CORA: CAPABILITY-ORIENTED REGION AGGREGATION FOR EXPERIENTIAL MEMORY IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents increasingly accumulate rich experiential memories, yet reusing this experience under task diversity remains a central challenge. Similarity-based retrieval selects memories by topical resemblance, but its ranking signal contains no information about previous task outcomes. Utility-aware retrieval incorporates outcome-based usefulness, but a scalar utility score combines results from different task categories into one value, while estimating utility independently for each memory makes sparse feedback noisy and prevents evidence sharing across memories with similar behavior. We therefore introduce **Capability-Oriented Region Aggregation (CORA)**, which represents each memory by a category-conditioned utility profile and groups memories with similar profiles into capability-oriented regions. CORA comprises three mechanisms: *Capability Profile Clustering* discovers these regions; *Hierarchical Utility Estimation* combines individual and regional utility for category-specific reranking; and *Region-Level Knowledge Aggregation* consolidates recurring failure patterns within capability-similar regions into retrieval-time guidance. Experiments across four benchmarks and five frozen language-model backbones show that CORA achieves final-test gains of up to 6.43 percentage points over the strongest competing method, demonstrating the effectiveness of capability-oriented regions for organizing and reusing experiential memory across diverse tasks.

1 INTRODUCTION

LLM agents increasingly rely on accumulated experience—past trajectories and reflections—to improve over time (Zhao et al., 2024; Wang et al., 2023; Shinn et al., 2023; Zhang et al., 2026). As memory grows, the central challenge is no longer merely storing experience, but reusing it under task diversity. A memory's observed usefulness varies across task categories, and memories drawn from different task instances exhibit recurring patterns of strengths and weaknesses. The challenge is to identify these patterns from past outcomes and use them to organize memory for future tasks.

Similarity-based retrieval ranks memories by their resemblance to the current task, as in retrieval-augmented generation and dense passage retrieval (Lewis et al., 2020; Karpukhin et al., 2020). This criterion captures topical relevance but is agnostic to past outcomes: a memory can be similar to the current task yet repeatedly fail when applied. Similarity alone therefore cannot distinguish memories that are merely relevant from those that have demonstrated practical usefulness.

Utility-aware approaches address this limitation by incorporating usefulness or estimated value into selection (Park et al., 2023; Xiang et al., 2024; Zhang et al., 2026). However, instance-level utility scores have limitations. First, scalar scores pool outcomes across task categories, obscuring where a memory is effective or ineffective. Second, estimating utility independently for each memory makes sparse feedback noisy and prevents evidence sharing across memories with similar behavior.

These limitations point to two complementary requirements for organizing experiential memory. The representation needs to preserve a memory's usefulness across task categories, so that its different strengths and weaknesses are not collapsed into one score. Evidence also needs to be shared among memories with similar outcome patterns, so that sparse observations of one memory can be interpreted alongside evidence from related memories. Therefore, we introduce **Capability-Oriented Region Aggregation (CORA)**. CORA organizes memories according to the task categories in which they help or fail. For retrieval, it combines each memory's category-specific utility

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.

estimate with a weighted estimate pooled from related memories in the same group. The pooled estimate has greater influence when the memory has little direct feedback, while the memory's own estimate receives greater weight as direct evidence accumulates. The same groups also consolidate recurring failures into guidance for future retrieval.

CORA is realized through three mechanisms. First, *Capability Profile Clustering* uses past task outcomes to maintain each memory's utility across task categories. These values form its *capability profile*, and memories with similar profiles are grouped into *capability-oriented regions* through density-based clustering (McInnes et al., 2017). Second, for a new task, *Hierarchical Utility Estimation* combines each memory's category-specific utility with a pooled regional utility estimate to rerank semantically recalled memories. Third, *Region-Level Knowledge Aggregation* aggregates recurring failure patterns and recovery strategies from failed memories within each region into a cached summary. When retrieval selects a failure memory with an available regional summary, CORA replaces its content in the same prompt slot, giving the agent guidance drawn from multiple failures. The prompt guides the agent's response, and the task outcome updates the utility estimates.

We evaluate CORA across four benchmarks spanning embodied navigation (ALFWorld), code generation (BigCodeBench), and operating-system and database tasks (Lifelong Agent Bench–OS/DB), using four frozen language-model backbones. Across the five settings with complete E1–E10 trajectories, CORA achieves the highest mean test success rate and the highest E10 test success rate in four. Against the strongest competing method in each setting, CORA improves E10 success by 2.85 and 6.43 percentage points on ALFWorld with Qwen3.6-35B-A3B and Qwen2.5-72B-Instruct, respectively, by 1.34 points on LLB-OS, and by 1.44 points on LLB-DB. Compared with MemRL, the most direct utility-aware baseline, CORA improves the trajectory mean in all five settings by 0.43–5.36 points and improves E10 in four by 1.44–6.43 points. Ablations demonstrate the benefits of capability-space clustering, hierarchical utility estimation, and region-level failure aggregation, while region-dynamics analysis reveals differentiated capability profiles across task categories.

Our contributions are threefold:

- We introduce a capability-oriented view of experiential memory that captures differences in usefulness across task categories. It represents each memory by a capability profile and groups memories with similar profiles into capability-oriented regions.
- We develop **Capability-Oriented Region Aggregation (CORA)** with three mechanisms. *Capability Profile Clustering* forms regions from category-conditioned utility profiles (§3.3); *Hierarchical Utility Estimation* weights individual and regional utility using direct feedback (§3.4); and *Region-Level Knowledge Aggregation* summarizes regional failures and conditionally replaces retrieved failure memories with the summary (§3.5).
- We provide an empirical evaluation across four benchmarks and five frozen language-model backbones, showing that CORA achieves final-test gains of up to 6.43 percentage points over the strongest competing method.

2 Related Work

Memory and Retrieval for LLM Agents. LLM-agent memory has progressed from similarity-based access to external evidence via RAG and dense retrieval (Lewis et al., 2020; Karpukhin et al., 2020) to persistent experiential memory (Huang et al., 2026). Self-RAG (Asai et al., 2023) adds learned retrieval and self-critique, while MemGPT, Mem0, MemInsight, and MemOS (Packer et al., 2024; Chhikara et al., 2025; Salama et al., 2025; Li et al., 2025) address memory tiers, storage, augmentation, and lifecycle management. Outcome-aware methods assign heuristic importance (Park et al., 2023), estimate candidate-action value with an offline critic (Xiang et al., 2024), or learn memory-operation policies and task-adaptive relevance graphs (Zhang et al., 2025; Forouzandeh et al., 2026). Memento, Memory-R1, and MemSearcher (Zhou et al., 2025; Yan et al., 2026; Yuan et al., 2026) instead optimize case selection, rewriting, or memory operations. Most directly related, MemRL (Zhang et al., 2026) reranks recalled memories using a reward-updated scalar Q-value.

Memory Organization and Clustering. Growing memory pools motivate structure beyond flat vector indices. AriGraph, A-MEM, and LiCoMemory (Anokhin et al., 2024; Xu et al., 2025; Huang et al., 2025) provide distinct graph-based organizations: AriGraph links relational world knowledge to source observations, A-MEM dynamically links and evolves semantically related memory

notes, and LiCoMemory indexes session summaries, relational triples, and source dialogue hierarchically. Task Memory Engine (Ye, 2025) represents evolving goals and task dependencies, whereas CraniMem (Mody et al., 2026) combines goal-conditioned gating, utility tags, a bounded episodic buffer, and graph consolidation. CLAG (Roh et al., 2026) forms semantic clusters and profiles for cluster-first retrieval.

Failure-Aware Experience Aggregation. Failure-aware methods turn trajectories into reusable knowledge at different granularities. Reflexion (Shinn et al., 2023) produces feedback per trajectory; ExpeL (Zhao et al., 2024) contrasts successes and failures into global insights; and Agent Workflow Memory, MemP, and Dynamic Cheatsheet (Wang et al., 2024; Fang et al., 2025; Suzgun et al., 2025) maintain workflows, procedures, or validated strategies. ReasoningBank (Ouyang et al., 2026) distills strategies from successes and counterfactual signals from failures, while MACLA (Forouzandeh et al., 2025) contrastively refines hierarchical procedures. RecMem, Auto-Dreamer, Experiential Reflective Learning, and Feedback as a Tool (Dai et al., 2026; Ye et al., 2026; Allard et al., 2026; Gallego, 2026) further consolidate recurring interactions, cross-session experience, selectively retrieved heuristics, or critiques.

3 METHOD

3.1 PROBLEM FORMULATION

Following MemRL (Zhang et al., 2026), we adopt the Memory-Based Markov Decision Process (M-MDP) introduced by Memento (Zhou et al., 2025), defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{M} \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the transition dynamics, $\mathcal{R}$ the reward function, γ the discount factor, and $\mathcal{M}$ the memory space. At step t, the agent observes state s_t, retrieves a memory $m \sim \mu(\cdot \mid s_t, M_t)$ from memory buffer $M_t \in \mathcal{M}$, and generates an action $a_t \sim p_{\text{LLM}}(\cdot \mid s_t, m)$. Here, $a_t \in \mathcal{A}$ is a complete finite-length token sequence, while p_{LLM} is a distribution over such action sequences. Marginalizing the retrieved memory gives the overall policy:

$$\pi(a_t \mid s_t, M_t) = \sum_{m \in M_t} \mu(m \mid s_t, M_t)\, p_{\text{LLM}}(a_t \mid s_t, m), \tag{1}$$

where $\mu(m \mid s_t, M_t)\, p_{\text{LLM}}(a_t \mid s_t, m)$ is the joint likelihood of retrieving m and generating a_t, and the sum marginalizes the latent retrieval choice. MemRL structures each memory as a triplet $m_i = (z_i, e_i, Q_i)$, where z_i is the intent, e_i is the stored experience (a successful trajectory or failure reflection), and $Q_i \in \mathbb{R}$ is a scalar utility estimate. Let $\text{Emb}(\cdot)$ denote a frozen text encoder; retrieval is implemented via two-phase selection—similarity recall followed by value-aware reranking:

$$\text{score}(s_t, z_i, e_i) = (1 - \lambda)\, d(s_t, z_i) + \lambda\, Q_i, \tag{2}$$

where $d(s_t, z_i) = \cos(\text{Emb}(s_t), \text{Emb}(z_i))$ is the cosine similarity, and $\lambda \in [0, 1]$ controls the relevance–utility trade-off. After receiving reward r_t, the utility of each retrieved memory is updated via a Monte Carlo style rule:

$$Q_i \leftarrow Q_i + \alpha\, (r_t - Q_i). \tag{3}$$

3.2 CORA FRAMEWORK

We propose **Capability-Oriented Region Aggregation (CORA)**, which decomposes scalar memory utility into per-category capability profiles, clusters memories into regions by shared capability patterns, and leverages region structure for both robust utility estimation and knowledge transfer. The framework comprises three mechanisms: (i) *Capability Profile Clustering* to discover regions (§3.3); (ii) *Hierarchical Utility Estimation* to obtain smoothed category-conditioned retrieval scores (§3.4); and (iii) *Region-Level Knowledge Aggregation* to distill transferable failure knowledge within each region (§3.5). Figure 1 illustrates the overall architecture.

3.3 CAPABILITY PROFILE CLUSTERING

From scalar to vector utility. The baseline (Eq. 2) maintains a single scalar Q_i per memory, thereby collapsing category-dependent value into one number. To capture capability heterogeneity

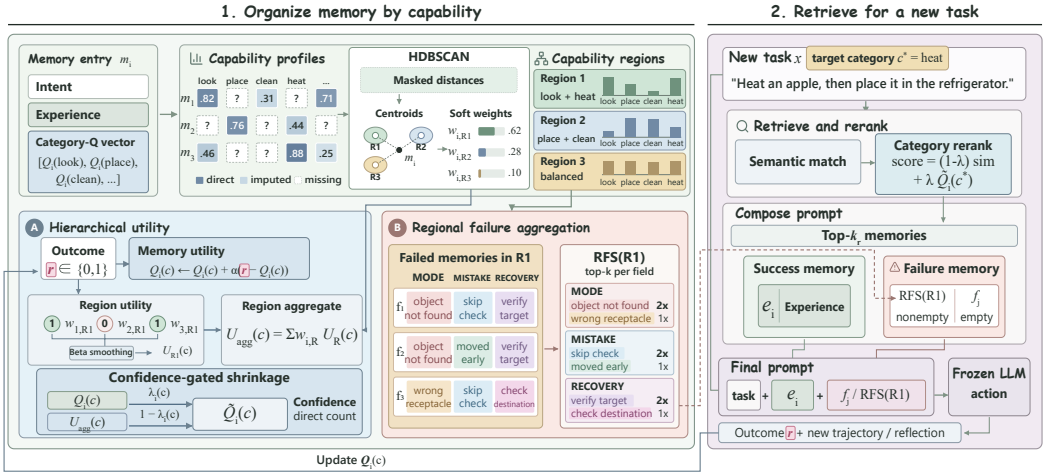


Figure 1: Overview of Capability-Oriented Region Aggregation (CORA). Memories store intents, experiences, and sparse category utilities. HDBSCAN groups masked capability profiles into regions, which cache recurring failure summaries. Hierarchical utility estimation combines memory- and region-level evidence for category-aware retrieval; task outcomes update both levels.

across categories, we generalize this representation to a per-category utility $Q_i(c)$ for each category $c \in \mathcal{C} = \{c_1, \ldots, c_K\}$. When a task of category c retrieves memory m_i and the agent receives reward $r \in \{0, 1\}$ (we drop the time subscript t for brevity in subsequent equations), we update:

$$Q_i(c) \leftarrow Q_i(c) + \alpha \cdot (r - Q_i(c)) \tag{4}$$

while the utilities for other categories remain unchanged. We call the resulting vector $\boldsymbol{q}_i = [Q_i(c_1), \ldots, Q_i(c_K)] \in [0, 1]^K$ the **capability profile** of memory m_i—it captures *what the memory is good at* and *how good it is* per category, rather than merely *how good it is* overall.

Region discovery in capability space. Memories that are textually dissimilar may nonetheless share similar patterns of strength and weakness across categories, and vice versa. To exploit these recurring capability patterns, CORA clusters memories in capability space rather than embedding space. Specifically, we construct the utility matrix $\boldsymbol{Q} \in \mathbb{R}^{|M_t| \times K}$ where row i is $\boldsymbol{q}_i$ and M_t is the memory pool at the current training step. Since most memories are only retrieved for a subset of categories, $\boldsymbol{Q}$ is typically sparse. We fill missing entries via pairwise cross-category correlation: for each unobserved $Q_i(c_j)$, we estimate it from the observed entries as

$$\hat{Q}_i(c_j) = \bar{Q}(c_j) + \frac{\sum_{c_k \in \mathcal{O}_i} |\rho_{jk}| \cdot \rho_{jk} \cdot (Q_i(c_k) - \bar{Q}(c_k))}{\sum_{c_k \in \mathcal{O}_i} |\rho_{jk}|} \tag{5}$$

where $\mathcal{O}_i$ is the set of categories for which m_i has been observed, $\bar{Q}(c)$ is the global mean of observed utilities for category c, and ρ_{jk} is the Pearson correlation estimated from reliable co-observations. Entries without a reliable predictor remain missing.

Let $\boldsymbol{q}_i^{\text{imp}}$ denote the partially imputed profile: its kth coordinate is $Q_i(c_k)$ when directly observed, $\hat{Q}_i(c_k)$ when reliably imputed, and undefined otherwise. All distances are evaluated only on coordinates defined in the relevant profiles. We apply HDBSCAN (McInnes et al., 2017) to masked pairwise distances over these profiles, thereby discovering variable-density regions without requiring their number in advance. Appendix B gives the missing-data and noise-point handling.

Soft membership. Hard assignments are brittle near region boundaries, where memories may express mixed capability signatures. To address this, CORA computes soft memberships via Euclidean distance to region centroids in capability space:

$$w_{i,\ell} = \frac{\exp(-\|\boldsymbol{q}_i^{\text{imp}} - \bar{\boldsymbol{q}}_\ell\|/T)}{\sum_{\ell'=1}^{L} \exp(-\|\boldsymbol{q}_i^{\text{imp}} - \bar{\boldsymbol{q}}_{\ell'}\|/T)} \tag{6}$$

where $\bar{\boldsymbol{q}}_\ell$ is the centroid of region ℓ and T is a temperature parameter. This defines a complete distribution over regions. Shrinkage, online credit allocation, and summary construction consume this distribution differently; their truncation and sharpening rules are specified in Appendix B.

Region capability spectrum. Each region ℓ maintains a per-category utility $U_\ell(c)$ estimated from member outcomes under soft membership. Since task rewards are binary, we use a Beta-smoothed region-level pooled estimator: for region ℓ on category c,

$$U_\ell(c) = \frac{S_\ell(c) + \alpha_0^{(\ell,c)}}{N_\ell(c) + \alpha_0^{(\ell,c)} + \beta_0^{(\ell,c)}} \tag{7}$$

where $S_\ell(c) = \sum_i w_{i,\ell} \cdot r$ is the soft-membership-weighted cumulative reward and $N_\ell(c) = \sum_i w_{i,\ell}$ is the corresponding total membership weight for retrievals on category c. The pair $(\alpha_0^{(\ell,c)}, \beta_0^{(\ell,c)})$ is a capped empirical Bayes prior warm-started from member profiles. The vector $[U_\ell(c_1), \ldots, U_\ell(c_K)]$ is the region's **capability spectrum**.

Dynamic region maintenance. Because memory collections evolve during training, CORA initializes regions after sufficient observations, assigns new memories against current centroids, and applies scheduled split/merge maintenance. Topology changes redistribute source-attributed evidence under the new memberships before utilities and summaries are rebuilt; the scheduling and conservation rules are deferred to Appendix B.

3.4 Hierarchical Utility Estimation

Confidence-gated shrinkage. With regions established, we combine the memory-level tracker with the pooled region estimate. The smoothed estimate is:

$$\tilde{Q}_i(c) = \lambda_i(c) \cdot Q_i(c) + (1 - \lambda_i(c)) \cdot U_{\text{agg}}(m_i, c) \tag{8}$$

where $U_{\text{agg}}(m_i, c) = \sum_{\ell \in \text{top-}N(m_i)} \bar{w}_{i,\ell} U_\ell(c)$ and $\bar{w}_{i,\ell}$ denotes membership renormalized within the top-N regions. The positive hyperparameter κ controls how quickly direct-evidence count increases the contribution of the individual estimate:

$$\lambda_i(c) = \lambda_{\max} \cdot \frac{n_i(c)}{n_i(c) + \kappa} \tag{9}$$

where $n_i(c)$ is the direct-evidence count, namely the number of retrieval events in which memory m_i receives an observed outcome on category c, and $\lambda_{\max} < 1$ is the maximum weight assigned to the individual estimate. With no direct feedback for a category, a memory relies on the regional estimate. As feedback accumulates, its own utility gains weight, while the cap preserves a contribution from related memories. This allows retrieval to move gradually from shared evidence toward memory-specific outcomes.

Region-aware retrieval score. The smoothed estimate directly replaces the scalar utility in the baseline retrieval score (Eq. 2). Reranking uses

$$\text{score}(s_t, z_i, e_i) = (1 - \lambda)\ \cos(\text{Emb}(s_t), \text{Emb}(z_i)) + \lambda\, \tilde{Q}_i(c_{\text{target}}), \tag{10}$$

where $\lambda \in [0, 1]$ is a tunable utility weight, with $1 - \lambda$ weighting embedding similarity.

3.5 Region-Level Knowledge Aggregation

Improving retrieval scoring through hierarchical estimation addresses *which* memories to select, but not the quality of *what* the agent sees. Individual failure memories are noisy, idiosyncratic, and bound to specific task instances—injecting them verbatim provides limited guidance. A naive alternative—aggregating failures globally across the entire memory pool or among embedding-similar memories—dilutes signal with unrelated failure modes from structurally dissimilar tasks. Region structure offers a principled middle ground: because memories within a region share similar capability weaknesses (as represented by the shared $U_\ell(c)$ in the decomposition of Appendix B.2), their failure patterns are coherent across tasks. CORA distills these correlated failures into structured region-level summaries, enriching the prompt with transferable pitfall knowledge that no single memory can provide alone.

Region failure summary. During memory generation, existing experiential memory systems (Zhang et al., 2026; Shinn et al., 2023) already extract structured fields from each failure memory m_i via LLM self-reflection: the failure mode (e.g., "Object Not Found"), specific mistakes made, and recovery strategies that could have avoided the failure. For each region ℓ, CORA aggregates these fields across all unsuccessful members via frequency counting:

$$\text{RFS}(\ell) = \text{Aggregate}\left(\{(\textsc{Mode}(m), \textsc{Mistakes}(m), \textsc{Recovery}(m)) : m \in \ell,\ \text{success}(m) = 0\}\right) \tag{11}$$

Failure modes, mistakes, and recovery strategies are each ranked by occurrence count; the top-k_s entries per field are retained and formatted into a structured summary. This requires no additional LLM calls—it operates purely on the structured fields already present in each memory. Summaries are regenerated after clustering and region maintenance operations.

Retrieval with region failure injection. At retrieval time, CORA first selects the ordinary top-k_r memories using Eq. 10. If this set naturally contains a failure memory f_i whose primary region $\ell_i^\star = \arg\max_\ell w_{i,\ell}$ has a non-empty summary, CORA conditionally replaces its content with $\text{RFS}(\ell_i^\star)$; otherwise it leaves the set unchanged. Writing ϕ for this replacement,

$$\text{Prompt} = [\text{Task}] \oplus [m_1, \ldots, m_{k_r - k_f}, \phi(f_1), \ldots, \phi(f_{k_f})]. \tag{12}$$

A configurable replacement budget k_f limits the number of naturally retrieved failures that can be replaced, supplying guidance within the fixed retrieval-slot budget and abstaining when no suitable failure is retrieved.

4 Experiments

4.1 Experimental Setup

Baselines and benchmarks. We compare CORA with a no-memory baseline, retrieval-augmented generation (RAG (Lewis et al., 2020) and Self-RAG (Asai et al., 2023)), and agentic-memory methods (Mem0 (Chhikara et al., 2025), MemP (Fang et al., 2025), and MemRL (Zhang et al., 2026)). All methods use a frozen backbone within each benchmark, so comparisons primarily reflect memory-selection quality and inference strategy. Evaluations span four agent domains: ALFWorld (Shridhar et al., 2021) for embodied navigation, BigCodeBench (Zhuo et al., 2025) for coding, and Lifelong Agent Bench–OS and –DB (Zheng et al., 2025) for operating-system and database tasks. We select backbones separately for each benchmark to avoid no-signal and ceiling regimes; the resulting backbone assignments are shown in Table 1. Semantic retrieval uses `Qwen3-Embedding-8B` with open-weight backbones and `text-embedding-3-large` with proprietary backbones.

Protocol and metrics. Iterative-memory methods run for ten epochs. Memory accumulates across epochs, while utilities are updated after each task outcome. Following the MemRL protocol, all methods use the same train/test partition, and Table 1 reports only test success rate (SR). E10 SR is the primary endpoint. Mean E1–E10 SR is trajectory-level supporting evidence of effectiveness across evolving memory states, while peak SR is a descriptive best-observed checkpoint and is not used for checkpoint selection. GPT-5-mini reports only peak test SR following the available MemRL protocol and is therefore excluded from trajectory-level comparisons. Generation uses temperature $T = 0$ throughout.

4.2 Main Results

Overall performance. Table 1 shows that CORA achieves the highest observed trajectory-mean test SR in all five settings with complete E1–E10 results and the highest final E10 test SR in four. At E10, CORA improves over the best competing method by 2.85 and 6.43 percentage points on ALFWorld with Qwen3.6-35B-A3B and Qwen2.5-72B-Instruct, respectively, by 1.34 points on LLB-OS, and by 1.44 points on LLB-DB. On BigCodeBench, CORA's E10 test SR is 0.58 points lower than MemRL (49.71 vs. 50.29), but its E1–E10 mean is 1.25 points higher (51.75 vs. 50.50). This small endpoint reversal coincides with late-stage region expansion as the memory pool evolves; we examine this late-stage performance variation through the region-dynamics analysis below. Taken together, these results demonstrate CORA's robust overall advantage across diverse tasks, model backbones, and evolving memory states.

Table 1: Main test results. Each trajectory cell reports E1–E10 mean / E10 / peak SR. GPT-5-mini reports only peak test SR following MemRL. The best result for each metric is bolded.

(a) ALFWorld

Method	*Qwen3.6-35B-A3B*	*Qwen2.5-72B-Instruct*	*GPT-5-mini*
No Memory	36.43 / 36.43 / 36.43	43.57 / 43.57 / 43.57	83.6
RAG	58.57 / 62.86 / 62.86	64.00 / 57.14 / 67.14	95.0
Self-RAG	58.25 / 54.29 / 61.43	63.71 / 63.57 / 64.29	95.0
Mem0	42.70 / 55.71 / 55.71	61.43 / 68.57 / 68.57	95.0
MemP	39.37 / 39.29 / 44.29	50.56 / 51.43 / 51.43	92.1
MemRL	73.49 / 74.29 / 74.29	74.71 / 76.43 / 77.14	**97.9**
CORA	**75.86 / 77.14 / 80.00**	**80.07 / 82.86 / 82.86**	**97.9**

(b) BigCodeBench and Lifelong Agent Bench

Method	BigCodeBench *DeepSeek-V3*	LLB-OS *GPT-4.1-mini*	LLB-DB *GPT-4.1-mini*
No Memory	46.49 / 46.49 / 46.49	72.67 / 72.67 / 72.67	77.70 / 77.70 / 77.70
RAG	47.89 / 45.91 / 49.71	75.60 / 75.33 / 78.00	78.35 / 77.70 / 79.86
Self-RAG	46.61 / 43.86 / 50.58	71.20 / 68.67 / 72.67	77.27 / 77.70 / 79.14
Mem0	51.46 / 48.54 / 52.92	73.60 / 70.67 / 76.00	77.92 / 76.98 / 79.86
MemP	48.63 / 48.54 / 49.12	69.13 / 69.33 / 73.33	76.91 / 76.98 / 78.42
MemRL	50.50 / **50.29** / 52.34	74.40 / 73.33 / 78.00	78.85 / 78.42 / 79.86
CORA	**51.75** / 49.71 / **54.09**	**76.40 / 76.67 / 78.67**	**79.28 / 79.86 / 80.58**

Comparison with MemRL. MemRL is the most direct baseline because both methods combine semantic recall with utility-based reranking; CORA extends this approach with category-conditioned capability profiles and region-level evidence sharing. Across the five complete settings, CORA improves the trajectory mean over MemRL by 0.43–5.36 points and improves E10 in four of five settings by 1.44–6.43 points. Together with the broader baseline comparison, these results support capability-oriented organization beyond scalar instance-level utility. In the near-saturated GPT-5-mini setting, both methods reach 97.9%, indicating parity rather than further improvement.

Region dynamics. Figure 2 provides a descriptive view of how the capability-space organization evolves and of the task-specific profiles that result; it is a mechanism analysis rather than a standalone effectiveness test. Each memory is observed only a few times for most task categories, making its individual estimate unreliable. CORA reduces this uncertainty by pooling observations from memories with similar capability profiles within the same region, while keeping different regions separate. At the final checkpoint, direct coverage is only 16.5%–22.8% across the four main runs, while their region profiles remain differentiated across task categories. On ALFWorld, for example, R4 is highly useful for `look` but weak for `clean`, whereas R2 is strongest on the `two`-object category. BigCodeBench exhibits the same rank-reversal structure: different regions lead computation (R17), cryptography (R5), general (R4), network (R2), system (R14), time (R17), and visualization (R6). No region is uniformly strongest across categories.

The region-count traces show different evolution patterns as memory accumulates. ALFWorld and LLB-OS settle after early growth. In contrast, BigCodeBench expands from 10 to 18 regions over the final two epochs as test SR decreases from 53.22% to 49.71%; this joint pattern provides a descriptive, mechanism-level account of the late-stage performance variation accompanying continued topology refinement. Its region utilities vary most across computation, general, and system tasks, whereas network utility is uniformly compressed (0.35–0.56), exposing a benchmark-wide weakness rather than a specialized region. The final scale plots show that these profiles are supported by groups of memories rather than isolated examples.

4.3 Ablation Studies

We evaluate the three mechanisms of CORA on ALFWorld and BigCodeBench. ALFWorld uses Qwen2.5-72B-Instruct and the training, in-distribution (ID), and out-of-distribution (OOD) splits. BigCodeBench uses the DeepSeek-V3 model, ten epochs, and the final validation success rate.

4.3.1 Hierarchical Utility Estimation

The shrinkage-ceiling panel in Figure 3 varies the ceiling on ALFWorld with capability-space regions and adaptive RFS held fixed. Complete shrinkage ($\lambda_{\max} = 0$) reaches 76.4% ID and 82.1%

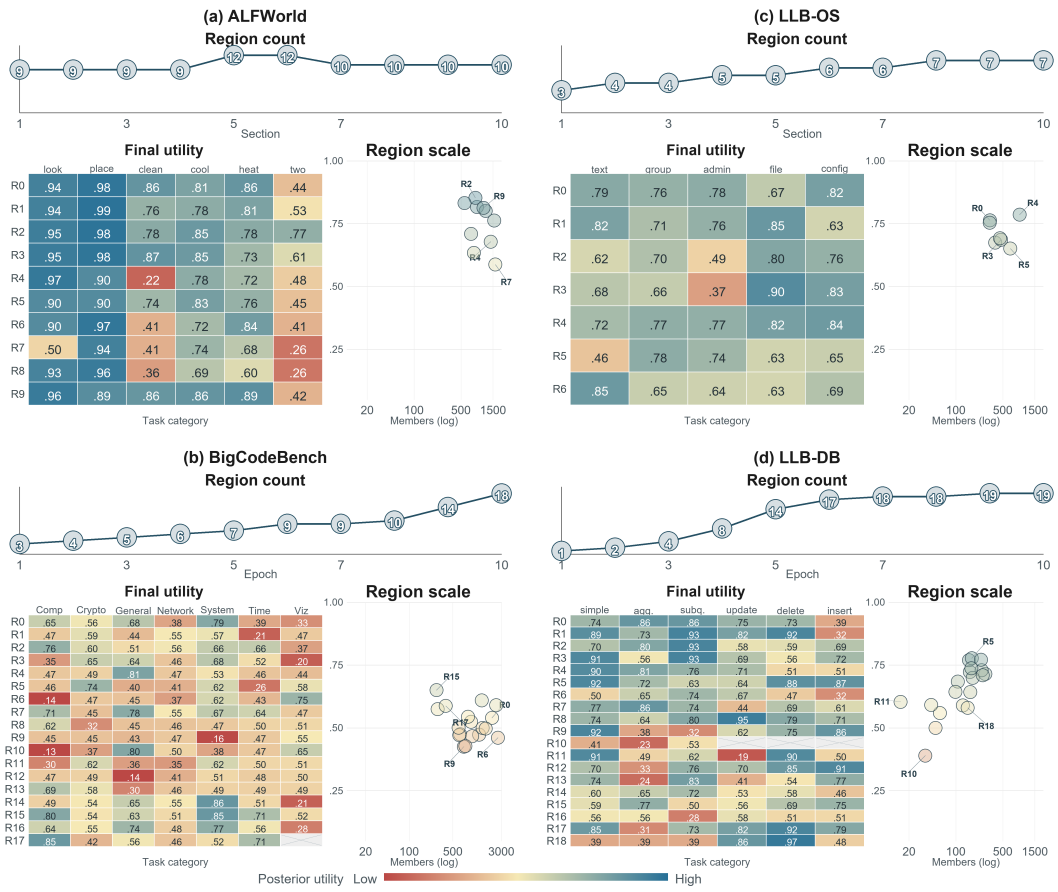


Figure 2: Region dynamics. Regions have different strengths and weaknesses across task categories. Top curves track the number of clustered regions over ten epochs. Heatmaps show the final posterior Q-value of each region for each task category; R0, R1, etc. denote region IDs. Scatterplots show region size versus mean Q-value.

OOD at E10. Allowing a moderate amount of direct per-memory evidence ($\lambda_{\max} = 0.15$) improves both scores to 80.7% and 85.1%, respectively, and also gives the highest final training SR (79.4%). The intermediate ceiling ($\lambda_{\max} = 0.45$) reaches 80.0% ID and 88.1% OOD, the highest OOD score in the sweep; relative to $\lambda_{\max} = 0.15$, it improves OOD by 3.0pp but gives up 0.7pp on ID and 1.5pp on final training SR. The largest ceiling ($\lambda_{\max} = 0.75$) reaches 77.9% ID and 85.1% OOD, providing no additional OOD gain. These endpoints suggest a favorable OOD–ID/training trade-off at an intermediate ceiling. On BigCodeBench, full CORA reaches 49.71% final validation SR, while the Category-Q-only variant reaches 47.37%. This comparison shows that raw category-conditioned Q alone is insufficient.

4.3.2 Capability Profile Clustering

We next hold adaptive RFS and the shrinkage estimator fixed, changing only the space in which regions are formed. Capability space gives 77.6% final training SR and 82.9%/86.6% ID/OOD SR, compared with 77.3% and 82.1%/85.8% for embedding space (the region-space panels in Figure 3). Capability space stabilizes at ten regions with E10 Gini 0.187 and maximum share 15.4%, and produces differentiated region-Q profiles. Here, Gini measures how concentrated membership is across regions, and maximum share is the fraction of all members in the largest region; higher values on either measure indicate stronger dominance by one region. Embedding space ends with fifteen declared regions but 100% of membership in a single active region, with Gini 0.933 and maximum share 100%. That dominant region retains a task-wise Q span of 0.53–0.82, but it cannot provide

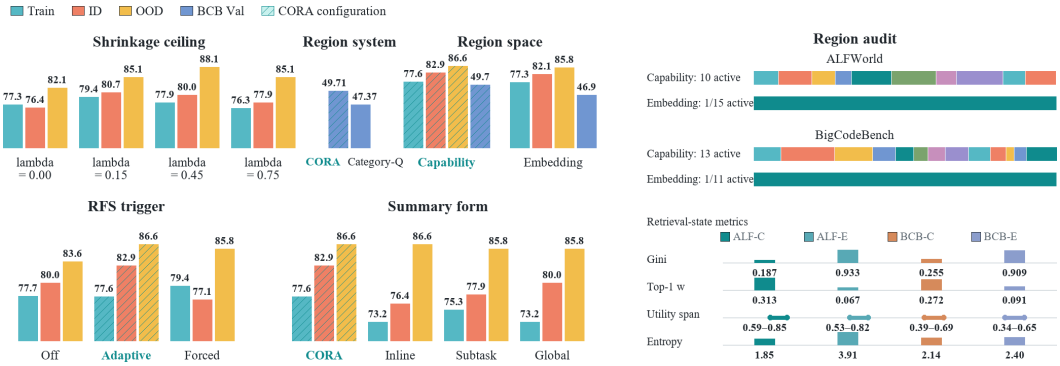


Figure 3: Ablation summary. We vary the shrinkage ceiling, region aggregation, region space, RFS trigger, and summary form; the audit shows region allocations and retrieval metrics. Bars report ALFWorld Train/ID/OOD and BigCodeBench validation SR; diagonal hatching marks CORA.

multiple capability-specific priors. On BigCodeBench, CORA reaches 49.71% final validation SR, compared with 46.9% for embedding space. The accompanying BCB utility profiles show thirteen active capability-space regions (Gini 0.255) versus one active region of eleven declared under embedding space (Gini 0.909).

4.3.3 Region-Level Knowledge Aggregation

With capability-space regions and shrinkage fixed, adaptive RFS improves ALFWorld from 80.0%/83.6% to 82.9%/86.6% ID/OOD over disabling summaries. Forcing a summary into every retrieval instead lowers ID SR to 77.1% and OOD SR to 85.8%, showing that the conditional trigger is important: summaries are useful when a relevant failure is naturally retrieved, not as a universal prompt prefix.

Summary representation. We compare three ways of presenting retrieved failure knowledge. Inline aggregation applies the same summarization procedure to the top-k failure memories (with $k = 10$) and injects the resulting failure summary into the retrieval context. Subtask-conditioned summaries first group failures by the current task or subtask, so the injected summary is matched to that query. Global aggregation pools failure memories across regions into a single summary, without region-specific conditioning. Under the same capability-region configuration, inline reaches 76.4% ID and 86.6% OOD SR, subtask-conditioned summaries reach 77.9% and 85.8%, and the global summary reaches 80.0% and 85.8%, respectively. CORA's region summary gives the best overall performance, reaching 82.9% ID and 86.6% OOD SR. Because each region groups memories with similar capability weaknesses, its summary aggregates coherent failure patterns while excluding unrelated failure modes, providing more transferable guidance.

5 Conclusion

Reusing experiential memory across tasks requires accounting for differences in usefulness across task categories and sharing evidence when feedback on individual memories is sparse. We introduced Capability-Oriented Region Aggregation (CORA) to address these challenges. CORA groups memories by their category-specific capability profiles, combines individual and regional utility for retrieval, and conditionally replaces a retrieved failure memory with guidance aggregated from related failures. Experiments across four benchmarks and multiple frozen language-model backbones show that CORA generally outperforms memory baselines. Component ablations support the contributions of its three mechanisms, while region analysis reveals distinct strengths and weaknesses across task categories. Together, these findings show the value of capability-oriented regions for organizing and reusing experiential memory. Future work could use finer-grained credit assignment to distinguish which subgoals or decisions made a memory useful, and uncertainty-aware capability profiles to form more reliable regions when feedback remains sparse.

POLICY ON LLM USE FOR RESEARCH AND WRITING PAPERS

In this work, we used generative AI tools to assist with refining the conceptual model framework, revising the research methodology, modifying and debugging method and experimental code, and interpreting experimental results. We also used generative AI tools to assist with identifying a portion of the related work and with drafting and editing portions of the manuscript, including improving readability, suggesting keywords, and refining presentation.

We did not use generative AI tools to generate synthetic datasets; formulate mathematical claims, theorems, propositions, or proofs; propose or refine research hypotheses; translate text; or clean, reformat, label, or conduct qualitative or thematic analysis of datasets.

All AI-assisted suggestions, code modifications, and result interpretations were reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including its methodology, implementation, experiments, empirical claims, and all text or artifacts produced with the aid of generative AI.

REFERENCES

Marc-Antoine Allard, Arnaud Teinturier, Victor Xing, and Gautier Viaud. Experiential reflective learning for self-improving llm agents, 2026. URL https://arxiv.org/abs/2603.24639.

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*, 2024.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection, 2023. URL https://arxiv.org/abs/2310.11511.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Zijie Dai, Shiyuan Deng, Sheng Guan, Yizhou Tian, Xin Yao, Xiao Yan, and James Cheng. Recmem: Recurrence-based memory consolidation for efficient and effective long-running llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32353–32376, 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. URL https://arxiv.org/abs/2508.06433.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for llm agents through bayesian selection and contrastive refinement. *arXiv preprint arXiv:2512.18950*, 2025.

Saman Forouzandeh, Kamal Berahmand, and Mahdi Jalili. Task-adaptive retrieval over agentic multi-modal web histories via learned graph memory. *arXiv preprint arXiv:2604.07863*, 2026.

Víctor Gallego. Distilling feedback into memory-as-a-tool, 2026. URL https://arxiv.org/abs/2601.05960.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. A survey of agent memory in the second half: Towards self-evolving and long-horizon agents, 2026. URL https://arxiv.org/abs/2602.06052.

Zhengjun Huang, Zhoujin Tian, Qintian Guo, Fangyuan Zhang, Yingli Zhou, Di Jiang, and Xiaofang Zhou. Licomemory: Lightweight and cognitive agentic memory for efficient long-term reasoning. *arXiv preprint arXiv:2511.01448*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pp. 6769–6781, 2020.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, et al. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*, 2025.

Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017. doi: 10.21105/joss.00205.

Pearl Mody, Mihir Panchal, Rishit Kar, Kiran Bhowmick, and Ruhina Karani. Cranimem: Cranial inspired gated and bounded memory for agentic systems, 2026. URL https://arxiv.org/abs/2603.15642.

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

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

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Taeyun Roh, Wonjune Jang, Junha Jung, and Jaewoo Kang. Clag: Adaptive memory organization via agent-driven clustering for small language model agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16707–16726, 2026.

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. Meminsight: Autonomous memory augmentation for llm agents. *arXiv preprint arXiv:2503.21760*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. URL https://arxiv.org/abs/2010.03768.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

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.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Yufei Xiang, Yiqun Shen, Yeqin Zhang, and Nguyen Cam-Tu. Retrospex: Language agent meets offline reinforcement learning critic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4650–4666, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

Chongrui Ye, Yuxiang Liu, Yu Wang, Haofei Yu, Yining Zhao, Ge Liu, Julian McAuley, and Jiaxuan You. Auto-dreamer: Learning offline memory consolidation for language agents, 2026. URL https://arxiv.org/abs/2605.20616.

Ye Ye. Task memory engine: Spatial memory for robust multi-step llm agents. *arXiv preprint arXiv:2505.19436*, 2025.

Qianhao Yuan, Jie Lou, Zichao Li, Jiawei Chen, Yaojie Lu, Hongyu Lin, Le Sun, Debing Zhang, and Xianpei Han. MemSearcher: Iterative memory integration for search agent via end-to-end reinforcement learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14965–14977, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.736. URL https://aclanthology.org/2026.findings-acl.736/.

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, 2026. URL https://arxiv.org/abs/2601.03192.

Zeyu Zhang, Quanyu Dai, Rui Li, Xiaohe Bo, Xu Chen, and Zhenhua Dong. Learn to memorize: Optimizing llm-based agents with adaptive memory framework. *arXiv preprint arXiv:2508.16629*, 2025.

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

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners, 2025. URL https://arxiv.org/abs/2505.11942.

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

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions, 2025. URL https://arxiv.org/abs/2406.15877.

A Parameter and Model Selection

A.1 Benchmark Hyperparameters

Table 2 records the configurations of the main experiments. The per-category learning rate α controls how quickly a memory's CORA utility follows a binary reward, whereas α_{scalar} is MemRL's scalar-Q rate. The cluster-size floor $s_{\min}$ and temperature T determine region granularity and membership uncertainty; N, C, and κ determine the strength of regional smoothing. The train/evaluation entries for $\lambda_{\max}$ distinguish the runtime evaluation override from the training value.

Parameter (symbol)	Role	ALFWorld	LLB-DB	LLB-OS	BCB
CORA-specific configuration					
Per-category utility update (α)	Direct binary-reward update rate	0.10	0.10	0.10	0.10
Region utility estimator	Region–category utility model	Beta	Beta	Beta	Beta
Minimum cluster size ($s_{\min}$)	HDBSCAN minimum cluster size	15	15	15	12
Minimum samples (s_{samp})	HDBSCAN core-point setting; 0 uses $s_{\min}/2$	0	0	0	0
Cluster selection	HDBSCAN extraction rule	EOM	EOM	EOM	leaf
Membership temperature (T)	Softmax temperature for region memberships	0.10	0.10	0.10	0.10
Shrinkage region count (N)	Top memberships used by shrinkage	1	1	1	1
Smoothing strength (C)	Cap on warm-start effective evidence	0.50	0.50	0.50	0.50
Evidence sharpening (p_e)	Exponent for online soft-credit allocation	2.0	2.0	2.0	2.0
Propagation ($\eta, k_n, \delta_{\min}$)	Rate, neighbors, and minimum cosine similarity	0.03, 30, 0.40	0.12, 30, 0.40	0.03, 30, 0.40	0.12, 30, 0.40
Shrinkage confidence (κ)	Evidence scale for overriding U_{agg}	2.50	3.00	2.50	3.00
Maximum individual weight ($\lambda_{\max}$)	Direct-Q weight (train/evaluation)	0.80 / 0.45	0.80 / 0.80	0.80 / 0.80	0.80 / 0.15
Region gating / Q normalization	Retrieval gate and similarity/Q normalization	additive / off	additive / off	additive / off	additive / on
Exploration schedule	Replacement slots in epochs 1–10	0, 1, 1, 1, 0, 0, 0, 0, 0, 0	0, 4, 3, 2, 2, 1, 1, 1, 1, 0	0, 1, 1, 1, 0, 0, 0, 0, 0, 0	0, 4, 3, 2, 2, 1, 1, 1, 1, 0
Failure-summary policy	Maximum slots	1	2	1	1
MemRL shared configuration					
Memory representation	Stored experience format	trajectory	proceduralization	proceduralization	proceduralization
Recall budget (K_{recall})	Similarity-retrieved memories before reranking	3	10	10	10
ϵ	Epsilon-greedy exploration probability	0	0.01	0.01	0.10
τ	Unknown-detection similarity threshold	0.60	0.35	0.60	0.18
Scalar-Q update (α_{scalar})	MemRL scalar utility learning rate	0.30	0.30	0.30	0.30
Discount (γ)	Scalar-Q discount factor	0	0	0	0
Initial scalar Q (q_0^+, q_0^-)	Positive and negative Q initialization	0, 0	0, 0	0, 0	0, 0
Scalar-Q floor (q_{floor})	Optional lower bound on scalar Q	none	none	none	none
Rewards (r_+, r_-)	Scalar-Q success and failure rewards	1, -1	1, 0	1, 0	1, -1
Similarity threshold	Retrieval filtering threshold	0.50	0.369	0.50	0.18
Reranking candidates (K_{RL})	Value-aware candidate set size	3	5	5	5
Novelty threshold	Similarity threshold for memory merging	0.85	0.85	0.85	0.85
Recency boost	Recency weight in scalar prioritization	0	0	0	0
Reward merge gain	Success-attribution gain for close memories	0.10	0.10	0.10	0.10
Minimum scalar Q	Lower selection threshold	-10	-0.8	-0.8	-0.8
Score weights (w_{sim}, w_Q)	Similarity and scalar/hierarchical-Q weights	0.45, 0.55	0.50, 0.50	0.45, 0.55	0.50, 0.50

Table 2: Hyperparameters for CORA. The upper block is specific to CORA; the lower block is shared with MemRL.

A.2 Model and Dataset Configuration

The backbone, model version, numerical precision, and tensor-parallel configuration used for each benchmark setting are listed in Table 3. Qwen3.6-35B-A3B uses BF16 precision with TP=1, Qwen2.5-72B-Instruct uses BF16 precision with TP=2, and DeepSeek-V3 uses FP8 precision with TP=8.

Table 4 summarizes the benchmark instances and data partitions used in our experiments.

B Implementation Details

B.1 Category Routing

Categories are obtained from benchmark task metadata. ALFWorld uses seven annotated task types: `look_at_obj_in_light`, `pick_and_place_simple`, `pick_and_place_with_movable_recep`, `pick_clean_then_place_in_recep`, `pick_cool_then_place_in_recep`, `pick_heat_then_place_in_recep`, and

Benchmark	Backbone	Version / weight configuration
ALFWorld	Qwen3.6-35B-A3B	Open-source weights; BF16, TP=1
ALFWorld	Qwen2.5-72B-Instruct	Open-source weights; BF16, TP=2
ALFWorld	GPT-5-mini	`gpt-5-mini-2025-08-07`
BigCodeBench	DeepSeek-V3	DeepSeek-V3; FP8, TP=8
Lifelong Agent (OS)	GPT-4.1-mini	`gpt-4.1-mini`
Lifelong Agent (DB)	GPT-4.1-mini	`gpt-4.1-mini`

Table 3: Backbones, model versions, numerical precision, and tensor-parallel configurations used for each benchmark setting.

Benchmark	Data	Partition
Lifelong Agent (OS)	500 tasks total	7:3 train/test split (seed 42)
Lifelong Agent (DB)	500 tasks total	7:3 train/test split (seed 42)
BigCodeBench	`Instruct (Full)`; 1,140 tasks	7:3 train/test split (seed 42)
ALFWorld	3,553 training tasks	Official training split
ALFWorld (main evaluation)	140 tasks	Official in-distribution evaluation split
ALFWorld (OOD evaluation)	134 tasks	Official out-of-distribution evaluation split

Table 4: Benchmark datasets and partitions used in the experiments.

`pick_two_obj_and_place`. For BigCodeBench, the first domain label in each task metadata record determines one of seven domains: computation, system, general, cryptography, network, time, or visualization. For LLB-OS, command keywords are mapped to user administration, user groups, text processing, file operations, or configuration setup. For LLB-DB, tasks are assigned to insertion, update, deletion, simple selection, aggregate selection, or subquery selection according to their operation and query structure. When several rules match, a fixed priority order resolves the ambiguity.

Open-weight backbone runs (ALFWorld with Qwen and BigCodeBench with DeepSeek-V3) use `Qwen3-Embedding-8B` as the frozen text encoder for semantic recall and similarity scoring. Proprietary backbone runs (ALFWorld with GPT-5-mini and LLB-OS/DB with GPT-4.1-mini) use `text-embedding-3-large`.

B.2 Capability-Region Decomposition

Estimating each capability profile $\boldsymbol{q}_i$ independently is data-inefficient and ignores shared structure across memories. We hypothesize that memories with similar capability patterns form latent groups—*regions*—whose members share a common category-conditioned component. Let $Y_e \in \{0, 1\}$ be the outcome of retrieval event e, with source memory $i(e)$ and category $c(e)$, and define the latent success rate

$$p_i(c) := \Pr(Y_e = 1 \mid i(e) = i,\, c(e) = c)\,. \tag{13}$$

For a latent region assignment $\ell_i \in \{1, \ldots, L\}$, we decompose this rate as

$$p_i(c) = \mu_0 + \theta_{\ell_i, c} + u_{i,c}, \tag{14}$$

where μ_0 is the global mean utility, $\theta_{\ell,c}$ is the region–category interaction, and $u_{i,c}$ captures memory-specific specialization within a region. The components are constrained so that $p_i(c) \in [0, 1]$. The observed binary outcome can equivalently be written as

$$Y_e = p_{i(e)}(c(e)) + \varepsilon_e, \qquad \mathbb{E}[\varepsilon_e \mid i(e), c(e)] = 0, \tag{15}$$

where ε_e captures event-level variation around the latent success rate. Absorbing the global component into $U_\ell(c) := \mu_0 + \theta_{\ell,c}$ yields $p_i(c) = U_{\ell_i}(c) + u_{i,c}$.

B.3 Region Formation, Membership, Evidence, and Dynamic Topology

Let $\boldsymbol{Q} = [Q_i(c_k)]$ denote the sparse utility matrix, with unobserved entries left undefined, and let $\boldsymbol{q}_i^{\mathrm{imp}}$ denote the partially imputed profile defined in the main text. Correlation-based imputation is applied only when a category pair has at least ten co-observations, non-negligible empirical variation, and a finite Pearson correlation; a prediction additionally requires an absolute correlation of at least 0.05. Coordinates without a reliable predictor remain undefined. For memories m_i and m_j, HDBSCAN receives the precomputed masked distance

$$d_{ij} = \left(\frac{\sum_{k \in S_{ij}} \omega_k \left[Q_i^{\mathrm{imp}}(c_k) - Q_j^{\mathrm{imp}}(c_k) \right]^2}{\sum_{k \in S_{ij}} \omega_k} \right)^{1/2}, \qquad S_{ij} = \left\{ k \in \{1, \dots, K\} : \begin{smallmatrix} Q_i^{\mathrm{imp}}(c_k) \text{ and} \\ Q_j^{\mathrm{imp}}(c_k) \text{ are defined} \end{smallmatrix} \right\}. \tag{16}$$

Thus, both directly observed and reliably imputed coordinates contribute to clustering, while undefined coordinates are masked out. The default configuration sets $\omega_k = 1$ for every category; an optional variant sets the weights proportional to the empirical variance of each category across its available memories. If $S_{ij} = \varnothing$, we set $d_{ij} = 1.0$, the maximum possible distance for utilities in $[0, 1]$.

We apply HDBSCAN with Eq. 16 as a precomputed metric. If it produces at least one non-noise cluster, noise memories are processed in order and assigned the label of their nearest currently non-noise memory under the same distance matrix. If HDBSCAN produces no non-noise cluster, all memories form a single fallback region.

For each region identified by an HDBSCAN cluster label (after this noise reassignment), its centroid is computed coordinate-wise from the original matrix $\boldsymbol{Q}$: only directly observed $Q_i(c)$ values of members with that HDBSCAN label are averaged, and imputed values are excluded. A category with no direct observation in such a region is stored as 0.0 in that centroid. This prevents correlation-based predictions from being reused as region-level direct evidence. Soft memberships are then computed from $\boldsymbol{q}_i^{\mathrm{imp}}$ to these fixed centroids, ignoring only coordinates that remain undefined; consequently, an imputed memory coordinate is compared against the stored 0.0 when the corresponding centroid coordinate has no direct observation. A softmax over these distances yields the complete distribution of region membership weights. Finally, the maximum-weight region provides each memory's canonical hard membership for downstream routing and summary construction; the centroids are not recomputed after this hard-membership rebuild.

The full soft membership is consumed differently by three downstream operations. Shrinkage selects the top-N memberships and renormalizes them without power sharpening; the main ALFWorld configuration uses $N = 1$. Online region credit instead selects the top three memberships and sharpens them with exponent 2.0 before distributing the binary outcome. Region failure summaries use the argmax membership as a hard primary region. These operation-specific transformations do not redefine the underlying soft membership.

Each region–category cell maintains a Beta-smoothed pooled estimate initialized from the member capability profiles with a capped pseudo-count mass. The initialization parameters $\alpha_0^{(\ell,c)}$ and $\beta_0^{(\ell,c)}$ contribute total mass $\alpha_0^{(\ell,c)} + \beta_0^{(\ell,c)}$, which controls the strength of the warm start and does not represent additional observed outcomes. Subsequent observations contribute membership-weighted binary success and direct-evidence mass. Conditional on the current memberships and initialization, the update is the Beta posterior mean for that region–category cell. Outcomes collected before the first clustering update each category's $Q_i(c)$ and therefore affect clustering and initialization, but they are not replayed as post-cluster observations. For a target category already observed anywhere in the system, an unobserved cell in a particular region returns the neutral value 0.5. Cross-category zero-shot transfer is activated only when the target category is globally unseen; it uses embedding-weighted residual transfer when category-description embeddings are available and otherwise falls back to the configured guarded/generalization estimator.

The confidence gate uses only the direct-evidence count $n_i(c)$. When $n_i(c) = 0$ and $Q_i(c)$ remains at its neutral initialization of 0.5, the implementation uses the pooled region estimate alone, as Eq. 9 prescribes. If $n_i(c) = 0$ but initialization or embedding-neighbor propagation has moved $Q_i(c)$

away from 0.5, the cold-start branch retains a small non-direct contribution:

$$\tilde{Q}_i(c) = 0.1Q_i(c) + 0.9U_{\text{agg}}(m_i, c). \tag{17}$$

This branch does not increment $n_i(c)$ and therefore cannot make inferred evidence pass as direct confidence. Before region initialization, or when a memory has no membership, $U_{\text{agg}}(m_i, c)$ falls back to 0.5.

Direct outcomes are retained in a source-attributed evidence ledger. For each evidence event e on category c, let $i(e)$ be its source memory and $r_e \in \{0, 1\}$ its outcome. Given the current membership weights $w_{i,\ell}$ from Eq. 6, the region–category statistics are reconstructed as

$$S_\ell(c) = \sum_{e \in \mathcal{E}_c} w_{i(e),\ell}\, r_e, \qquad N_\ell(c) = \sum_{e \in \mathcal{E}_c} w_{i(e),\ell}, \tag{18}$$

where $\mathcal{E}_c$ is the set of direct ledger events for category c.

B.3.1 Evidence Conservation Across Topology Updates

Proposition 1 (Direct-evidence conservation)**.** *For a category c, let $S_\ell(c)$ and $N_\ell(c)$ be reconstructed from the source-attributed ledger by Eq. 18. If the current memberships form a complete distribution, then*

$$\sum_{\ell=1}^{L} S_\ell(c) = \sum_{e \in \mathcal{E}_c} r_e, \qquad \sum_{\ell=1}^{L} N_\ell(c) = |\mathcal{E}_c|. \tag{19}$$

Consequently, a split or merge that recomputes memberships and then reconstructs these statistics conserves total direct success mass and total direct evidence mass for c.

Proof. Because the softmax memberships in Eq. 6 sum to one, $\sum_{\ell=1}^{L} w_{i(e),\ell} = 1$. Summing Eq. 18 over regions and exchanging the finite sums therefore gives

$$\sum_{\ell=1}^{L} S_\ell(c) = \sum_{e \in \mathcal{E}_c} r_e \sum_{\ell=1}^{L} w_{i(e),\ell} = \sum_{e \in \mathcal{E}_c} r_e.$$

The same argument without r_e proves the statement for $N_\ell(c)$. □

Remark 1 (What is conserved)**.** *Proposition 1 applies to direct ledger evidence only. It does not state that region smoothed means $U_\ell(c)$, capped warm-start pseudo-counts, hard memberships, or individual propagated values remain unchanged after a topology edit.*

After initial region discovery, new memories are lazily assigned using the current centroids. Scheduled topology maintenance splits a sufficiently large, high-variance region with two-way KMeans and merges the closest pair whose mean absolute utility difference is below a threshold relative to the global utility range. After a split or merge, CORA recomputes centroids and all soft memberships, migrates evidence, and rebuilds region summaries. The same source events are then evaluated with the new membership weights, and the resulting $S_\ell(c)$ and $N_\ell(c)$ are used in Eq. 7; thus the evidence is redistributed while preserving its source attribution. Split child regions inherit the affected parent's capped initialization pseudo-counts, while a merged region forms new capped initialization pseudo-counts from the merged member profiles.

B.4 Embedding-Space Q Propagation and Training-Time Exploration

Embedding-space Q propagation is an auxiliary implementation detail rather than a core CORA mechanism. When memory m_i is retrieved on category c and receives reward r, the implementation can update each of its top-k_n embedding neighbors m_j whose cosine similarity exceeds $\delta_{\min}$:

$$Q_j(c) \leftarrow Q_j(c) + \eta\, \zeta_{i,c} \cdot \left(\frac{\cos(\boldsymbol{e}(m_i), \boldsymbol{e}(m_j)) - \delta_{\min}}{1 - \delta_{\min}} \right)^{\xi} \cdot (r - Q_j(c)) \tag{20}$$

Here, η is the propagation learning rate, $\delta_{\min}$ is the minimum similarity threshold, and $\xi > 1$ concentrates propagation on close neighbors. The source evidence mass $\zeta_{i,c}$ is the normalized weight assigned to category c by the task that directly retrieved m_i:

$$\zeta_{i,c} = \begin{cases} 1/|\mathcal{C}(x)|, & c \in \mathcal{C}(x), \\ 0, & c \notin \mathcal{C}(x), \end{cases} \qquad \sum_{c \in \mathcal{C}(x)} \zeta_{i,c} = 1, \tag{21}$$

where x is the task that produced reward r and $\mathcal{C}(x)$ is its set of annotated capability categories. Thus, the single-label setting used in the main experiments has $\zeta_{i,c} = 1$ for its sole category; a multi-label task distributes one unit of direct evidence equally across its labels. This evidence mass is distinct from the region soft membership $w_{i,\ell}$ in Eq. 6. Propagated updates do *not* increment $n_j(c)$, preserving the distinction between direct and inferred evidence. The neighbor update in Eq. 20 is one-hop and is additionally multiplied by the source evidence mass $\zeta_{i,c}$. The main configuration uses quadratic similarity decay ($\xi = 2$). Propagated updates do not increment direct observation counts, so inferred evidence cannot increase the confidence gate.

Training-time exploration is implemented as replacement after the ordinary composite ranking, rather than as epsilon-greedy retrieval or random sampling within a region. After excluding already selected memories, CORA ranks the remaining candidates by a UCB-style uncertainty bonus computed from their total direct observation counts. Successful and failed candidates are considered separately, the exploration budget is divided between them according to a fixed preference, and the most uncertain candidates are sampled to replace the lowest-ranked selected memories. Evaluation uses the deterministic ranked set without exploration injection. Consequently, a training prompt need not equal the strict top-K under Eq. 10, while an evaluation prompt does.

B.5 Region Failure Summary Construction

All ALFWorld RFS variants share the same parsing, normalization, and formatting procedure; they differ only in the set of failed memories supplied to that procedure. When a failed trajectory is stored, the shared memory-generation pipeline makes a reflection call with temperature 0.2 and a maximum output length of 1,024 tokens. The requested response has four structured fields: FAILURE-MODE, MISTAKES, FIXES, and AVOID. A fallback reflection is used when the initial response cannot be parsed. A record is eligible for aggregation only if parsing recovers a failure mode or at least one mistake.

Before aggregation, parsed entries pass through a benchmark-specific canonicalization step that replaces instance-specific entities with generic type placeholders when such mappings are available. For example, in ALFWorld, concrete object and receptacle names are replaced with `[object]` and `[receptacle]`; analogous benchmark-specific mappings can be used for other domains, while entries without a defined mapping remain unchanged. The formatter then counts the resulting canonicalized strings independently for failure modes, mistakes, and fixes and retains the three most frequent entries in each field. Thus, entries coalesce only when their canonicalized strings are identical; the aggregation performs no embedding-based or paraphrase matching and imposes no support threshold beyond occurrence in the evidence set. Frequency ties follow the order in which entries are first encountered. Although AVOID is parsed, the current formatter does not emit it. The output reports the number of failed records represented, failure modes with their counts, up to three key mistakes, and up to three recovery strategies. This deterministic counting and formatting step requires no additional LLM call.

Let $\mathcal{A}(E)$ denote this common aggregation operator applied to an ordered evidence set E: it filters unusable records, parses and normalizes the retained records, constructs the three frequency counters, takes the top three entries from each counter, and fills the fixed summary template. Let $\mathcal{M}^-$ be the failed records in the current memory cache, $r(m) = \arg\max_\ell w_{m,\ell}$ the primary region of memory m, and $s(m)$ its stored source-subtask label. The four RFS forms evaluated in Section 4.3 are produced as follows.

Regional RFS (CORA). After region formation, the system assigns every memory to $r(m)$ and constructs one evidence set per region,

$$E_{\text{region}}(\ell) = \{m \in \mathcal{M}^- : r(m) = \ell\}, \qquad \text{RFS}_{\text{region}}(\ell) = \mathcal{A}(E_{\text{region}}(\ell)). \tag{22}$$

It reads the stored reflection of every failed primary member, parses its fields, and aggregates all valid records assigned to that region. The resulting string is stored with the region and rebuilt after clustering or topology maintenance. If ordinary retrieval later selects a failed memory m, CORA looks up the cached summary for $r(m)$ and uses it to replace that memory's content. An empty regional summary causes the replacement to abstain, leaving the original failed memory unchanged.

Inline RFS. When a naturally retrieved failure is to be replaced for task description x, the inline variant runs a second, failure-only semantic retrieval over the vector-backed memory pool. It embeds x, restricts candidates to failed memories, ranks them by semantic relevance, and returns the top ten:

$$E_{\text{inline}}(x) = \text{Top10}_{m \in \mathcal{M}^-} \text{sim}(x, m), \qquad \text{RFS}_{\text{inline}}(x) = \mathcal{A}(E_{\text{inline}}(x)). \tag{23}$$

The ten returned records are read in retrieval order; their reflection fields are parsed and normalized, and their failure modes, mistakes, and fixes are frequency-counted to produce the inline RFS. The implementation records the contributing memory identifiers and evidence count for attribution. This summary is query-specific, is constructed online for the replacement, and is not cached; consequently, inline RFS incurs one additional query embedding and vector search whenever this construction is triggered.

Global RFS. The global variant scans the current memory cache and selects every record marked unsuccessful with nonempty reflection content:

$$E_{\text{global}} = \mathcal{M}^-, \qquad \text{RFS}_{\text{global}} = \mathcal{A}(E_{\text{global}}). \tag{24}$$

It does not perform a query-conditioned retrieval: all valid failures contribute to the same counters irrespective of region, semantic similarity, or source subtask.

Subtask RFS. For target subtask c, this variant scans the same failed-memory cache but retains only records whose stored `source_subtask` exactly equals c:

$$E_{\text{subtask}}(c) = \{m \in \mathcal{M}^- : s(m) = c\}, \qquad \text{RFS}_{\text{subtask}}(c) = \mathcal{A}(E_{\text{subtask}}(c)). \tag{25}$$

It then parses, normalizes, counts, and formats that subset using the same top-three procedure. This path performs no additional embedding retrieval, but it rescans the cache and rebuilds the summary for the requested subtask rather than using the global-summary cache.

Thus, all four forms use identical field extraction and output formatting. Regional RFS pools failures with the same primary capability region, inline RFS pools the ten query-relevant failures, global RFS pools all failures, and subtask RFS pools failures with an exactly matching source-subtask label. The ablation therefore changes the source and scope of failure evidence rather than the summary schema or output budget.

C Additional Experimental Results

C.1 Training Results

Table 5 reports the training results corresponding to the test results in Table 1. For iterative-memory methods, each cell gives the E10 independent success rate (SR), the fraction of tasks solved in the final epoch, followed by the cumulative success rate (CSR), the fraction of distinct tasks solved at least once over all ten epochs. In the first row, the two values instead report the single-run no-memory SR and the Pass@10 success rate, respectively.

Training performance. Table 5 shows that CORA achieves the highest cumulative training SR in four of the six settings, with the clearest gains on ALFWorld. Under Qwen3.6-35B-A3B, CORA improves over MemRL by 5.49 percentage points in E10 SR and 4.86 points in cumulative SR; under Qwen2.5-72B-Instruct, the corresponding gains are 5.91 and 6.48 points. With GPT-5-mini, CORA attains the highest cumulative SR (99.0%) while its E10 SR is 1.3 points below MemRL, suggesting broader task coverage despite a slightly lower final-epoch result. On LLB-OS, CORA similarly achieves the highest cumulative SR and comes within 0.29 points of the best E10 SR. CORA does not maximize the training metrics on BigCodeBench or LLB-DB, where RAG or MemRL performs better in sample. Nevertheless, CORA obtains stronger test results in Table 1, indicating that its test-time advantage is not merely explained by solving more training tasks and is consistent with improved reuse of accumulated memories across unseen instances.

Table 5: Training results. Each cell reports E10 independent SR / cumulative SR (%). In the first row, the two values are the single-run No Memory SR / Pass@10 success rate. The best result for each metric is bolded.

(a) ALFWorld

Method	*Qwen3.6-35B-A3B*	*Qwen2.5-72B-Instruct*	*GPT-5-mini*
No Memory / Pass@10	36.42 / 63.52	44.72 / 52.32	77.7 / 92.8
RAG	60.36 / 74.30	59.19 / 68.98	88.7 / 93.0
Self-RAG	54.18 / 77.48	59.08 / 61.27	90.7 / 96.2
Mem0	46.50 / 75.29	67.32 / 84.89	89.4 / 96.9
MemP	41.89 / 66.17	48.16 / 55.76	88.5 / 91.9
MemRL	68.17 / 89.45	71.69 / 83.28	**94.9** / 98.1
CORA	**73.66 / 94.31**	**77.60 / 89.76**	93.6 / **99.0**

(b) BigCodeBench and Lifelong Agent Bench

Method	BigCodeBench *DeepSeek-V3*	LLB-OS *GPT-4.1-mini*	LLB-DB *GPT-4.1-mini*
No Memory / Pass@10	46.49 / 48.62	64.29 / 77.33	79.50 / 83.10
RAG	**59.77** / 62.78	**76.86** / 77.14	**86.15 / 86.70**
Self-RAG	53.63 / 60.78	74.29 / 77.14	84.76 / 85.32
Mem0	46.62 / 57.89	69.14 / 76.29	79.50 / 83.10
MemP	50.25 / 62.28	74.86 / 78.29	82.83 / 85.04
MemRL	55.01 / **66.79**	74.29 / 78.00	83.66 / 85.32
CORA	52.01 / 65.29	76.57 / **78.86**	83.38 / 84.76

	Train SR (%)	Validation SR (%)		
Method	E10	E1–E10 mean	E10	Peak
CLAG	50.13	48.60	47.66	49.71
A-MEM	51.63	48.48	48.54	49.42
CORA	**52.01**	**51.75**	**49.71**	**54.09**

Table 6: BigCodeBench training and validation success rates with DeepSeek-V3.

C.2 Comparison with Graph-Based Memory Baselines

We further compare CORA with two structured-memory methods on BigCodeBench: CLAG (Roh et al., 2026), which routes memories through semantic clusters, and A-MEM (Xu et al., 2025), which links and evolves related memory notes. All three methods use DeepSeek-V3, Qwen3-Embedding-8B, the same 798/342 train/validation split, and ten epochs of memory accumulation. Their training and validation results are summarized in Table 6.

At E10, CORA reaches 49.71% validation SR, compared with 47.66% for CLAG and 48.54% for A-MEM. CLAG and A-MEM exhibit only modest fluctuations in validation performance across epochs, with mean success rates of 48.60% and 48.48%, respectively. Their peak success rates of 49.71% and 49.42% also remain below CORA's 54.09%. CORA achieves a higher mean validation success rate of 51.75%, indicating stronger performance over the course of memory accumulation. These results suggest that organizing memories by observed task-specific utility supports more effective reuse than semantic organization alone.

C.3 Holdout Subtask Transfer

C.3.1 Setup

Setup. We introduce a zero-shot subtask transfer setting to evaluate whether CORA's capability-coherent region structure—capturing consistent patterns of competence and failure—transfers to an unseen task category. Given a benchmark with K categories $\mathcal{C} = \{c_1, \ldots, c_K\}$, we hold out $c^* \in \mathcal{C}$ and train exclusively on the remaining $K - 1$ categories. At evaluation, the memory bank therefore contains only trajectories accumulated on the observed categories, while the retrieval query is the held-out episode's task description x.

Held-out utility. No direct outcome is available for c^* during training, so the memory-level estimate remains at its neutral initialization and contributes no category-specific evidence. The confidence-gated estimate in Eq. 8 therefore relies entirely on the region-level estimate:

$$\tilde{Q}_i(c^*) = U_{\text{agg}}(m_i, c^*), \qquad U_{\text{agg}}(m_i, c^*) = \sum_{\ell \in \text{top-}N(m_i)} \bar{w}_{i,\ell} \widehat{U}_\ell(c^*), \tag{26}$$

where N is the number of highest-membership regions used for aggregation and $\bar{w}_{i,\ell}$ denotes their normalized membership weights. Because no region has direct evidence for c^*, its utility $\widehat{U}_\ell(c^*)$ is inferred by transferring capability residuals from semantically related observed categories. This transfer operates across task categories to estimate region utility for the unseen category.

More precisely, let $\mathcal{S} = \mathcal{C} \setminus \{c^*\}$ denote the globally observed categories, let $n_{\ell,s}$ be the direct evidence count for region ℓ and category s, and let $U_\ell(s)$ be the corresponding region utility. The source-category baseline is

$$\mu_s = \frac{\sum_{\ell:\, n_{\ell,s}>0} n_{\ell,s} U_\ell(s)}{\sum_{\ell:\, n_{\ell,s}>0} n_{\ell,s}}. \tag{27}$$

The global reference pools the same evidence across all observed categories:

$$\mu_{\text{global}} = \frac{\sum_\ell \sum_{s\in\mathcal{S}:\, n_{\ell,s}>0} n_{\ell,s} U_\ell(s)}{\sum_\ell \sum_{s\in\mathcal{S}:\, n_{\ell,s}>0} n_{\ell,s}}. \tag{28}$$

Let e_s and e_{c^*} denote the source- and target-category embeddings. For each region, let $\mathcal{S}_{\ell,\text{emb}} \subseteq \mathcal{S}$ contain source categories with an estimated region utility and an embedding. Their transfer weights are

$$\alpha_{\ell,s}(c^*) = \frac{\exp(\tau_z \cos(e_{c^*}, e_s))}{\sum_{t\in\mathcal{S}_{\ell,\text{emb}}} \exp(\tau_z \cos(e_{c^*}, e_t))}, \qquad s \in \mathcal{S}_{\ell,\text{emb}}. \tag{29}$$

Here, τ_z is a temperature hyperparameter controlling how strongly transfer concentrates on the most similar source categories. The unseen-category region utility is then estimated as

$$\widehat{U}_\ell(c^*) = \text{clip}_{[u_{\min},\, u_{\max}]} \left[\mu_{\text{global}} + \sum_{s\in\mathcal{S}_{\ell,\text{emb}}} \alpha_{\ell,s}(c^*) \big(U_\ell(s) - \mu_s\big) \right], \tag{30}$$

where $u_{\min}$ and $u_{\max}$ bound the transferred estimate. Centering each source utility by μ_s removes source-category difficulty, so the transferred signal captures whether region ℓ is relatively stronger or weaker than average. Adding this residual to μ_{global} converts the relative capability signal back to the utility scale, while direct-evidence mass determines how much each observed cell contributes to the source baselines and global reference.

Retrieval. A region utility alone does not select memories. For each held-out description x, semantic retrieval first recalls a query-specific candidate pool and then reranks it using Eq. 10, with $c_{\text{target}} = c^*$ and the transferred estimate $\tilde{Q}_i(c^*)$ from Eq. 26. Subtask-to-subtask similarity in Eq. 29 transfers capability information to the unseen category, whereas the query-to-memory similarity in Eq. 10 preserves episode-specific relevance during recall and reranking.

C.3.2 Results

Setting. We conduct an exploratory zero-shot validation study on two ALFWorld subtasks, holding out `pick_and_place` and `pick_two` in separate runs and training the memory system only on the remaining subtask categories. All methods use Qwen2.5-72B-Instruct, and performance is measured by the peak validation success rate across the evaluated checkpoints. For CORA, the rapid early improvement in training SR largely plateaus by S4, with subsequent stage-to-stage gains generally around one percentage point or less; this training-side trend provides a practical indicator of where peak validation performance occurs, and both held-out subtasks attain their observed peak at S4. We set $N = 1$, $\tau_z = 5$, $(u_{\min}, u_{\max}) = (0.05, 0.95)$, and $\lambda = 0.55$, which assigns a weight of 0.45 to query–memory similarity.

Results. As shown in Table 7, CORA achieves the highest observed validation success rate on both held-out subtasks: 95.76% on `pick_and_place` and 55.79% on `pick_two`. These exploratory results suggest that utility patterns learned from observed categories can improve memory retrieval for a held-out category.

Table 7: Exploratory ALFWorld holdout results. Values are peak validation success rates (%) on each held-out subtask.

	ALFWorld	
Method	`pick_and_place` ($n = 825$)	`pick_two` ($n = 837$)
No Memory	91.88	39.67
RAG	94.79	50.54
Self-RAG	92.85	46.36
Mem0	93.45	43.37
MemP	93.09	41.94
MemRL	94.42	55.08
CORA	**95.76**	**55.79**

Table 8: Task-paired E10 comparisons against MemRL. SR values are percentages; gain and its paired bootstrap 95% CI are percentage points. Unique wins are CORA-only/MemRL-only successes.

Benchmark endpoint	N	CORA SR	MemRL SR	Gain [95% CI]	Unique wins	Exact p
ALFWorld Qwen2.5-72B-Instruct train	3,553	**77.60**	71.69	**+5.91 [5.09, 6.75]**	**226/16**	< 0.001
LLB-OS train	350	**76.57**	74.29	+2.29 [-0.29, 4.86]	**14/6**	0.115
LLB-DB train	361	83.38	83.66	-0.28 [-1.94, 1.39]	5/6	1.000

D Statistical Analysis and Computational Overhead

D.1 Paired Task-Level Evidence

D.1.1 Main Experiments

We complement the aggregate results with task-paired E10 training comparisons against MemRL wherever stable task identifiers are available. We report the paired success-rate gain with a 95% confidence interval obtained by resampling task IDs with replacement 10,000 times (seed 42). "Unique wins" counts tasks solved only by one of the two methods, and the exact p value is obtained from the two-sided McNemar test on these discordant outcomes.

The paired analysis strengthens the ALFWorld result: CORA improves E10 success by 5.91 points, with the entire confidence interval above zero. It solves 226 trajectories that MemRL misses, while MemRL has only 16 unique wins, yielding a highly significant exact test ($p < 0.001$). LLB-OS provides additional favorable task-level evidence, with a 2.29-point gain and 14 unique wins versus 6 for MemRL. On LLB-DB, the two methods differ by only one successful task at the paired endpoint, indicating essentially matched performance.

A paired validation analysis is not available for the main ALFWorld and LLB experiments because their evaluation records retain aggregate validation success rates but not per-game binary outcomes with stable task identifiers. In addition, some checkpoints needed to regenerate these outcomes were removed under the experiment storage-retention policy. Exact validation pairs therefore cannot be reconstructed retrospectively.

D.1.2 Ablation Experiments

BigCodeBench retains stable validation task IDs across ablation configurations, enabling paired comparisons of the full CORA main run against Category-Q alone and embedding-space regions.

Full CORA achieves higher validation success than both component alternatives. It improves over Category-Q alone by 2.34 points and records 23 CORA-only wins versus 15 comparison-only wins, demonstrating the benefit of region-level estimation and aggregation beyond category-conditioned utility. It also improves over embedding-space regions by 3.22 points, with 25 CORA-only wins versus 14 comparison-only wins, supporting capability profiles as a more effective basis for region formation.

Table 9: BigCodeBench task-paired ablations on 342 validation tasks at E10. Unique wins are CORA-only/comparison-only successes.

Comparison	CORA SR	Comparison SR	Gain [95% CI]	Unique wins	Exact p
Category-Q only	**49.71**	47.37	+2.34 [-1.17, 5.85]	**23/15**	0.256
Embedding-space regions	**49.71**	46.49	+3.22 [-0.29, 6.73]	**25/14**	0.108

Table 10: Mean ALFWorld Qwen2.5-72B-Instruct prompt tokens per call by RFS summary form.

Summary form	Train	Evaluation
Regional	3,670.0	6,044.0
Inline ($k = 10$)	3,660.6	6,047.8
Global	3,574.8	5,889.9
Subtask	3,577.0	5,918.4

D.2 ALFWorld Online Retrieval and Storage Audit

CORA retains MemRL's two-stage online retrieval structure: a query embedding and vector search first form the semantic candidate set, after which the candidates are reranked. CORA does not issue an additional query-embedding call on this normal retrieval path. Instead, its additional online work consists of reading cached category-conditioned memory utilities, region utilities, and memberships and combining them for the recalled candidates. Conditional RFS likewise introduces neither an additional retrieval nor an online summarization call: when the naturally selected set contains a failed memory with an available cached region summary, it replaces at most one retrieved failure slot with that summary.

The archived runs do not provide a controlled end-to-end latency comparison: their execution histories differ, and the logs do not isolate vector search, CORA reranking, conditional replacement, and transport under matched requests. We therefore characterize the online path by its additional operations and report the recoverable token and storage measurements below, rather than comparing wall-clock spans or embedding timings across runs.

For example, in one CORA run on ALFWorld, the full snapshot occupies 6.3 GB: 4.9 GB stores the textual memories, 1.4 GB stores the vector index, and approximately 70 MB stores CORA's persistent region state. This state comprises the region topology, centroids, memberships, region-level utility and evidence statistics, and per-memory category-conditioned utilities and metadata. The persistent region state is therefore at the MB scale, while the underlying textual and vector memory stores are at the GB scale.

The per-run `token_usage.jsonl` files do, however, recover prompt-token counts for the four ALFWorld RFS summary forms. Table 10 reports per-call means separately for training and evaluation. Regional and inline summaries have nearly identical means, while global and subtask summaries are slightly shorter; the range across forms is approximately 2.6% in each phase. RFS replaces one retrieved failure slot rather than adding another slot. In general, the resulting structured summary is shorter than the raw failure trajectory it replaces because it retains only the most frequent failure modes, mistakes, and recovery strategies.

The summary block itself is produced by deterministic aggregation and incurs no online summarization call, although the stored failure records used as its inputs are created by the shared store-time reflection procedure. Generating agent responses remains the dominant LLM workload.