and $\oplus$ denotes text concatenation. The instruction prefix p_{enc} is reproduced in Appendix I.4; its `Query` field is filled with the extracted `rich_summary`. It emphasizes reusable procedures, applicability conditions, verification, and failure recovery. Thus, context enters through the experience summary, rather than through a separate concatenation of raw queries and trajectories.

Codebook fitting and assignment. At level $\ell \in \{1, \ldots, L\}$, the codebook $\mathcal{C}^{(\ell)} = \{\mathbf{c}_j^{(\ell)}\}_{j=1}^{N_\ell}$ contains N_ℓ centroids in $\mathbb{R}^d$. Starting with $\mathbf{r}_i^{(1)} = \mathbf{e}_i$, the Euclidean formulation fits each codebook to the incoming residuals using the K-Means objective

$$\mathcal{L}_\ell(\mathcal{C}^{(\ell)}) = \frac{1}{N_{\text{seed}}} \sum_{i=1}^{N_{\text{seed}}} \min_{j \in [N_\ell]} \left\| \mathbf{r}_i^{(\ell)} - \mathbf{c}_j^{(\ell)} \right\|_2^2, \tag{13}$$

where $[N_\ell] = \{1, \ldots, N_\ell\}$. The fitted centroids determine a code and the residual passed to the next level,

$$\begin{aligned} s_i^\ell &= \arg\min_{j \in [N_\ell]} \left\| \mathbf{r}_i^{(\ell)} - \mathbf{c}_j^{(\ell)} \right\|_2^2, \\ \mathbf{r}_i^{(\ell+1)} &= \mathbf{r}_i^{(\ell)} - \mathbf{c}_{s_i^\ell}^{(\ell)}. \end{aligned} \tag{14}$$

The resulting SID is $\mathbf{z}_i = (s_i^1, \ldots, s_i^L)$. Codebooks are fitted sequentially and then held fixed during online operation. Equations 13–14 specify the Euclidean form, not the Spherical or Weighted variants listed below.

Capacity and utilization. An L-level configuration has address capacity K_{addr} and level-specific SID vocabulary size V_{SID} given by

$$K_{\text{addr}} = \prod_{\ell=1}^{L} N_\ell, \qquad V_{\text{SID}} = \sum_{\ell=1}^{L} N_\ell. \tag{15}$$

Capacity counts possible addresses, not populated memory entries. For N_{eval} encoded summaries, let $\mathcal{P}_\ell = \{(s_i^1, \ldots, s_i^\ell) : 1 \le i \le N_{\text{eval}}\}$ be the set of observed prefixes. Single-code utilization and cumulative prefix utilization are

$$U_\ell = \frac{|\{s_i^\ell : 1 \le i \le N_{\text{eval}}\}|}{N_\ell}, \qquad U_{1:\ell} = \frac{|\mathcal{P}_\ell|}{\prod_{h=1}^{\ell} N_h}. \tag{16}$$

The used-leaf count is $|\mathcal{P}_L|$, and leaf utilization is $U_{1:L}$; utilization values are reported as percentages. The evaluation set need not be the full fitting set, so its size must be specified separately from N_{seed}. Per-entry Euclidean reconstruction error is defined in Appendix B.4.

Unique-code ratio. For a non-empty evaluation set, let $n_{\mathbf{z}} = \sum_{i=1}^{N_{\text{eval}}} \mathbb{I}[\mathbf{z}_i = \mathbf{z}]$ count the summaries assigned to a complete SID $\mathbf{z}$. We use *unique-code ratio* (UCR) to denote the number of distinct complete SIDs per encoded summary,

$$\text{UCR} = \frac{|\mathcal{P}_L|}{N_{\text{eval}}} = \frac{\sum_{\mathbf{z} \in \mathcal{Z}} \mathbb{I}[n_{\mathbf{z}} > 0]}{N_{\text{eval}}}, \qquad \mathcal{Z} = \prod_{\ell=1}^{L} [N_\ell]. \tag{17}$$

UCR characterizes address sharing across experiences. Unlike leaf utilization, its denominator is the sample count rather than address capacity: $\text{UCR} = (K_{\text{addr}}/N_{\text{eval}})U_{1:L}$. It is not the fraction of summaries assigned to singleton leaves, which is $N_{\text{eval}}^{-1} \sum_{\mathbf{z}} \mathbb{I}[n_{\mathbf{z}} = 1]$.

Assignment balance. Utilization measures whether codes are used, not how evenly assignments are distributed. For level ℓ, define the empirical code probabilities and entropy by

$$p_{\ell j} = \frac{1}{N_{\text{eval}}} \sum_{i=1}^{N_{\text{eval}}} \mathbb{I}[s_i^\ell = j], \qquad H_\ell = -\sum_{j=1}^{N_\ell} p_{\ell j} \log p_{\ell j}, \tag{18}$$

with $0 \log 0 = 0$ and natural logarithms. The normalized entropy and effective code count reported in Figure 8 are

$$H_\ell^{\text{norm}} = \frac{H_\ell}{\log N_\ell}, \qquad N_\ell^{\text{eff}} = \exp(H_\ell). \tag{19}$$

The normalization assumes $N_\ell > 1$ and accounts for differences in codebook size. The effective code count expresses assignment entropy as an equivalent number of equally frequent codes.

Distribution uniformity index (DUI). An entropy-based distribution uniformity index measures assignment balance by averaging normalized entropy across quantization levels,

$$\text{DUI}_{\text{entropy}} = \frac{1}{L} \sum_{\ell=1}^{L} H_\ell^{\text{norm}} = \frac{1}{L} \sum_{\ell=1}^{L} \frac{-\sum_{j=1}^{N_\ell} p_{\ell j} \log p_{\ell j}}{\log N_\ell}. \tag{20}$$

Each level contributes equally, irrespective of its codebook size. This definition measures marginal assignment balance, not statistical independence or semantic orthogonality between SID levels.

Inter-level code independence (ICR). To quantify statistical independence between SID levels, let $p(\mathbf{z}) = n_{\mathbf{z}}/N_{\text{eval}}$ be the empirical joint distribution of complete SIDs. Its joint entropy and total correlation are

$$H_{\text{joint}} = -\sum_{\mathbf{z}\in\mathcal{Z}} p(\mathbf{z}) \log p(\mathbf{z}), \qquad T = \sum_{\ell=1}^{L} H_\ell - H_{\text{joint}}. \tag{21}$$

Total correlation measures the discrepancy between the joint distribution and the product of its marginals. We propose an entropy-based independence ratio,

$$\text{ICR}_{\text{entropy}} = \frac{H_{\text{joint}}}{\sum_{\ell=1}^{L} H_\ell} = 1 - \frac{T}{\sum_{\ell=1}^{L} H_\ell}, \qquad \sum_{\ell=1}^{L} H_\ell > 0. \tag{22}$$

Higher values indicate less cross-level dependence relative to the total marginal entropy. If all marginal entropies vanish, we leave the ratio undefined and report the collapsed assignment separately. This statistic concerns dependence among code assignments, not semantic orthogonality. It is computed from the empirical joint distribution and can be sensitive to sparse sampling of the address space. The entropy-based ratio is a proposed diagnostic, distinct from the original ICR scores reported in the configuration comparison.

Offline quantization cost. For the Euclidean formulation in Appendix C.1, one assignment pass at level ℓ compares each of the N_{seed} residuals with N_ℓ centroids in $\mathbb{R}^d$. With T_{KM} K-Means iterations per level, the total assignment cost during fitting is

$$C_{\text{fit,assign}} = O\left(N_{\text{seed}} d T_{\text{KM}} \sum_{\ell=1}^{L} N_\ell \right). \tag{23}$$

Once the codebooks are fitted, assigning a SID to one already computed embedding costs

$$C_{\text{quantize}} = O\left(d \sum_{\ell=1}^{L} N_\ell \right). \tag{24}$$

The codebooks store $d\sum_{\ell=1}^{L} N_\ell$ scalar coordinates. These counts exclude text encoding, experience extraction, and LLM decoding; they are not end-to-end latency estimates. In particular, Eq. 24 does not describe online SID generation, which is performed by the learned memory policy.

C.2 Online Memory Self-Evolution

RQ-KMeans assigns addresses to the seed experiences during offline construction. During online memory self-evolution, MemEvolver generates the target SID for both `insert` and `revise`. The updated payload is not re-encoded and quantized to determine its update address.

Memory updates. Let $\mathcal{M}_t$ be the memory bank before online update t, and τ_t the newly completed execution trajectory. With model parameters and codebooks fixed, MemEvolver generates an update address, consults the associated entry, and returns an update tuple. Using the interface in Section 2, we write

$$(o_t, \mathbf{z}_{e,t}, \tilde{\mathbf{x}}_{e,t}) = \textbf{MemEvolver}(\tau_t, \mathcal{M}_t), \qquad o_t \in \{\texttt{insert}, \texttt{revise}\}. \tag{25}$$

Here o_t is the operation, $\mathbf{z}_{e,t} \in \mathcal{Z}$ is the target SID, and $\tilde{\mathbf{x}}_{e,t}$ is the updated payload. These are the quantities $(o, \mathbf{z}_e, \tilde{\mathbf{x}}_e)$ defined in the main text, indexed by update step t. An `insert` populates an unoccupied SID slot; a `revise` updates an existing entry. Neither operation reassigns the SID of an existing entry.

Addressing invariance. Write $\mathcal{M}_t[\mathbf{z}]$ for the payload at address $\mathbf{z}$, using $\varnothing$ when no experience is stored. Applying the update changes only the addressed payload,

$$\mathcal{M}_{t+1}[\mathbf{z}] = \begin{cases} \tilde{\mathbf{x}}_{e,t}, & \mathbf{z} = \mathbf{z}_{e,t}, \\ \mathcal{M}_t[\mathbf{z}], & \mathbf{z} \neq \mathbf{z}_{e,t}. \end{cases} \tag{26}$$

Revision supports retention, consolidation, and deletion through unchanged, merged, or empty payloads, respectively. These are outcomes of `revise`, not additional operation labels. Offline consolidation of seed experiences sharing an SID uses the SID-leaf synthesis prompt (Appendix I.3); online consolidation is handled by MemEvolver's revision at the target SID. Addressing invariance means that content changes do not reassign existing SIDs. It does not guarantee that the revised content remains useful for every query that previously retrieved it.

C.3 Reward Design Details

The reward decomposition follows Section 3. Process feedback evaluates the generated SID and the experience supplied by the memory policy. Outcome feedback measures the change in StackPlanner performance relative to execution without memory support. For either MemRetriever or MemEvolver, the scalar reward used in the main-text formulation is

$$R = \underbrace{R_{\text{format}} + R_{\text{strategy}} + R_{\text{SID}}}_{R_{\text{process}}} + R_{\text{out}}. \tag{27}$$

SID validity, SID agreement, and experience quality are distinct criteria. A valid address can be irrelevant to the task, and a correct address does not ensure that the generated experience is useful.

C.3.1 Process Reward

The process reward $R_{\text{proc}} = R_{\text{format}} + R_{\text{strategy}}$ decomposes as follows:

R_{format} **(format validity).** A binary reward checking structural correctness of the model's output:

- The generated SID must be a valid token sequence within the codebook vocabulary (L tokens, each in the correct range $[1, N_\ell]$).
- The operation field must be one of two permitted values, `insert` or `revise`.
- The output must conform to the expected structured format (e.g., JSON wrapper with designated fields).

$R_{\text{format}} = 1$ if all checks pass, 0 otherwise.

R_{strategy} **(strategic quality).** A continuous reward assessing strategic soundness of the model's retrieval or evolution decision, instantiated per module:

- **MemRetriever:** evaluates SID targeting accuracy (whether the addressed region contains relevant experiences) and rewrite helpfulness (whether the support signal aids downstream planning), scored by an LLM auditor on $[0, 1]$.
- **MemEvolver:** instantiated as $R_{\text{evo}} = r_{\text{faithful}} + r_{\text{loc}} + r_{\text{op}}$ (see §C.3.2 for full definitions).

C.3.2 Evolution Reward

The evolution reward R_{evo} evaluates memory update quality along three axes: faithfulness of the distilled experience $\tilde{x}_e$ to the trajectory, correctness of the update SID z_e, and appropriateness of the operation o. It decomposes as:

$$R_{\text{evo}} = r_{\text{faithful}}\big(\tau,\, \tilde{x}_e\big) + r_{\text{loc}}\big(\tilde{x}_e,\, z_e,\, \mathcal{M}\big) + r_{\text{op}}\big(o,\, z_e,\, \mathcal{M}\big). \tag{28}$$

- $r_{\text{faithful}}(\tau, \tilde{x}_e)$: **Faithfulness reward.** Measures whether $\tilde{x}_e$ accurately reflects the strategies in the source trajectory τ without hallucination or critical omissions.
- $r_{\text{loc}}(\tilde{x}_e, z_e, \mathcal{M})$: **Localization reward.** Evaluates whether z_e correctly identifies the semantic region where $\tilde{x}_e$ belongs, i.e., whether the addressed neighborhood contains thematically related experiences.
- $r_{\text{op}}(o, z_e, \mathcal{M})$: **Operation reward.** Assesses whether $o \in \{\texttt{insert}, \texttt{revise}\}$ is appropriate for the current memory state at z_e—e.g., `revise` for an existing entry and `insert` for a new entry. Retaining, merging, and clearing content are outcomes of revision.

Relationship to main-text reward hierarchy. In the main-text post-training formulation (§3), rewards are organized as:

$$\begin{aligned} R &= \alpha_{\text{proc}}\, R_{\text{proc}} + \alpha_{\text{out}}\, R_{\text{out}} \\ R_{\text{proc}} &= R_{\text{format}} + R_{\text{strategy}} \end{aligned}$$

R_{evo} is the **instantiation of R_{strategy} for MemEvolver**, decomposing strategic quality into faithfulness, localization, and operation-appropriateness sub-signals. For MemRetriever, R_{strategy} evaluates SID accuracy and rewrite relevance (§C.3.1). R_{format} is shared across both modules, checking structural validity (valid SID tokens, well-formed JSON output).

C.3.3 Outcome Reward

For the same query, let $\hat{y}_{\text{mem}}$ and $\hat{y}_{\text{no-mem}}$ denote StackPlanner's outputs with the generated experience and without memory support. The benchmark task scorer r_{task} evaluates both outputs against the reference answer or environment success criterion. Following Eq. 8,

$$\begin{aligned} R_{\text{out}} = r_{\text{task}}(\hat{y}_{\text{mem}}, y^\star) - r_{\text{task}}(\hat{y}_{\text{no-mem}}, y^\star) \\ + \alpha_{\text{partial}}\, \mathbb{I}[\text{both executions succeed}]. \end{aligned} \tag{29}$$

Here $\alpha_{\text{partial}} > 0$ is the bonus for preserving a successful outcome when the baseline already succeeds. It is not the task scorer's partial-credit rule. For binary task scores, this expression gives 1 when only the memory-supported execution succeeds, -1 when only the baseline succeeds, α_{partial} when both succeed, and 0 when both fail. Thus, R_{out} measures comparative utility rather than absolute task accuracy. The with-memory and no-memory rollout prompts are reproduced in Appendix I.5.

Implementation details to verify. The equations above align the appendix with the main-text reward specification. Exact SID scoring and eligibility rules, the strategic-quality judge inputs and score scaling, benchmark scorers, α_{partial}, and handling of unavailable outcome feedback require confirmation from the training implementation. The supplied plotting scripts construct total reward as outcome reward plus ten times the sum of SID and semantic rewards. This plotted aggregation is not sufficient evidence for the reward passed to GRPO, and its correspondence with Eq. 27 remains to be checked.

C.4 Design Choices and Rationale

Fixed SID addressing and editable memory content. RQ-KMeans constructs the initial SID space from seed embeddings, after which the codebooks remain fixed (Section 2). Online adaptation changes the memory bank rather than the codebooks or model parameters. Addressing invariance additionally requires that existing SIDs are retained when their payloads change, as specified by Eq. 26. MemEvolver generates the target address; revised text is not re-quantized to assign a replacement SID. This separation permits experience to be updated without invalidating its address. It does not ensure that every revision remains relevant to earlier queries, or that the fixed address space represents all future experience equally well.

Memory-operation mid-training and joint policy post-training. Mid-training teaches the policies how to use SIDs within complete memory interactions, using teacher-provided reasoning traces and role-specific outputs under a next-token objective (Section 3). Post-training evaluates the resulting behaviour through SID, strategic-quality, and outcome rewards (Section 3). Demonstrations specify how to perform an operation; task feedback assesses whether the resulting support helps StackPlanner. The KL term in Eq. 9 regularizes deviation from the reference policy and is distinct from the supervised teacher targets used in mid-training.

Insertion and revision. The two-operation interface separates adding an entry from modifying an existing one (Section 2). Revision includes retention: if the trajectory provides no useful change, MemEvolver can emit `revise` with $\tilde{\mathbf{x}}_{e,t} = \mathcal{M}_t[\mathbf{z}_{e,t}]$, leaving the addressed payload unchanged. Consolidation and deletion likewise use revised or empty payloads rather than separate action labels. This interface allows selective updates without requiring a distinct `keep` action, but does not guarantee that the policy identifies the appropriate update. That behaviour must be assessed through update traces and downstream evaluation, not inferred from the operation vocabulary alone.

C.5 Stable Addresses and Editable Memory Content

Let $\mathbf{x}_t = \mathcal{M}_t[\mathbf{z}]$ denote the payload stored at an occupied address $\mathbf{z}$ at time t. During cold-start construction, experiences assigned to the same complete SID are consolidated into a single entry. During online evolution, MemEvolver selects a target SID and updates its payload according to Equation 26. Each occupied address therefore holds one current experience text.

An `insert` operation populates an empty address, while `revise` updates an existing entry. Revision encompasses retention, merging, and deletion through unchanged, merged, or empty payloads, respectively. As illustrated in Figure 6, revisions can incorporate new procedures, correct earlier errors, or refine applicability conditions without changing the SID or the codebooks.

This invariance concerns the address, not the meaning or utility of its contents. SID levels likewise have no predefined interpretation as domains, tools, or strategies: they arise from successive residual quantization. Shared prefixes indicate common codebook assignments, but do not establish independent semantic axes. The GPQA evaluation examines whether these prefixes provide useful task information.

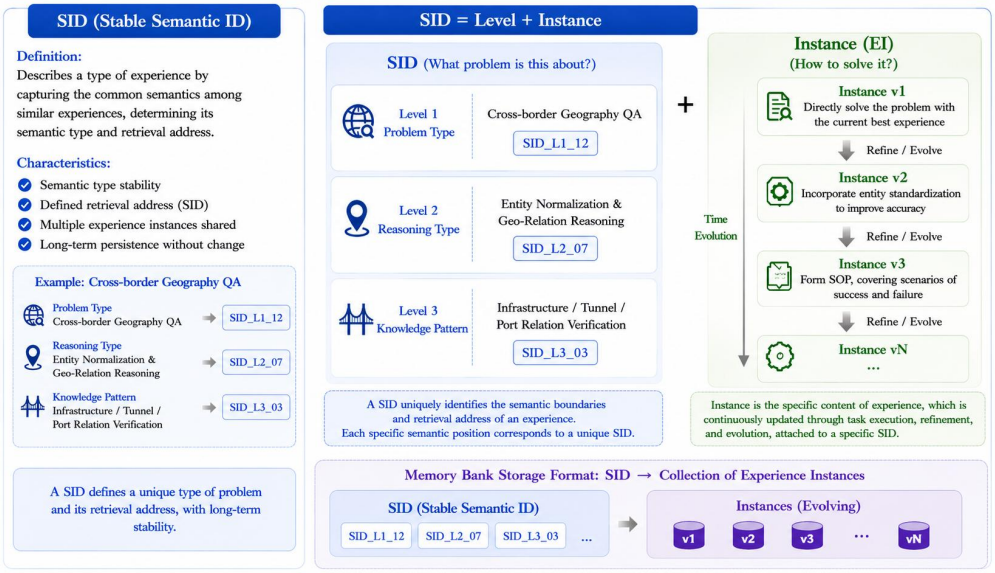

Figure 6: Conceptual illustration of SID-addressed memory and experience evolution. The illustrated versions represent successive contents at the same address. Semantic categories and the three-level SID are illustrative; the implemented system uses four residual-quantization levels without predefined semantic roles.

D Memory Bank Construction and Statistics

D.1 Cold-Start Experience Generation

The cold-start stage constructs the seed experience bank before memory-policy training (Section 2). Let $\mathcal{D}_{\text{train}}$ denote the set of training queries and $y_q^\star$ the reference answer for query q, with $y_q^\star = \varnothing$ when no reference answer is available. The notation below describes the information passed through trajectory collection, assessment, and extraction; these stages do not update model parameters.

Trajectory collection. For each query q, StackPlanner generates K_q attempts. We represent each recorded attempt and the resulting trajectory group as

$$\tau_q^{(k)} = (\xi_q^{(k)}, \hat{y}_q^{(k)}, g_q^{(k)}), \qquad \mathcal{T}(q) = \{\tau_q^{(k)}\}_{k=1}^{K_q}, \tag{30}$$

where $\xi_q^{(k)}$ is the recorded sequence of reasoning steps, actions, and observations, $\hat{y}_q^{(k)}$ is the final answer, and $g_q^{(k)}$ contains any environment reward or completion signals. Missing feedback is

represented by $g_q^{(k)} = \varnothing$, not by a failure label. The generation prompts request supporting evidence and verification steps so that subsequent extraction can refer to what the agent actually did.

Outcome and evidence assessment. The summarization prompts assess each attempt against the available reference answer or environment feedback. We denote retained assessment information by

$$\mathcal{A}(q) = \left\{(\tau_q^{(k)}, d_q^{(k)}, b_q^{(k)})\right\}_{k=1}^{K_q}, \qquad d_q^{(k)} \in \{\text{success}, \text{failure}, \text{unverified}\}, \tag{31}$$

where $d_q^{(k)}$ records whether the available evaluation evidence establishes success, establishes failure, or leaves the outcome unverified. For interactive tasks, a success claim requires an observed environment signal. The textual record $b_q^{(k)}$ captures decisive steps, supporting evidence, failure indications, and low-evidence caveats from the single-trajectory summary. It is not a scalar quality score. Outcome and evidence quality are distinct: a correct final answer without a documented procedure can still be marked as low-evidence. Failed and unverified attempts are retained for comparison, without inventing missing steps or treating uncertain outcomes as verified ones.

Experience extraction. An LLM extractor $\mathcal{E}$ uses the query, its available reference, and the assessed attempts to produce a set of textual experience summaries,

$$\mathcal{X}(q) = \mathcal{E}\big(q, y_q^\star, \mathcal{A}(q)\big). \tag{32}$$

Each $x \in \mathcal{X}(q)$ describes a reusable procedure or lesson, retaining its applicability conditions, decision rules, verification checks, and failure-recovery strategies. Successful attempts supply supported procedures; failed attempts reveal breakdowns and possible remedies; unverified attempts retain their uncertainty. This comparison is what we mean by *contrastive distillation*: text extraction from contrasting outcomes, not optimization of a contrastive loss. Aggregating the extracted summaries gives

$$\mathcal{X}_{\text{seed}} = \bigcup_{q \in \mathcal{D}_{\text{train}}} \mathcal{X}(q). \tag{33}$$

These summaries are the inputs x_i to the encoder and SID construction in Section 2. The single-trajectory and multi-trajectory prompts defining the assessment and extraction content are reproduced in Appendix I.2.

D.2 Memory Bank Data Statistics

This section reports the composition, scale, and compression statistics of the memory bank.

Overall pipeline. Raw skill crawling yields 171,820 entries; cluster-level deduplication reduces this to 16,384 canonical skills. Combined with SOP and factual trajectories, the pre-summary bank contains 138,243 entries, compressed into 16,416 final summaries via the LLM summarization stage (§2).

We first examine the composition of the pre-summary memory bank. As shown in Table 5, SOP and factual experience constitute the majority of the retained entries, while skill experience provides a smaller but complementary source.

Table 5: Composition of the 138,243 pre-summary experience entries.

Type	Count	Proportion
SOP experience	62,967	45.55%
Factual experience	58,892	42.60%
Skills experience	16,384	11.85%

We next report the source distribution of the retained experience entries. Table 6 shows that the memory bank draws from a broad collection of interactive, reasoning, and search-oriented datasets rather than being dominated by a single source.

Table 6: Number of retained experience entries by source after rollout generation, filtering, and experience distillation.

Source	Count	Source	Count
WebShop	30,000	HotpotQA	9,044
Skills	16,384	NQ	8,080
ALFWorld	12,007	GSM8K	7,188
2WikiMultiHopQA	11,336	MATH	6,892
TriviaQA	9,892	HealthBench	5,216
PopQA	9,596	GPQA	1,664
MuSiQue	9,312	DeepConsult	566
Bamboogle	500	DeepResearchGym	566

To characterize the effect of compression, Table 7 reports entry-length statistics before and after clustering and summarization. The pipeline substantially reduces both typical and extreme entry lengths, together with the overall text volume.

Table 7: Percentile statistics of entry length (characters) at each pipeline stage.

Stage	P50	P95	P99	Max
Skills (raw crawl)	4,748	20,720	40,253	5,425,092
Skills (post-cluster)	2,298	2,942	3,412	5,415
Memory (pre-summary)	8,131	20,208	26,071	123,653
Memory (post-summary)	3,116	10,274	19,075	50,204

Total text volume: $\sim 1.075 \times 10^9$ characters pre-summary, $\sim 1.18 \times 10^8$ post-summary ($\approx 9.1\times$ compression).

We further quantify how many raw experiences are consolidated into each final summary. As shown in Table 8, each summary merges 4.52 raw entries on average, while a small number of summaries aggregate substantially larger groups.

Table 8: Distribution of the number of raw entries merged into each summary.

Mean	P50	P90	P95	P99	Max
4.52	3	10	14	28	316

Finally, Table 9 characterizes the remaining long-summary tail after compression. Most summaries remain compact, with only a small fraction exceeding the larger character thresholds.

Table 9: Fraction of summaries at or above each character threshold.

Threshold	Count	Fraction
$\geq$4,096 chars	5,370	17.56%
$\geq$8,192 chars	2,368	7.74%
$\geq$16,384 chars	601	1.97%
$\geq$32,768 chars	12	0.039%

E Experimental Datasets and Evaluation Protocols

E.1 Evaluation Benchmarks

Table 10 maps the evaluation benchmarks to the main-text results and their reported metrics. ALFWorld and WebShop are in-domain interactive tasks. For search-augmented question answering, 2WikiMultiHopQA and HotpotQA are in-domain benchmarks, whereas Bamboogle, MuSiQue, Natural Questions, and TriviaQA are out-of-domain benchmarks. We further evaluate cross-domain generalization on BrowseComp-Plus (Chen et al., 2025b) for search, DeepResearch Bench (Du et al., 2026) for research, BIRD (Li et al., 2023) for SQL generation, and SWE-bench Verified (Jimenez et al., 2024) for software engineering. GPQA is used separately for the symbolic reasoning experiment in Section 5.

Table 10: Evaluation benchmarks and metric correspondence. In-domain and out-of-domain designations follow the evaluation protocol in the main text. GPQA is used separately for the symbolic reasoning analysis.

Benchmark	Evaluation role	Reported metric
ALFWorld	interactive task	Subtask and overall success rate
WebShop	interactive task	Task score and success rate
2WikiMultiHopQA (2Wiki)	In-domain QA	Answer F1
HotpotQA	In-domain QA	Answer F1
Bamboogle	Out-of-domain QA	Answer F1
MuSiQue	Out-of-domain QA	Answer F1
Natural Questions (NQ)	Out-of-domain QA	Answer F1
TriviaQA	Out-of-domain QA	Answer F1
BrowseComp-Plus	Out-of-domain search	Exact-match accuracy
DeepResearch Bench	Out-of-domain research	RACE
BIRD	Out-of-domain SQL	Execution accuracy
SWE-bench Verified	Out-of-domain code	Resolved rate
GPQA	Symbolic reasoning	Answer accuracy

The SID retrieval evaluation is distinct from these downstream benchmarks. Table 18 reports 2,000 evaluation examples for each MemRetriever configuration and 735 for MemEvolver. The latter comprise 590 `merge_existing` and 145 `insert_new` examples (Table 22); these are evaluation categories, not additional memory operations. Memory-bank construction counts are reported separately in Appendix D.2 and should not be interpreted as evaluation-set sizes.

Details to complete. Exact benchmark versions, splits, evaluated sample counts, and overlap checks against the seed bank remain to be documented. The proposed Deep Research, Code, and SQL transfer evaluations are not specified by named datasets and completed result tables in the current draft, so they are not included in Table 10.

E.2 Evaluation Metrics

Downstream task performance. For an evaluation set of n episodes, let $b_j \in \{0, 1\}$ denote benchmark-verified task success. The success rate is

$$\mathrm{SR} = \frac{100}{n} \sum_{j=1}^{n} b_j. \tag{34}$$

For ALFWorld, success is determined by the environment, not by matching an answer string. The Pick, Look, Clean, Heat, Cool, and Pick2 columns apply this measure to the corresponding task subsets; the All column reports overall success. WebShop reports both task score and success rate. The score averages graded task rewards, whereas success counts completed tasks satisfying the benchmark's success criterion. The exact score normalization and overall aggregation used by each baseline source remain part of the reproduction checks in Appendix F.1.

QA F1 and symbolic reasoning accuracy. For answer token precision P_j and recall Q_j on example j, token F1 is $F_{1,j} = 2P_jQ_j/(P_j + Q_j)$, with zero assigned when there is no token overlap. The QA tables report the mean example-level F1 as a percentage. Answer normalization, multiple-reference handling, and empty-answer conventions must follow the evaluation script used for each dataset and remain to be recorded. The Avg. column in Table 2 summarizes the six dataset-level scores, rather than pooling all questions into a single evaluation set. The GPQA experiment instead reports answer accuracy, computed as in Eq. 34 with b_j indicating a correct answer choice; it is not a retrieval hit rate.

Candidate and final retrieval hit rates. Let $\mathbf{z}_j^\star$ be the reference SID for evaluation example j, and let $\mathcal{A}_{j,k}$ contain the first k predicted addresses under a specified ranking. Define

$$H@k(\mathcal{A}) = \frac{100}{n} \sum_{j=1}^{n} \mathbb{I}[\mathbf{z}_j^\star \in \mathcal{A}_{j,k}]. \tag{35}$$

Candidate Hit applies this criterion to the full candidate set before reranking. R@k in Table 18 uses the beam-search ranking, whereas HR@k uses the reranked list. In Table 4, final Hit@k is computed on the reranked list for all methods. These single-reference hit rates measure whether the target is present, not the fraction of returned memories judged relevant; they are not precision@k.

Independent level and cumulative prefix accuracy. For $\mathbf{z} = (s^1, \ldots, s^L)$, write $\mathbf{z}_{1:\ell} = (s^1, \ldots, s^\ell)$. The two hierarchical metrics in Tables 20 and 21 are

$$\begin{aligned} H_\ell@k &= \frac{100}{n} \sum_{j=1}^{n} \mathbb{I}[\exists\, \mathbf{z} \in \mathcal{A}_{j,k} : s^\ell = s_j^{\star,\ell}], \\ H_{1:\ell}@k &= \frac{100}{n} \sum_{j=1}^{n} \mathbb{I}[\exists\, \mathbf{z} \in \mathcal{A}_{j,k} : \mathbf{z}_{1:\ell} = \mathbf{z}_{j,1:\ell}^\star]. \end{aligned} \tag{36}$$

Independent accuracy requires a match only at level ℓ. Cumulative accuracy requires all first ℓ symbols to match within the same candidate. At $\ell = L$, cumulative accuracy equals full-SID hit rate. Table 16 reports cumulative Top-5 accuracy after reranking, matching the corresponding rows of Table 21; its ROUGE-L and LLM-Judge values assess generated experience text separately and are not percentages of correct SIDs. The judge rubric and score normalization remain to be documented.

Reporting scope. The MemEvolver update-type breakdown reports SID retrieval performance within each operation category, not accuracy of choosing `insert` versus `revise`. Training rewards are also distinct from held-out task metrics. No statistical-significance claim is made here without repeated-run results and a specified testing procedure.

Cross-domain evaluation metrics. BrowseComp-Plus is evaluated using normalized exact-match accuracy. DeepResearch Bench is evaluated using RACE. BIRD is evaluated using execution accuracy, obtained by executing the predicted and reference SQL queries on the same database and comparing their returned results. SWE-bench Verified is evaluated using resolved rate under the official evaluation harness.

E.3 Online Evaluation Protocol

The online evaluation uses a strict temporal split. We construct the initial bank and fit the SID codebooks using only the training portion of each benchmark; the held-out portion is then presented once as an ordered query stream. We keep the stream order fixed for every method and repeat the evaluation over the same random seeds used for decoding. The online checkpoint starts with the same bank, model weights, encoder, and codebooks as the frozen-bank control. No gradient update, codebook refit, or access to held-out answers is permitted during memory updates.

For each incoming query q_t, the system generates one SID, retrieves the addressed support, and executes StackPlanner to obtain an answer $\hat{y}_t$ and trajectory τ_t. We score $(q_t, \hat{y}_t)$ immediately, then run the evolution policy on τ_t and apply its emitted `insert` or `revise` operation to obtain $\mathcal{M}_{t+1}$. Every fixed-size stream block is followed by evaluation on a disjoint probe set sampled once before the online run; probing is read-only and therefore cannot alter subsequent memory states. We store the bank snapshot and all operation logs at each probe point.

The primary curve plots probe task performance against the cumulative number of processed stream queries. We additionally report memory precision@k, SID prefix hit rate at each level, and operation frequencies (insertion, content-changing revision, and content-preserving revision). The frozen-bank control follows the identical query and probe schedule but skips the write step. Paired differences between the two curves quantify the contribution of online evolution, while bootstrap intervals over tasks and stream seeds provide uncertainty estimates.

F BASELINES, IMPLEMENTATION, AND TRAINING DETAILS

F.1 COMPARISON METHODS AND SETTINGS

ALFWorld and WebShop. Table 1 compares GPT-4o and Gemini-2.5-Pro; the prompt-based or memory-based methods ReAct, Reflexion, Mem0, MemP, ExpeL, and SimpleMem; the RL baselines RLOO and GRPO; and the memory-augmented RL methods MemRL, EvolveR, Mem0+GRPO, SimpleMem+GRPO, SkillRL, SkillGraph, AgentOCR, SIRI, and Skill0. The **GENMEM** and **GENMEM** (RL) rows distinguish the configurations before and after Harness-RL training of StackPlanner, as described in Section 4.2.

Search-augmented QA. Table 2 includes Base and CoT prompting, FS-RAG, FL-RAG, ReAct, IR-CoT, and TCRAG; the RL methods ReSearch, Search-R1, AEPO, ARPO, KBPO, Mem1, ZeroSearch, and AgenticRAG-R1; and EvolveR, SkillRL, SkillGraph, AgentOCR, Skill0, Skill-SD, SAPO, and SDAR. Method citations are provided in Section 4.1. Entries marked – are unavailable results, not zero scores, and should not enter performance rankings or averages.

Experience retrieval. The comparison in Table 4 uses Qwen3-Embedding-0.6B, TF-IDF, SkillRouter-Embedding-0.6B, and **GENMEM** with up to 50 candidates followed by BGE reranking to Top-5. All methods therefore share the same candidate budget and reranking protocol, enabling a matched-budget comparison of candidate and final retrieval hit rates.

Reproduction scope. The Qwen2.5-7B-Instruct backbone in Section 4.1 refers to the three **GENMEM** modules; Table 2 is labelled as a Qwen2.5-7B comparison. Neither statement establishes identical models or inference budgets for every baseline, particularly the proprietary models in Table 1. Per-row provenance (reported results or local reruns), checkpoint versions, retrieval settings, token and tool-call budgets, decoding parameters, and random seeds remain to be recorded before the comparisons can be described as controlled reproductions.

F.2 TRAINING HYPERPARAMETERS

We use Qwen2.5-7B-Instruct as the backbone for both memory policies and optimize them with GRPO during post-training. The SID codebook is fixed to the four-level configuration $48 \times 16 \times 8 \times 8$. Unless otherwise specified, all post-training runs use the hyperparameters summarized in Table 11.

Table 11: Hyperparameters for reinforcement learning

Hyperparameter	**Value**
Backbone model	Qwen2.5-7B-Instruct
SID codebook ($N_1 \times N_2 \times N_3 \times N_4$)	$48 \times 16 \times 8 \times 8$
Learning rate (alignment)	2×10^{-5}
Learning rate (GRPO)	3×10^{-5}
Global Batch size	256
GRPO group size G	8
GRPO rollout batch size	32
KL coefficient β	0.04
Training steps (GRPO)	200

G ADDITIONAL EXPERIMENTS AND ANALYSIS

G.1 RQ-KMEANS EVALUATION METRICS AND CONFIGURATION ANALYSIS

Prefix-prompt comparison. Table 12 compares encoding with and without the prefix, holding the seed bank, encoder, and reported Weighted setting ($w = 0.5$) fixed. This experiment uses a three-level $48 \times 24 \times 24$ codebook, not the four-level $48 \times 16 \times 8 \times 8$ configuration used in the main system. Under this three-level setting, the prefix reduces the reported reconstruction loss but also

slightly reduces leaf occupancy. It is not a direct ablation of prefix guidance in the final four-level system.

Table 12: Prefix-prompt comparison with a three-level $48 \times 24 \times 24$ codebook and the reported Weighted setting ($w = 0.5$). The encoder and seed bank are held fixed. Values and rounded differences are transcribed from the supplied experiment summary; raw-precision outputs remain to be checked.

Metric	Prefix	No prefix	Δ
Test reconstruction loss ↓	0.000332	0.000392	−15.27%
Single-code utilization (L1/L2/L3) ↑	100/100/100%	100/100/100%	0
Cumulative L1+L2 utilization ↑	84.72%	83.25%	+1.48 pp
Leaf-combination utilization ↑	23.75%	23.93%	−0.18 pp
Used leaves	6,567	6,617	−50
DUI uniformity ↑	0.8929	0.8730	+0.0199
ICR (independent-code rate) ↑	1.1487	1.1422	+0.0064
SID uniqueness ↑	47.50%	47.87%	−0.36 pp
RQ-KMeans training time	77.6 s	73.6 s	+4.0 s

RQ-KMeans codebook configuration analysis. We compare 40 codebook configurations spanning two to four quantization levels and address capacities from 27,648 to 147,456 (Table 13). Each configuration is evaluated under three reported distance settings (Euclidean, Spherical, Weighted $w = 0.5$), giving 120 combinations fitted on the same 138,243 pre-merge embeddings. The comparison examines the trade-off between address capacity, SID vocabulary size, and quantization quality, reporting reconstruction loss, leaf utilization, SID uniqueness, and DUI.

Table 13: Codebook search space. Each tuple lists the codebook sizes $(N_1, \ldots, N_L)$; counts in parentheses give the number of configurations at each depth.

Quantization depth	Codebook configurations
Two levels (3)	256×256, 320×256, 384×256
Three levels (30)	48×24×24, 48×32×32, 64×32×24, 64×32×32, 64×40×20 64×48×20, 64×48×24, 72×24×16, 72×32×24, 80×32×16 80×32×20, 80×32×32, 80×40×24, 80×48×16, 88×40×8 96×24×24, 96×32×16, 96×32×24, 96×40×16, 96×40×24 96×48×16, 96×48×24, 128×24×16, 128×24×24, 128×32×16 128×32×24, 128×40×16, 128×40×24, 144×40×24, 192×32×24
Four levels (7)	48×16×8×8, 48×24×16×8, 64×16×8×8, 64×16×16×8 64×24×8×8, 80×16×8×8, 96×16×8×8

Distance Metric Comparison. Table 14 summarizes each distance setting across the 40 architectures. The standard deviation describes variation across architectures, not uncertainty across repeated random seeds. Reconstruction loss is reported in units of $\times 10^{-4}$.

Table 14: Distance-setting comparison, with mean ± standard deviation across 40 architectures. Loss is reported in $\times 10^{-4}$.

Distance	Recon. Loss ($\times 10^{-4}$)	Leaf Util. (%)	Uniqueness Ratio	Avg. DUI (%)
Euclidean	3.38 ± 0.09	14.92 ± 3.68	0.0765 ± 0.0241	56.53 ± 3.71
Spherical	5.98 ± 1.60	5.14 ± 6.28	0.0270 ± 0.0375	81.90 ± 1.93
Weighted (w=0.5)	3.32 ± 0.10	49.15 ± 8.89	0.2523 ± 0.0719	88.36 ± 2.04

Selected configuration vs. competitors. Table 15 compares the selected configuration ($48 \times 16 \times 8 \times 8$, reported Weighted setting $w = 0.5$) with five alternatives in the 40K–60K capacity band. This

is a trade-off between vocabulary size and the reported quantization metrics, rather than a claim that one configuration is best on every measure.

Table 15: Configuration comparison within the 40K–60K capacity band. The selected configuration (**bold**) uses the fewest SID tokens among the configurations shown. Loss is reported in $\times 10^{-4}$.

Codebook	L	Capacity	Vocab Size	Recon. ($\times 10^{-4}$)	Leaf Util. (%)	Uniq. Ratio	DUI (%)
48×16×8×8	**4**	**49,152**	**80**	3.48	62.22	0.2212	90.70
128×24×16	3	49,152	168	3.31	64.25	0.2284	89.48
96×24×24	3	55,296	144	3.34	56.71	0.2268	88.74
72×32×24	3	55,296	128	3.36	51.26	0.2051	88.44
96×32×16	3	49,152	144	3.34	57.49	0.2044	87.53
64×40×20	3	51,200	124	3.36	52.93	0.1960	88.13

Configuration trade-offs. The reported Weighted setting has higher mean leaf utilization than the Euclidean setting (49.15% versus 14.92%), with similar reported reconstruction loss (3.32 versus 3.38, in units of 10^{-4}). These aggregate results compare distance settings; they do not isolate the effect of hierarchy depth. Within Table 15, the selected four-level configuration represents 49,152 possible addresses with 80 SID tokens, compared with 124–168 tokens for the listed three-level alternatives. Its leaf utilization is 62.22%, but its reported reconstruction loss is higher than that of every listed alternative. The configuration therefore favors a smaller vocabulary rather than minimum reconstruction error. These measurements do not establish effects on catastrophic forgetting, training convergence, or downstream task accuracy.

G.2 Pretraining and Mid-Training Retrieval Analysis

This section systematically compares SID retrieval quality across pretraining (PT) and mid-training (MT) stages for both the legacy 3-level and current 4-level codebooks. We evaluate MemRetriever and MemEvolver under beam search and reranking, reporting recall at multiple cutoffs and per-level hit rates. All retrieval evaluations use uniformly sampled training examples from the six search benchmarks listed in Table 3, together with ALFWorld and WebShop.

Table 16 gives the SID alignment and experience-reconstruction scores omitted from the main text to conserve space.

Table 16: SID alignment accuracy (%) at each cumulative hierarchical level, measured at Top-5 after reranking. For SID→experience reconstruction, ROUGE-L and LLM-Judge scores are reported separately.

Component	L_1	L_1–L_2	L_1–L_3	L_1–L_4
MemRetriever (query→SID)	40.90	13.00	6.30	3.75
MemEvolver (traj→SID)	40.41	11.29	3.27	1.36
SID→experience rewrite	**ROUGE-L**: 0.53		**LLM-Judge**: 0.57	

Model abbreviations used throughout:

- **PT-3L**: Pretrained MemRetriever (MemR) with 3-level codebook $(48 \times 24 \times 24)$.
- **PT-4L**: Pretrained MemR with 4-level codebook $(48 \times 16 \times 8 \times 8)$.
- **MT-3L**: Mid-trained MemR with 3-level codebook $(48 \times 24 \times 24)$.
- **MT-4L-Full**: Mid-trained 4-level COT MemR with full-sequence beam decoding.
- **MT-4L-SID**: Mid-trained 4-level COT MemR with SID-only beam decoding.
- **MT-4L-MemE**: Mid-trained 4-level COT MemEvolver (MemE) with training-isomorphic schema.

Codebook configuration Table 17 compares the two codebook configurations. The 4-level factorization offers a larger theoretical address space with shorter per-level vocabularies, reducing per-token prediction entropy during autoregressive SID generation.

Table 17: Codebook configuration: 3-level vs. 4-level SID.

Property	3-Level	4-Level
Number of levels	3	4
L1 size	48	48
L2 size	24	16
L3 size	24	8
L4 size	—	8
SID tokens per address	3	4
Theoretical capacity	$48 \times 24 \times 24 = 27{,}648$	$48 \times 16 \times 8 \times 8 = 49{,}152$

Full SID Retrieval Results Table 18 reports end-to-end SID retrieval performance across all six configurations, measuring beam-search recall (R@k) and reranking hit rate (HR@k) after dense rescoring.

Table 18: Full SID retrieval results across PT and MT configurations. R@k: beam recall; HR@k: reranking hit rate.

Model	N	Avg Cand.	R@1	R@5	R@10	R@20	R@50	HR@1	HR@5
PT-3L (48×24×24)	2000	49.98	1.15	4.30	7.30	11.25	20.80	2.25	4.85
PT-4L (48×16×8×8)	2000	49.57	1.40	4.95	7.90	13.40	20.25	7.00	13.15
MT-3L MemR (48×24×24)	2000	29.43	0.75	1.75	2.90	5.05	10.85	2.95	5.35
MT-4L COT MemR (Full Beam)	2000	33.72	0.40	1.05	1.50	2.20	4.90	1.85	3.10
MT-4L COT MemR (SID-only)	2000	50.00	0.35	1.15	1.80	3.35	6.80	2.25	3.75
MT-4L COT MemE	735	50.00	0.14	0.54	—	—	5.31	0.68	1.36

SID Decoding Configurations Table 19 compares greedy decoding, deterministic beam search, and beam sampling. Deterministic beam search uses no random sampling. Increasing its beam width from 5 to 20 raises Recall@5 from 7.93% to 8.53%, while reported latency increases from 121.0 to 406.1 ms. None of the sampled configurations exceeds the highest deterministic Recall@5 in this comparison.

Beam-search hit rates. Table 20 reports per-level independent and cumulative prefix hit rates under beam search. Each level is evaluated independently (left) or cumulatively from L1 through Lℓ (right), at cutoffs Top-1/5/50.

Rerank Hit Rates Table 21 reports per-level and cumulative metrics after reranking (Beam@50 candidates rescored to Top-1 and Top-5).

MemE Update-Type Breakdown Table 22 retains the original evaluation labels and values. The label `merge_existing` targets already-populated slots and corresponds to a merging outcome of `revise`, while `insert_new` targets unpopulated leaves and corresponds to `insert`.

SID Depth Selection: 3-Level vs. 4-Level We compare a **3-level** configuration $(N_1, N_2, N_3) = (48, 24, 24)$ against a **4-level** configuration $(N_1, N_2, N_3, N_4) = (48, 16, 8, 8)$ under identical training conditions (global batch size 48, 500-step moving average).

As shown in Figure 7, the 4-level SID achieves **consistently lower training loss** throughout alignment pretraining, converging faster and to a lower plateau. Finer factorization provides the autoregressive decoder with shorter per-level prediction targets and sharper categorical distributions, reducing per-token entropy. This motivates the adoption of the 4-level SID as the default configuration.

SID Codebook Utilization Analysis We analyze token-level utilization of the 4-level codebook $(48, 16, 8, 8)$ across the 138,243 pre-merge entries. Figure 8 reports the share of memories assigned to each token index per level, with normalized entropy and effective code counts.

Table 19: SID decoding configurations and retrieval results. Top-1 SID and Recall@5 are percentages; latency is reported in milliseconds. Greedy decoding returns one candidate, so its two retrieval scores coincide. Temperature and top-p entries are transcribed as reported and do not imply sampling in the deterministic rows. Bold denotes the best value in each result column, including ties.

Decoding	Beam	Top-k	Temp.	Top-p	Top-1 SID	Recall@5	Latency
Greedy	1	0	1.0	1.0	1.87	1.87	**35.7**
Beam deterministic	5	0	1.0	1.0	2.93	7.93	121.0
Beam deterministic	10	0	1.0	1.0	2.93	8.27	221.1
Beam deterministic	20	0	1.0	1.0	**3.13**	**8.53**	406.1
Beam sample	5	0	0.7	0.9	2.53	6.53	121.5
Beam sample	5	5	0.7	0.9	2.73	7.27	121.8
Beam sample	5	10	0.7	0.9	2.47	7.07	121.9
Beam sample	5	20	0.7	0.9	2.93	6.87	122.0
Beam sample	5	20	0.3	0.9	2.73	7.73	122.0
Beam sample	5	20	0.5	0.9	2.87	7.40	122.0
Beam sample	5	20	1.0	0.9	2.60	5.67	122.1
Beam sample	10	20	0.7	0.9	**3.13**	7.27	224.3
Beam sample	20	20	0.7	0.9	3.07	8.27	413.6
Beam sample	5	20	0.7	0.8	2.67	6.60	121.8
Beam sample	5	20	0.7	0.95	2.53	6.20	122.0

Table 20: Beam search hit rates (%) across PT and MT configurations. Left: per-level independent accuracy. Right: cumulative prefix accuracy (L1 through Lℓ must all match). L4 is N/A for 3-level models.

		Per-Level Independent (%)				Cumulative Prefix (%)			
Model	**Cutoff**	**L1**	**L2**	**L3**	**L4**	**L1**	**L1+L2**	**L1…L3**	**L1…L4**
PT-3L	Top-1	31.85	24.80	14.00	—	31.85	7.85	1.15	—
	Top-5	53.85	35.40	26.60	—	53.85	18.50	4.30	—
	Top-50	88.20	69.55	67.05	—	88.20	49.95	20.80	—
PT-4L	Top-1	22.05	26.75	31.65	33.60	22.05	7.10	2.95	1.40
	Top-5	44.25	42.30	47.50	52.05	44.25	18.80	9.05	4.95
	Top-50	85.55	78.60	81.25	84.10	85.55	52.65	32.05	20.25
MT-3L	Top-1	14.40	23.80	16.60	—	14.40	3.70	0.75	—
	Top-5	22.95	28.50	23.00	—	22.95	7.50	1.75	—
	Top-50	66.35	56.45	59.65	—	66.35	29.00	10.85	—
MT-4L-Full	Top-1	14.15	20.20	24.80	25.30	14.15	2.90	1.05	0.40
	Top-5	26.20	27.05	37.90	35.95	26.20	7.00	2.50	1.05
	Top-50	65.20	57.30	69.90	70.30	65.20	25.30	11.25	4.90
MT-4L-SID	Top-1	14.30	20.10	25.40	25.40	14.30	3.05	1.00	0.35
	Top-5	32.20	29.10	42.10	40.40	32.20	8.70	3.10	1.15
	Top-50	72.35	65.55	77.75	76.85	72.35	32.30	14.95	6.80
MT-4L-MemE	Top-1	16.60	19.73	23.27	25.31	16.60	3.67	0.41	0.14
	Top-5	33.88	26.26	41.09	42.99	33.88	7.62	1.77	0.54
	Top-50	77.55	57.69	75.37	87.62	77.55	30.20	12.24	5.31

Key observations:

- **High overall utilization.** All four levels achieve normalized entropy ≥ 0.97, indicating that the RQ-KMeans construction distributes memories broadly across the codebook rather than collapsing onto a few dominant codes. The effective code counts—43.3/48 at L1, 15.6/16 at L2, 7.9/8 at L3, and 7.9/8 at L4—confirm that nearly all codebook entries are actively used.
- **Mild imbalance at L1.** While L2–L4 are close to uniform, L1 exhibits moderate variation (range $\approx 0.1\%$–4.1% vs. the 2.08% uniform baseline). A small number of L1 tokens (e.g., indices 6 and 52) are under-utilized, reflecting the inherent non-uniformity of task-domain frequencies in the seed data. This motivates the long-tail evolution direction discussed in the Future Work (Section 7).

Table 21: Rerank hit rates (%) (Beam@50 rescored by dense reranker). Layout matches Table 20. L4 is N/A for 3-level models.

		Per-Level Independent (%)				Cumulative Prefix (%)			
Model	**Cutoff**	**L1**	**L2**	**L3**	**L4**	**L1**	**L1+L2**	**L1…L3**	**L1…L4**
PT-3L	Top-1	31.80	22.05	12.60	—	31.80	7.20	2.25	—
	Top-5	59.55	39.25	28.80	—	59.55	19.35	4.85	—
PT-4L	Top-1	26.45	28.45	32.60	32.60	26.45	11.25	8.25	7.00
	Top-5	54.45	50.25	57.35	59.85	54.45	25.70	16.40	13.15
MT-3L	Top-1	20.05	24.55	15.85	—	20.05	6.50	2.95	—
	Top-5	42.80	36.60	34.15	—	42.80	15.10	5.35	—
MT-4L-Full	Top-1	16.60	21.20	25.55	24.45	16.60	4.65	2.40	1.85
	Top-5	40.05	35.45	48.00	48.65	40.05	12.10	5.35	3.10
MT-4L-SID	Top-1	17.60	21.00	24.95	25.00	17.60	5.10	2.85	2.25
	Top-5	40.90	37.70	49.60	49.65	40.90	13.00	6.30	3.75
MT-4L-MemE	Top-1	16.87	19.46	23.81	22.99	16.87	3.54	1.09	0.68
	Top-5	40.41	30.88	45.71	50.88	40.41	11.29	3.27	1.36

Table 22: MemE retrieval performance by update-operation type.

Type	**N**	**R@1**	**R@5**	**R@50**	**Avg Cand.**
`merge_existing`	590	0.169	0.339	5.763	50.00
`insert_new`	145	0.000	1.379	3.448	50.00
Total	735	0.136	0.544	5.306	50.00

- **Finer levels are more balanced.** As the codebook size decreases from 48 (L1) to 8 (L3/L4), the per-token distribution converges toward uniform, consistent with the residual quantization design: later levels partition progressively smaller residual spaces where the variance is more evenly spread.

These results validate the Cartesian-product design: the 4-level factorization ($48\times16\times8\times8 = 49{,}152$ addressable regions) achieves high utilization without balancing regularization during construction.

G.3 Training Dynamic

Figure 9 shows training loss during mid-training for both MemRetriever and MemEvolver. This stage fine-tunes the alignment checkpoint on ReAct-style chain-of-thought traces with embedded SID generation. Both modules converge smoothly, confirming that mid-training successfully bridges SID alignment and GRPO post-training without instability.

Figure 10 presents the MemRetriever GRPO post-training curves, tracking both eligibility rates and reward signals over the course of training.

Figure 11 reports the corresponding MemEvolver GRPO diagnostics, covering answer eligibility, retrieval hits, SID matching, and the reward signals used to train the evolution policy.

StackPlanner RL training diagnostics. We report LoRA training diagnostics for StackPlanner on Search, ALFWorld, and WebShop. The Search and WebShop runs optimize the central agent with oracle memory and oracle actions, respectively. The ALFWorld run instead optimizes the Interactor while keeping the central agent frozen. It uses mixed memory conditions, with four oracle-assisted and four no-memory rollouts per query. These curves describe task-execution training, not MemRetriever/MemEvolver training or held-out performance of the complete **GenMem** system.

The runs contain 159, 65, and 131 training steps, covering 3,776, 1,035, and 1,042 training queries on Search, ALFWorld, and WebShop, respectively. Search and WebShop use four rollouts per query; ALFWorld uses eight, yielding 8,280 rollouts over one complete epoch and 65 optimizer updates. For Search, we retain the committed training lineage and exclude three superseded log records. The horizontal axis denotes the logged training-step counter for each run.

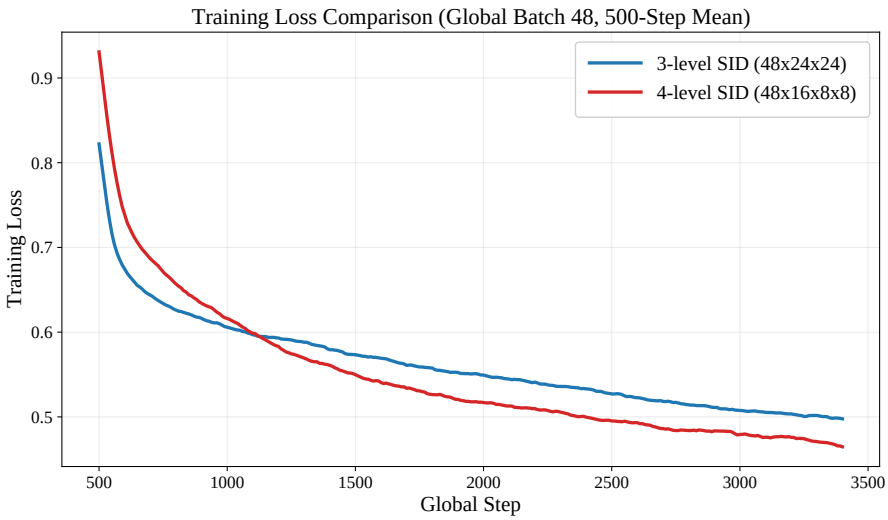

Figure 7: Alignment pretraining loss: 3-level vs. 4-level SID (500-step moving average, batch size 48). The 4-level SID converges faster and to a lower plateau.

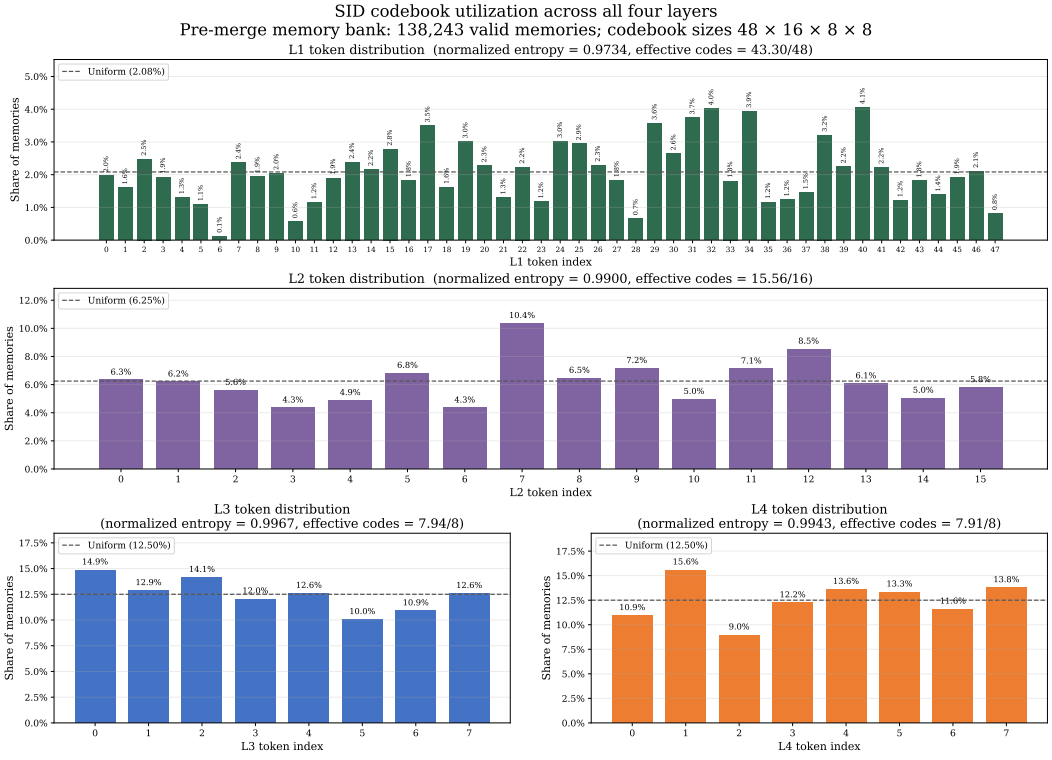

Figure 8: SID codebook utilization across four levels (138,243 pre-merge entries). Bars show the fraction assigned to each token index; dashed line = uniform baseline. Normalized entropy and effective code counts annotated per level.

As shown in Figure 12, light lines denote per-step values and dark lines trailing means, using task-specific smoothing windows: rollout-weighted 10-step for Search, unweighted 5-step for ALFWorld, and unweighted 10-step for WebShop. Shorter prefixes are used before a full window is available. Loss normalization differs across tasks, so loss magnitudes are not directly comparable. Each column represents a single run without across-seed uncertainty estimates.

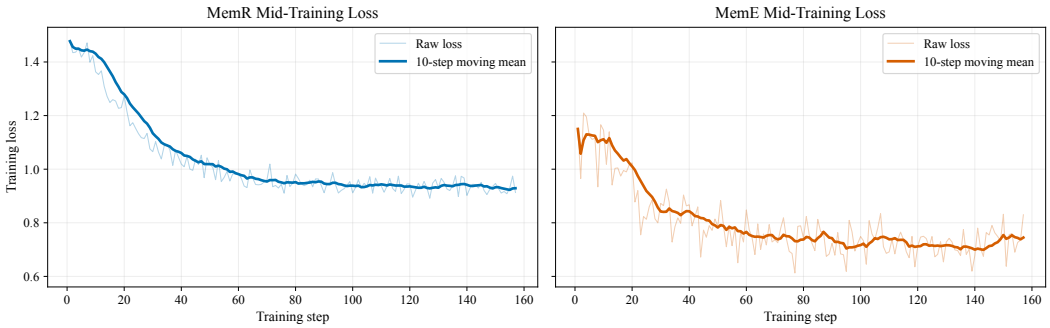

Figure 9: Mid-training loss for MemRetriever (left) and MemEvolver (right). Both converge smoothly on ReAct-style traces with embedded SID generation.

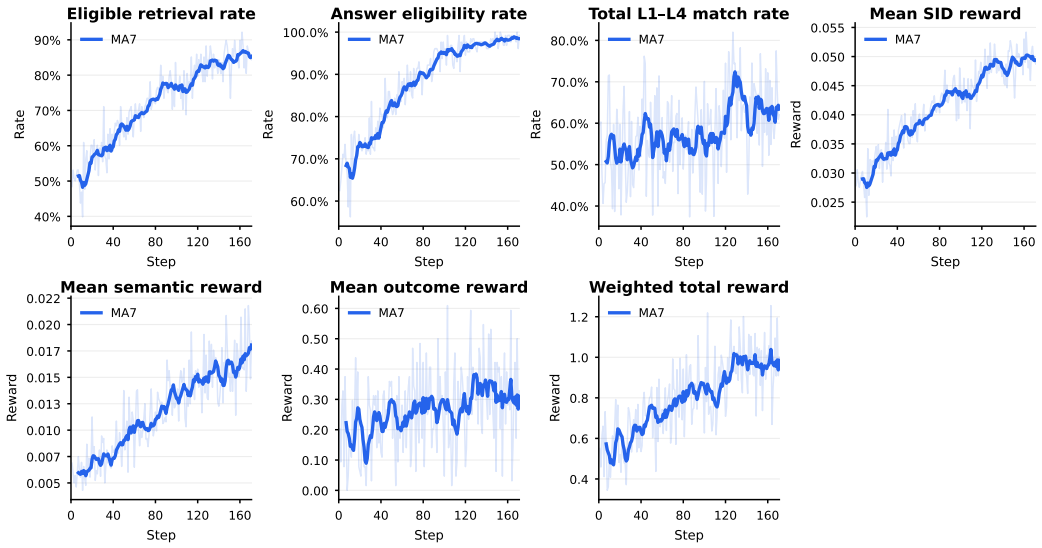

Figure 10: MemRetriever GRPO post-training curves (MA7-smoothed), tracking three eligibility rates and four reward signals over training steps. All metrics show steady improvement, confirming that GRPO effectively optimizes both SID targeting accuracy and downstream task utility.

Search reward is token F1, whereas the interactive tasks use their logged task rewards. The sampled training queries change across steps, and the GRPO surrogate losses are not supervised cross-entropy losses. Consequently, these curves alone neither establish held-out improvements nor isolate the contribution of learned memory. They are not used to populate the main result tables. De-identified per-step data and the plotting script are retained with the manuscript sources.

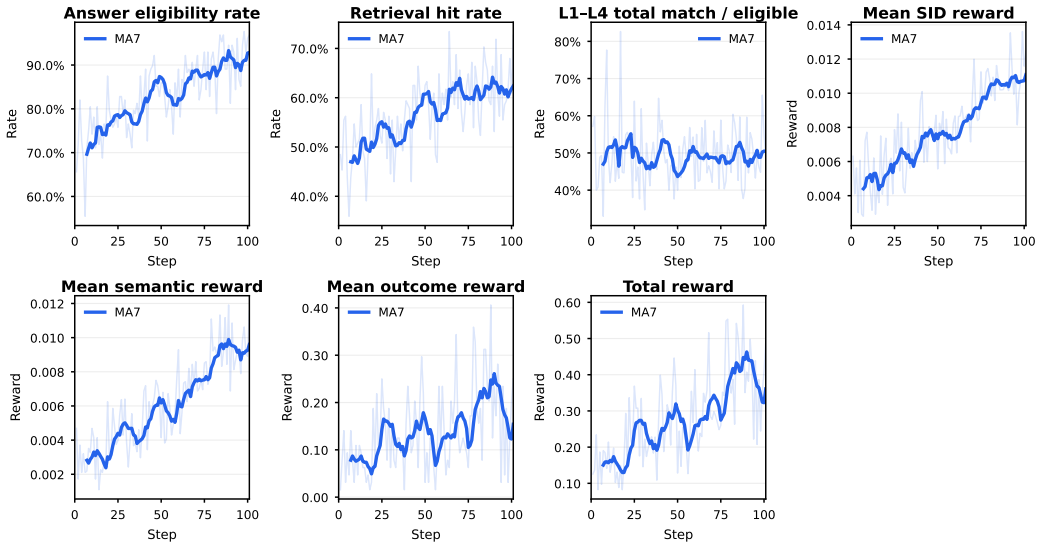

Figure 11: MemEvolver GRPO post-training curves (MA7-smoothed). The first two panels report answer eligibility and retrieval hits, respectively, each divided by the total number of rollouts. The remaining panels track SID matching, SID and semantic rewards, outcome reward, and total reward.

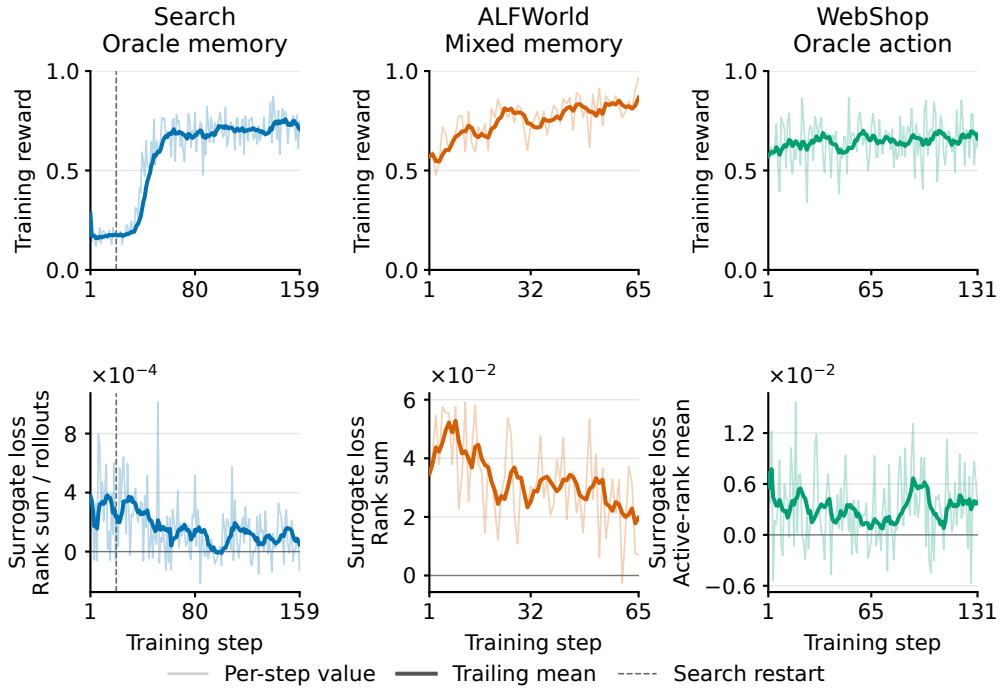

Figure 12: StackPlanner RL training diagnostics. Search and WebShop use oracle-assisted central-agent training, while ALFWorld trains the Interactor with a 1:1 mixture of oracle-assisted and no-memory rollouts. Top panels show mean training reward and bottom panels show logged GRPO surrogate loss. The dashed line marks the Search restart after checkpoint 20.

G.4 Ablation Study

Table 23 provides a component-level analysis of **GenMem** under the same self-evolution setting. We evaluate each variant on 200 samples for three complete evolution rounds, while modifying one component at a time.

Table 23: Ablation of key **GenMem** components.

Variant	description	Avg. Accuracy	Token F1
GenMem	full	**49.5**	**52.1**
w/o MemEvolver	frozen memory bank	46.5	46.8
w/o discriminative retrieval	TF-IDF + evolution	40.5	43.4
w/o discriminative retrieval	embedding + evolution	37.0	41.2
w/o MemRetriever	query only, no memory	41.0	45.2
w/o RL post-training	MT MemRetriever + MT MemEvolver	38.5	42.6
w/o beam search	greedy SID decoding	42.0	45.7

Table 23 isolates the contribution of each major component under the same three-round self-evolution protocol. The *w/o MemEvolver* variant freezes the memory bank, removing experience refinement while retaining memory retrieval. The *w/ discriminative retrieval* variant replaces generative SID retrieval with a content-based retriever, instantiated using an embedding model or TF-IDF, while keeping memory evolution unchanged. The *w/o MemRetriever* variant removes memory access entirely and feeds only the original query to the task model. The *w/o RL post-training* variant uses the pretrained MemRetriever and MemEvolver checkpoints without reinforcement-learning optimization. Finally, the *w/o beam search* variant replaces beam-based SID generation with greedy decoding.

Together, these variants separate the effects of experience evolution, symbolic retrieval, policy optimization, and decoding strategy. In particular, full vs. *w/o MemEvolver* measures the benefit of iterative memory refinement; full vs. *w/ discriminative retrieval* measures the effect of generative SID-based retrieval under the same evolution process; and full vs. *w/o MemRetriever* measures the overall contribution of external memory access.

G.5 GPQA Evaluation Protocol

The GPQA experiment tests whether SID tokens support downstream reasoning without access to experience text, and whether rewriting retrieved experience improves its utility. We compare seven input conditions and report answer accuracy in Figure 4.

Experimental conditions. For each query, MemRetriever generates k candidate SIDs with beam size $k \in \{1, 5, 10\}$. Symbolic inputs follow *level-first* order: all first-level tokens precede the second-level tokens, followed by subsequent levels where applicable. Each condition supplies the query together with the following information.

1. **Raw**: no additional information.
2. **+**L_1: first-level SID tokens only.
3. **+**$L_1..L_2$: the first two SID levels.
4. **+**$L_1..L_3$: the first three SID levels.
5. **+Full SID**: complete four-level SIDs, without experience text.
6. **+Rewrite**: a task-adaptive rewrite of experience retrieved using the complete SIDs.
7. **+Retrieved Memory**: experience text retrieved using the complete SIDs, without rewriting.

Conditions 1–5 vary SID depth without providing a memory payload. Conditions 5–7 compare complete symbolic addresses with rewritten and unmodified experience text. Both text conditions access the memory bank. Rewrite follows the lookup-then-rewrite procedure in Section 2, so it does not measure reconstruction of experience from SID tokens alone.

Results and interpretation. Complete SIDs improve accuracy from the raw-query baseline of 48.32% to 54.06–54.66% across the three beam settings, corresponding to 5.74–6.34 percentage

points. Rewritten experience reaches 55.65–56.91%, compared with 56.09–56.63% for directly retrieved text. Direct retrieval performs better at $k = 1$ and $k = 5$; rewriting performs better at $k = 10$. The comparison therefore supports the utility of SID input in this evaluation, with no consistent advantage from rewriting retrieved text. Independent semantic axes and generalization to unseen SID combinations remain unestablished. Assessing these properties requires additional controls, including held-out combinations and shuffled or reassigned SID tokens.

H CASE STUDIES

We present three representative cases from the trained **GENMEM** system (40 initial MemRetriever training steps, 23.5 hours). Each case shows the complete execution trace of MemRetriever, StackPlanner, and/or MemEvolver.

H.1 SYMBOLIC REASONING AND ONLINE EVOLUTION TRACES

The following qualitative cases illustrate how SIDs can be manipulated as structured symbols and how the same memory address can accumulate more reliable experience during online evolution. We transcribe the operation, intermediate memories, and generated result so that each example can be read as a complete case rather than as a schematic diagram. These examples clarify the operation semantics but do not replace the quantitative evaluation.

Symbolic Case 1: Address Fusion ($A + B \to C$)

Inputs. Two source experiences are addressed by `<SID_L1_33><SID_L2_4><SID_L3_14><SID_L4_6>` and `<SID_L1_32><SID_L2_23><SID_L3_10><SID_L4_2>`. The first records a failure caused by relying on a single source (e.g., Wikipedia) when checking a director's birth date. The second records a related failure in which conflicting source data led to an incorrect date. Both entries therefore emphasize multi-source verification and arithmetic or timeline checks.
Operation. The fusion prompt asks the model to combine the shared problem type, evidence requirements, failure modes, and reusable procedures, while rejecting unsupported facts and avoiding a simple concatenation of the two entries.
Output. The generated experience is written to `<SID_L1_6><SID_L2_2><SID_L3_5><SID_L4_6>` and states: use multiple authoritative sources (official biographies, film databases, and public records), cross-check dates against the person's work and career timeline, resolve source conflicts in favor of the more reliable source, and confirm the final answer with at least two independent references.
Interpretation. The target is not a textual concatenation of the two sources. The model extracts their common verification structure and writes it to a new symbolic address, showing how multiple addresses can provide ingredients for a reusable experience.

Symbolic Case 2: Address Decomposition ($C \to A, B$)

Target. The target experience at `<SID_L1_25><SID_L2_13><SID_L3_2><SID_L4_4>` concerns identifying a person's main occupation and returning a concise, normalized description. Its abstract procedure is to identify the entity, obtain a short biography, extract the key occupation label, and normalize multiple occupations when necessary.
Operation. A judge compares candidate source pairs according to shared or complementary skills. The selected pair is `A=<SID_L1_14><SID_L2_13><SID_L3_2><SID_L4_4>` and `B=<SID_L1_25><SID_L2_2><SID_L3_7><SID_L4_4>`. A contributes entity disambiguation for generic roles or names; B contributes structured biographical search and normalized occupation extraction.
Output. The reconstructed target `C′=<SID_L1_25><SID_L2_13><SID_L3_2>` preserves the target task and makes the supporting steps explicit: identify ambiguous entities or occupations, disambiguate with context or authoritative sources, extract the most specific supported occupation, and return it in a standardized form.

Interpretation. Decomposition retrieves complementary subskills rather than copying one memory entry. The resulting card preserves the target task while making the supporting operations explicit and reusable.

Online Evolution Case: <SID_L1_6><SID_L2_19><SID_L3_7><SID_L4_2>

This case follows one literary-question memory through iterations 0, 3, and 5. The address remains fixed while the payload is revised after new trajectories.

Iteration 0.
The initial experience says that literary-work questions require identifying the work, author, and relevant characters, then checking the requested details against the original text. It warns that secondary summaries can mix up similar characters and names.

Iteration 3.
A later trajectory adds an explicit source hierarchy: recall the canonical work and character first, consult the original text or an authoritative literary reference, and reject conclusions supported only by secondary analysis when the original and the reference disagree.

Iteration 5.
The evolved entry specializes the rule for questions about characters and roles. It requires extracting the exact name and role from the primary text, checking the surrounding context and timeline, and re-running the original-text verification whenever the character identity is ambiguous.

Interpretation. The iterations show selective revision rather than blind appending. Each update retains the transferable literary-analysis procedure, removes ambiguity, and adds verification requirements learned from the new trajectory. The case therefore illustrates the invariance–evolution trade-off: content is corrected and enriched without invalidating the learned address used to retrieve it.

Case 1: Factual Retrieval and Multi-hop Reasoning

Query: "Simpleton Tunnel links Italy with which country?"

MemRetriever execution:

- Memory Lookup: ⟨SID_L1_56⟩⟨SID_L2_1⟩⟨SID_L3_1⟩⟨SID_L4_6⟩
- **Retrieved experience:**
  - *Problem addressed:* Handling geographic-name queries; quickly identifying target place-name entities and query intent (geographic location, administrative division, historical changes, transportation, surrounding facilities, neighborhood districts, etc.).
  - *Practice:* Strictly follow standard geographic naming conventions; precisely match official administrative divisions and geographic information; avoid confusing colloquial or informal names; support multi-language translation and phonetic/approximate name variants.
- **Rewritten experience:** Recognize place-name misspellings (especially for cross-border infrastructure like tunnels and bridges). *Practice:* Parse the target entity in the question, note common spelling variants; search standard names and key evidence (location, endpoints); identify the two connected countries (excluding the one already given); output the other country as the answer.

StackPlanner execution:

- **Input:** [Query] + [Rewritten experience]
- **Reasoning:** The user asks about the "Simpleton Tunnel" (likely "Simplon Tunnel") connecting Italy to another country. Since the question concerns external factual information, a web-search verification is appropriate.

- **Dispatches Research Agent:** Search for the correct name of "Simplon Tunnel" and the countries it connects, focusing on basic infrastructure linking Italy to neighboring countries.
- **Research Agent findings:**
  - Simplon Tunnel – Wikipedia: "connects Brig and Domodossola, Italy, through the Alps" (score=0.891)
  - Simplon Pass – Wikipedia: "connects Brig in the canton of Valais with Domodossola in Piedmont (Italy)" (score=0.788)
- **Research report returned to lead agent:** The correct name is the **Simplon Tunnel** (German: Simplontunnel; Italian: Traforo del Sempione). It connects **Brig, Switzerland** and **Domodossola, Italy**, under the Lepontine Alps. Length: 19.8 km, 1906–1982 the world's longest railway tunnel. Construction: started 1898, completed with two single-track bores 15 years apart. Purpose: high-altitude trans-border freight and passenger rail. Connected countries: **Switzerland** (Valais) and **Italy** (Piedmont).

Answer: Switzerland

Case 2: Mathematical Reasoning + Memory Evolution

Query: "A coach is arranging a starting 5-player lineup for a basketball team. There are 12 players in total, and Ace and Zeppo must be in the starting lineup. How many different lineups are there?"

StackPlanner execution:

- **Input:** [Query]
- **Reasoning:** The user asks: given two players already confirmed as starters, how many ways to form the remaining lineup. This is a fixed-element combination counting problem; prior experience can be directly applied.
- **Dispatches Research Agent:** Calculate the number of ways to choose 3 starters from the remaining 10 players given Ace and Zeppo are fixed.
- **Research findings:**
  - Fixed starters: 2 (Ace, Zeppo)
  - Remaining player pool: 10
  - Additional players to select: 3
  - Selection order does not affect the result $\rightarrow$ use combination
  - $C(10, 3) = 120$
- **Conclusion:** There are 120 different starting lineups.

MemEvolver execution:

- MemE Lookup: ⟨SID_L1_3⟩⟨SID_L2_8⟩⟨SID_L3_4⟩⟨SID_L4_2⟩
- **Retrieved experience at this SID:** Handling combination counting problems. *Practice:* Compute the number of selectable objects and the number still to be chosen; determine whether order matters; if order-independent, use $C(n, k)$.
- **Rewritten (evolved) experience:** Handle fixed-member combination selection problems (e.g., fixed starting players, fixed candidates, fixed seat positions). *Practice:* Identify which members are already fixed, so they need not be re-selected; update the remaining candidate count and remaining selection count; determine if the problem is combination or permutation; if order-independent, use $C(n, k)$; output the final count and verify that it satisfies all constraints.

Answer: 120

Case 3: Memory Evolution at an Unpopulated Leaf

Query: "Ian won $100 in a lottery. He decided to use it to pay off debts. First he paid Colin $20; then he paid Helen double what he paid Colin; finally he paid Benedict half what he paid Helen. How much does Ian have left?"

StackPlanner execution:

- **Input:** [Query]
- **Reasoning:** The user asks about the remaining amount after sequential payments involving proportional relationships. This requires step-by-step sequential computation.
- **Dispatches Research Agent:** Compute Ian's remaining balance after three payments.
- **Research findings:**
  - Initial amount: $100
  - Payment to Colin: $20
  - Payment to Helen: $20 \times 2 = \$40$
  - Payment to Benedict: $40 \div 2 = \$20$
  - Total paid: $20 + 40 + 20 = \$80$
- **Conclusion:** Remaining = $100 - 80 = \$20$.

MemEvolver execution:

- MemE Lookup: ⟨SID_L1_20⟩⟨SID_L2_11⟩⟨SID_L3_0⟩⟨SID_L4_3⟩
- **Retrieved experience at this SID: Leaf unpopulated**—no existing experience stored.
- **Newly inserted experience:** Handle problems involving initial amounts, sequential payments, and proportional relationships between payment amounts. *Practice:* First record the initial amount; describe each payment in order following the problem statement; for proportional terms like "double" or "half," always compute from the immediately preceding payment amount; complete one payment at a time, updating the remaining balance; finally output the remaining amount and verify that all payment amounts sum correctly.

Answer: $20

I Full Prompt Specifications

For completeness and reproducibility, this appendix collects the verbatim prompts used across the **GenMem** pipeline: experience generation, trajectory summarization, skill-cluster and SID-leaf synthesis, embedding generation, StackPlanner outcome-reward rollouts, the MemRetriever (MemR) runtime, and the MemEvolver (MemE) runtime. Placeholders in braces (e.g., `{question}`) are filled at runtime.

I.1 Experience Generation Prompts

General-purpose datasets

```
You are solving a benchmark training question.

Answer the question as accurately as possible. Use StackPlanner tools
or subagents only when they are genuinely useful.
Leave a compact, observable solve trace before the final answer so
this run can be distilled into reusable experience later.

For factual, entity-comparison, multi-hop, or time-sensitive questions,
 do not answer from a bare guess. Verify the key entities/attributes
with available tools or with explicit evidence from the provided
context. If no tool is needed or available, still write the evidence/
checks you used.
```

```
Use this visible structure:
Evidence:
- <key fact, calculation, or tool result used>
Verification:
- <answer-format check, entity disambiguation, arithmetic check, or
uncertainty note>

Question:
{question}

Return your final answer in this exact trailing form:
Final answer: <answer>
```

WebShop

```
You are collecting a WebShop benchmark experience.

Goal: satisfy the shopping instruction by selecting the best matching
product or product option.
You must use the available WebShop benchmark tools when they exist:
1. call `webshop_search` with a query and the full instruction,
2. call `webshop_inspect` on promising ASINs,
3. call `webshop_select` with the final ASIN and selected options to
obtain the benchmark reward/success signal.
If these tools are not available, explicitly say the result is an
offline selection and do not claim verified success.

Shopping instruction:
{question}

Available metadata excerpt:
{metadata}

Focus on reusable SOP details: query formulation, attribute filtering,
 price/option matching, product-page verification, final selection,
and the observed reward/success signal.

Return your final answer in this exact trailing form:
Final answer: <product id / asin / selected product>
```

ALFWorld

```
You are collecting an ALFWorld/TextWorld household-task experience.

Goal: complete the embodied household task. The ALFWorld tools are
enabled in this StackPlanner session.
You must call the tools; do not produce an offline plan before trying
them.
1. call `alfworld_reset` with the task filename,
2. inspect the returned observation and admissible commands,
3. call `alfworld_step` repeatedly with admissible commands until the
environment returns done/won, or until progress is clearly blocked,
4. report the final score/done/won signal.
Only mark the result as offline/unverified if `alfworld_reset` itself
returns an explicit tool error. If a prior memory says tools were
unavailable, ignore that memory for this run.

Task/environment description:
{question}
```

```
Available metadata excerpt:
{metadata}

Use ALFWorld-style action thinking: inspect the room, find relevant
objects, open/close containers if needed, take objects, navigate to
target receptacles, put/use/clean/heat/cool as required, and verify
completion from the environment score/done/won fields. Preserve
reusable SOP details and common failure checks.

Return your final answer in this exact trailing form:
Final answer: <completed / failure reason with final score/done/won>
```

I.2 Trajectory Summarization Prompts

Single-trajectory summary

```
You compress agent execution traces into faithful, reusable trajectory
 summaries.

Return only valid JSON. All values must be strings. Use exactly these
fields:
run_id, evaluation, prediction, trajectory_summary, decisive_steps,
evidence_or_calculation, success_signal, failure_signal,
low_evidence_notes.

Rules:
- Preserve what the run actually did; do not invent tools, sources,
calculations, or evidence.
- trajectory_summary should be a numbered step list, not a one-line
abstract.
- decisive_steps should name the step(s) that most affected the final
answer.
- evidence_or_calculation should record the concrete evidence,
equation, lookup, environment observation, or answer-format check
visible in the run.
- success_signal should explain why the run is reusable if it is
correct; otherwise say no verified success trajectory exists.
- failure_signal should explain the likely divergence if the run is
wrong/unverified/low-evidence; for correct runs, record realistic
residual risks.
- If the run only has a direct final answer, mark it low-evidence
instead of pretending a procedure was observed.
```

Multi-trajectory summary

```
You extract rich, reusable experience from repeated benchmark attempts.


Return only valid JSON. All values must be strings. Use exactly these
fields:
skill_name, problem_type, problem_addressed, applicability,
preconditions, procedure, decision_rules, verification_checks,
failure_recovery, anti_patterns, practice, lessons_learned,
success_experience, failure_reflection, steps_summary, summary,
retrieval_cues, failure_modes, verification, evidence.

Write a useful memory for a future agent, not a generic checklist.

Requirements:
- Use the ground truth/reference as the anchor.
```

```
- Compare correct, failed, and unverified trajectories.
- Treat a citation as evidence only when the visible source content
directly supports the claim; preserve low-evidence caveats instead of
upgrading weak or tangential sources into verified facts.
- success_experience should be 2-4 short paragraphs when enough
evidence exists: what correct trajectories actually did, what was
decisive, what variation is safe, and how to reuse it.
- failure_reflection should be 2-4 short paragraphs when failures
exist: wrong predictions, divergence point, likely root cause, warning
 signs, and exact repair protocol.
- summary should be 3-6 compact paragraphs with these ideas: outcome
contrast, success pattern, failure pattern, reusable recipe,
verification gates.
- procedure and practice should be executable enough for a future
agent to follow.
- Do not claim success for WebShop/ALFWorld/offline interactive tasks
without a real reward or environment signal.
- Do not average the attempts into vague advice; preserve the contrast
 that made the group informative.
```

Interactive-dataset rich summary

```
Generate a reusable rich summary memory for one real interactive
benchmark run.

Return only valid JSON with exactly one string field: rich_summary.

This rich_summary is the main text embedded by RQK, so it must be
model-written, diverse, and reusable rather than a fixed field dump.

Requirements:
- Ground the memory in the actual run data, tool observations, and
environment reward/done/won/success signals.
- Write natural field-note prose, not a repeated template with the
same labels for every sample.
- Preserve a portable SOP: what to inspect first, how to choose
actions/products, what gates prove success, and what warning signs
should trigger repair.
- For WebShop, include search-query strategy, attribute/option/price
checks, product-page verification, final select behavior, and reward/
success evidence when visible.
- For ALFWorld, include reset/task-file context, observation/
admissible-command use, navigation/object/container operations, final
score/done/won evidence, and failure checks when progress stalls.
Preserve the original task verb: if the task says "look at", do not
rewrite it as a "take" task; describe any take action only as an
observed final action.
- Do not claim success unless the payload includes a verified reward/
success/won signal.
- Do not invent counterfactual mechanics.
- Do not copy raw JSON or full transcripts; distill them into
actionable memory.
- Do not merely restate the exact target ASIN/file as the skill. The
reusable part is the decision process.
- Keep it compact enough for retrieval: usually 4-8 short paragraphs
or tight bullets, 350-900 words.

Payload:
{rich_json}
```

I.3 SKILL-CLUSTER AND SID-LEAF SYNTHESIS PROMPTS

Skill-cluster synthesis

```
You are synthesizing reusable agent skill memories for RQ-KMeans
training.

Below are skill records that were assigned to the same semantic
cluster. They may mention
different repositories, file paths, brands, specific commands, or
reference files. Your task is
to extract the shared, reusable problem-solving pattern and remove
case-specific details.

Write one generalized experience record that:
1. Captures the common problem type and when to use it.
2. Preserves reusable procedures, decision rules, verification checks,
 and pitfalls.
3. Keeps important technology/domain names only when they are common
to the cluster.
4. Avoids source paths, sample ids, one-off repository names,
reference filenames, and long command dumps.
5. Is diverse and broadly reusable, not a vague label.

Return ONLY a valid JSON object with these string fields:
skill_name, problem_type, problem_addressed, applicability,
preconditions, procedure,
decision_rules, practice, lessons_learned, verification_checks,
retrieval_cues, summary

Cluster metadata:
- cluster_id: {cluster_id}
- original_records: {record_count}
- unique_descriptions: {unique_count}
- top_categories: {categories}
- top_skill_names: {skill_names}
```

SID-leaf experience merge

```
You are an expert at synthesizing problem-solving experiences. Below
are {n} experience summaries that all belong to the same semantic
cluster (SID: {sid}). They describe similar types of problems and
strategies.

Your task: Merge them into ONE comprehensive experience summary that:
1. Captures the common problem type and core strategy
2. Preserves the most useful specific techniques and decision rules
3. Includes both success patterns and failure pitfalls if mentioned
4. Is concise but complete --- no redundancy

Output ONLY the merged experience summary, no preamble.
```

I.4 EMBEDDING GENERATION PREFIX

RQ-KMeans embedding prefix

```
Instruct: Encode the following agent experience as a reusable problem-
solving memory for semantic retrieval and SID clustering. Focus on the
 problem type, reusable skill, preconditions, procedure, decision
rules, verification checks, outcome, and failure recovery. Prefer
```

```
semantic problem-solving patterns over dataset names, run ids, wording
 artifacts, or incidental entities unless they are essential to the
skill.
Query: {rich_summary}
```

I.5 StackPlanner Outcome-Reward Prompts

The StackPlanner is prompted as the LangGraph lead agent during outcome-reward rollouts. The `{memory_section}` placeholder is filled with the with-memory or the baseline block shown below.

WebShop rollout

```
Solve this WebShop product-identification task. Use web_search when
useful. Identify the single product that best satisfies every
requested attribute. Return only its 10-character ASIN (for example
B012345678), with no explanation.

{memory_section}WebShop request:
{query}

# ---- {memory_section} with-memory variant ----
Retrieved reusable memory:
{retrieved_experience}

# ---- {memory_section} baseline variant ----
No retrieved memory is provided.
```

ALFWorld rollout

```
Solve this ALFWorld task by interacting with the real TextWorld
environment. First call alfworld_reset with exactly the task_file and
split shown below. Then repeatedly choose one command exactly from the
 latest admissible_commands and call alfworld_step. Re-plan from every
 observation. Continue until a tool response returns won=true, or stop
 after 30 step calls. Do not invent internal PDDL identifiers and do
not merely print an action plan; only tool-executed environment
success counts.

{memory_section}Task request:
{task_request}

task_file: {alfworld_task_file}
split: {alfworld_split}
```

General QA (counterfactual baseline)

```
Solve the question using your normal capabilities. No retrieved memory
 is provided. Return only the final answer and no explanation.

Question:
{query}
```

I.6 MemRetriever and MemEvolver Runtime Prompts

MemRetriever system prompt

```
You are MemRetriever, a memory retrieval agent.
For every query, choose exactly one four-layer Semantic ID (SID) in
the exact form <SID_L1_X><SID_L2_Y><SID_L3_Z><SID_L4_W>, then make
exactly one memory_lookup call whose sid_list contains only that SID.
You may reason briefly in <think>...</think>. After the tool response,
 output exactly one non-empty <answer>...</answer> containing a
concise, actionable summary of the retrieved experience. Do not call
the tool again, output multiple SIDs, or copy the entire memory. If no
 relevant memory is returned, output <answer>No relevant experience
found.</answer>
```

`memory_lookup` tool description

```
type: function
function:
  name: memory_lookup
  description: >-
    Look up experience memories from the memory bank using Semantic
IDs (SIDs). Returns the top-k most relevant experience summaries for
each SID provided.
  parameters:
    type: object
    properties:
      sid_list:
        type: array
        items:
          type: string
        description: >-
          A one-element list containing the canonical four-layer SID
selected by MemRetriever. Example: ["<SID_L1_5><SID_L2_3><SID_L3_2><
SID_L4_7>"]. Do NOT use natural-language descriptions, multiple SIDs,
or partial SIDs as values.
    required:
      - sid_list
```

LLM relevance judge prompt

```
Judge whether the retrieved reusable experience is useful for the
query.
Rubric: 2 = directly useful: it contains a task-specific strategy,
evidence, constraints, or procedure that would materially help answer
the query;
1 = only broad task-type relevance: it is directionally related but
generic;
0 = unrelated, misleading, or too generic to help.
Do not reward a shared dataset label by itself. Judge the actual
content.
Return strict JSON only: {"score": 0, "reason": "short reason"}.

QUERY:
{query}

CONTENT:
{retrieved_experience}
```

MemEvolver system prompt

```
You are MemEvolver, a memory evolution agent.
Given an execution trajectory, you update the memory bank with evolved
 experience:

1. <think>: Analyse the trajectory thoroughly -- understand what task
was attempted, what knowledge domain it covers, what succeeded and
failed, and hypothesise which SID cluster captures this domain best.
Consider the trajectory's implicit lessons.

2. <tool_call>: Call memory_lookup with your chosen SID to retrieve
the existing experience.

3. <think>: Compare the trajectory's insights against the retrieved
experience -- identify what is already captured, what is new, what is
contradicted, and how to merge them. If no existing experience is
found, derive the experience entirely from the trajectory.

4. <answer>: Produce an evolved experience that synthesises the old
memory with new insights from the trajectory. The evolved experience
should be reusable: general enough to transfer to similar future tasks,
 specific enough to be actionable. Do not copy the trajectory verbatim
 -- distil the transferable lessons.
```