BENCHMARKING CONTINUAL AGENT MEMORY FOR ONLINE LEARNING, TRANSFER, AND FORGETTING

Zihang Ma[1†] **Jinyi Liu**[1†]
Hongyao Tang[1] Yi Ma[2] Ruitao Wang[1] Yifu Yuan[1] Yan Zheng[1] Jianye Hao[1]
[1]Tianjin University [2]Shanxi University
3023209299@tju.edu.cn
[†]Both authors contributed equally to this research.

ABSTRACT

Effective long-term memory for LLM-based agents encompasses two fundamentally distinct capabilities: *system memory* (a.k.a. experiential memory), which distills reusable procedural knowledge from task execution, and *personal memory* (a.k.a. factual memory), which retains user-specific facts and preferences across sessions. In real-world deployments, these two memory types co-evolve with blurred boundaries—yet existing methods and benchmarks treat them in isolation, an assumption that breaks down the moment an agent must simultaneously execute tasks and serve a persistent user. We introduce **AgentMemoryBench**, the first benchmark to jointly evaluate system and personal memory under a unified continual-learning framework. It spans six datasets across four environment types and standardizes five complementary evaluation modes (offline, online, replay, transfer, and repair) to measure improvement, retention, forgetting, generalization, and knowledge conflict resolution over time. Building on this benchmark, we propose **MEMs**, a multi-memory coordination framework that maintains separate specialized stores and employs a lightweight trigger model as a meta-cognitive router to selectively retrieve and update each store. Experiments reveal that single-memory and in-context learning designs suffer systematic performance collapse in mixed-task regimes due to memory contamination and architectural mismatch, while MEMs maintains stable learning across task boundaries. AgentMemoryBench establishes a reproducible evaluation loop and provides practical guidance for developing memory systems that hold up in the real world.

Source code: https://github.com/s010m00n/AgentMemoryBench.

1 INTRODUCTION

Effective long-term memory for LLM-based agents encompasses two fundamentally distinct capabilities. *System memory* (a.k.a. experiential memory) distills reusable procedural knowledge from task execution—tool traces, intermediate artifacts, and workflows—so that agents improve their task success over time. *Personal memory* (a.k.a. factual memory) retains user-specific preferences, facts, and dialogue events across sessions, enabling long-horizon personalization. Both are prerequisites for continual adaptation (Hu et al., 2026b; Wu et al., 2025b): without system memory, agents repeat the same mistakes; without personal memory, every session starts from scratch.

In real-world deployments, however, these two memory types co-evolve with blurred boundaries. Consider an agent that alternates between database query tasks and open-ended user dialogue: a system-memory method (e.g., AWM (Wang et al., 2024)) can induce reusable SQL workflows from task trajectories but cannot track the user's personal preferences; a personal-memory method (e.g., Mem0 (Chhikara et al., 2025)) retains factual user context but cannot generalize procedural experience across task variants. Neither type alone is sufficient, as shown in Figure 1—yet existing methods and benchmarks treat them in isolation, an assumption that breaks down the moment an agent must simultaneously execute tasks and serve a persistent user (Wu et al., 2024; Zheng et al., 2025; Zhang et al., 2025; Zhi et al., 2025).

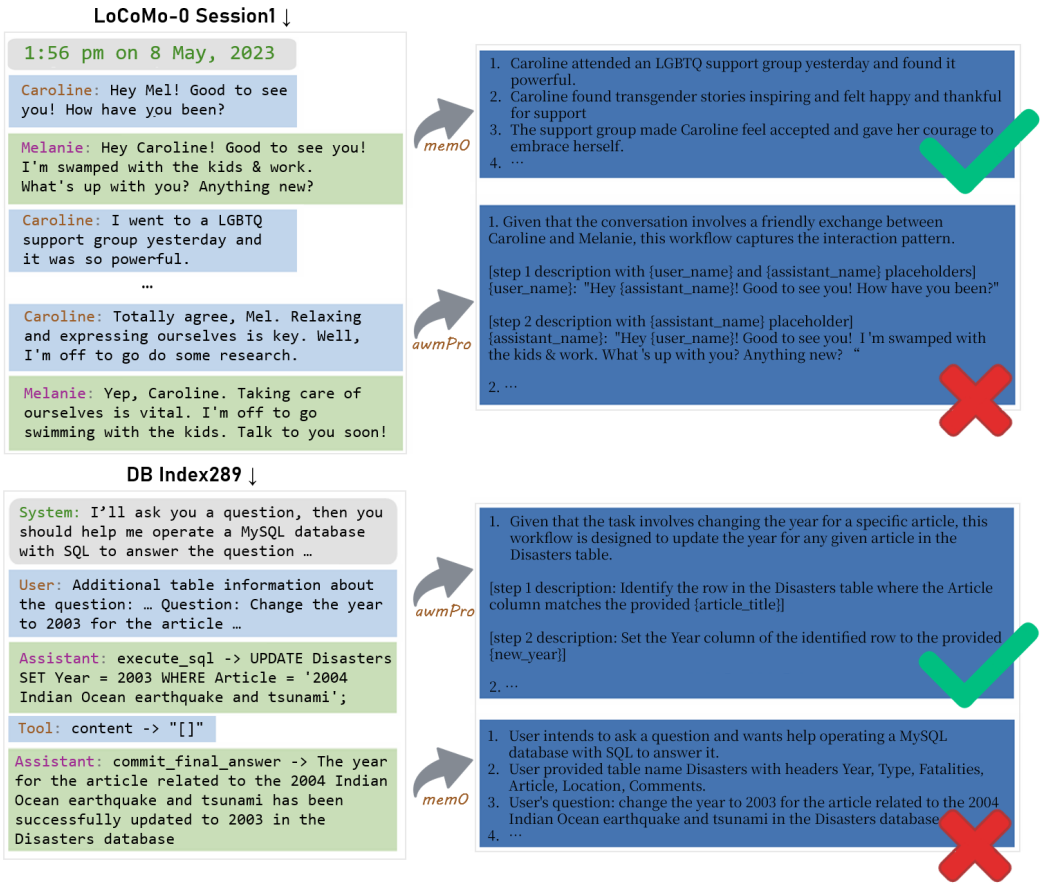

Figure 1: Comparison of Personal Memory Method (mem0) and System Memory Method (awmPro) Performance on Two Real-world Sub-task Trajectories.

Additionally, existing benchmarks further fail to capture the dynamic nature of memory. Most adopt static, offline evaluation that tests performance at a single time point, concealing learning dynamics such as knowledge retention, forgetting, forward/backward transfer, and conflict resolution (Ai et al., 2025; Maharana et al., 2024; Wu et al., 2025a). Even stream-based protocols (Wu et al., 2024; Zheng et al., 2025; Wei et al., 2025) cover only one memory type and lack the complete **CONTINUAL-LEARNING** metrics needed to capture retention, forgetting, and transfer (Shi et al., 2024), as well as dedicated conflict-resolution metrics for evaluating how agents reconcile logical contradictions between new and prior knowledge—analogous to the **EDIT SUCCESS** measures introduced in KnowEdit (Zhang et al., 2024).

To close this gap, we introduce **AgentMemoryBench**, the first benchmark to jointly evaluate system and personal memory under a unified continual-learning framework with five complementary modes: offline, online, replay, transfer, and repair (Figure 3)—designed to measure improvement, retention, forgetting, generalization, and knowledge-conflict resolution respectively. AgentMemoryBench spans four grounding types—code-grounded (Liu et al., 2025), embodied (Shridhar et al., 2021), web-grounded (Yao et al., 2023), and dialogue-grounded (Maharana et al., 2024) environments—and supports seeded task interleaving to emulate boundary-blurred real-world streams, as shown in Figure 2. Building on AgentMemoryBench, we propose **MEMs**, a multi-memory coordination framework that maintains separate specialized stores and employs a lightweight trigger model as a meta-cognitive router to selectively retrieve from and update each store.

Our contributions are threefold. **First**, we present **AgentMemoryBench**, the first unified benchmark that simultaneously evaluates system memory (experiential) and personal memory (factual) across

five complementary modes (as shown in Figure 3), enabling rigorous measurement of learning dynamics, retention, forgetting, generalization, and conflict resolution in a single reproducible framework. **Second**, we propose **MEMs**, a multi-memory coordination framework that maintains separate specialized stores and employs a lightweight trigger model to route retrieval and updates, enabling stable procedural reuse and personalization under blurred task boundaries. **Third**, experiments reveal that single-memory and in-context learning designs suffer systematic performance collapse in mixed personal–system regimes, while MEMs maintains stable learning across task boundaries, as shown in Figure 5.

2 Related Work

Memory methods vs. in-context learning. In-context learning (ICL) is a prompt-learning paradigm (Dou et al., 2026): it incorporates a few input–output examples into the prompt, allowing models to infer task format and expected behavior without updating any parameters. Memory methods operate on a fundamentally different principle: they persist knowledge in external stores and follow a structured retrieve–form–evolve lifecycle (Section 3.1.2), enabling agents to accumulate and reuse experience across tasks. Memory methods are further divided into *system memory* methods, which extract reusable procedural knowledge from task execution (e.g., AWM (Wang et al., 2024), MemRL (Zhang et al., 2026a), AgeMem (Yu et al., 2026)), and *personal memory* methods, which retain user-specific facts and preferences (e.g., Mem0 (Chhikara et al., 2025), MemOS (Li et al., 2025), EverMemOS (Hu et al., 2026a)).

Benchmarks have driven rapid progress in agent memory, mirroring their role in advancing reasoning (Wu et al., 2025b), general task-solving (Liu et al., 2025), and agentic abilities (Yao et al., 2023; Shridhar et al., 2021). However, existing agent memory benchmarks bifurcate into personal memory (e.g., LoCoMo (Maharana et al., 2024), HaluMem (Chen et al., 2026)) and system memory (e.g., StreamBench (Wu et al., 2024), LifelongAgentBench (Zheng et al., 2025)), each evaluated in isolation. This fragmentation leaves a critical gap, as shown in Figure 1: methods that attempt to bridge both types—such as MemGen (Zhang et al., 2025), which unifies heterogeneous memory representations via latent token sequences—have no benchmark that supports the mixed-scenario streams in which such coordination is actually needed. MemoryBench (Ai et al., 2025) partially addresses this by covering both memory types, but restricts evaluation to offline, static snapshots—concealing how memory quality evolves under continuous interaction.

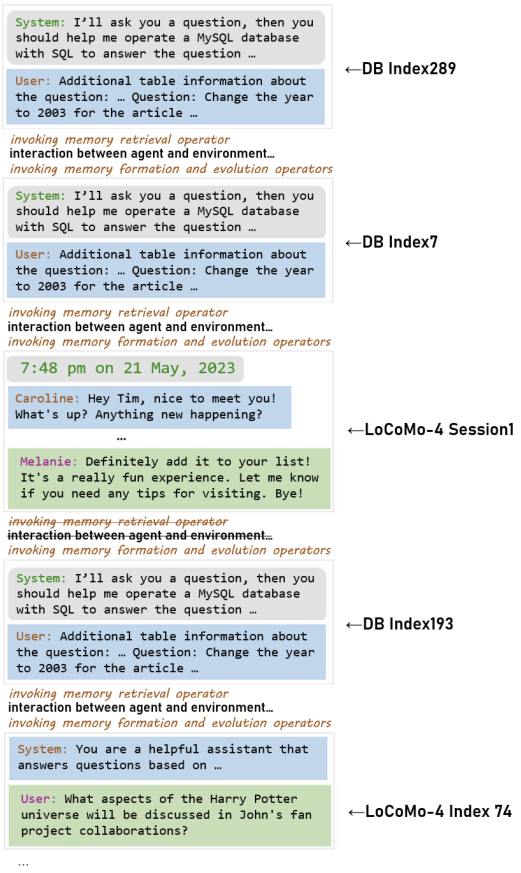

Figure 2: Illustration of the memory lifecycle (retrieval, formation, and evolution operators) invoked across a mixed task stream (DB and LoCoMo-4 interleaved, and the seed is 66).

Offline evaluation conceals learning dynamics. Most existing benchmarks adopt a static train-test split (e.g., LongMemEval (Wu et al., 2025a), KnowMe-Bench (Wu et al., 2026), PersonaMem-v2 (Jiang et al., 2025), and DMR (Packer et al., 2024)) , measuring only terminal performance. This snapshot cannot reveal how memory quality evolves over time. Stream-based protocols (e.g., StreamBench (Wu et al., 2024), LifelongAgentBench (Zheng et al., 2025), Evo-Memory (Wei et al., 2025)) introduce online learning curves, but cumulative success rate alone is insufficient: ICL

Table 1: Comparison of AgentMemoryBench with Existing Benchmarks across Memory Categorization, Evaluation Modes, and Continual Learning Metrics. See Appendix E.6 for column definitions and symbol guide.

Benchmark	Memory Eval	CL Metrics	Distinguish Learn	Hybrid Memory	Multi-turn	Tool-use	Single-task Online	Multi-task Online	Offline	Intra-env Transfer	Cross-env Transfer	Replay	Repair
AgentMemoryBench (ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LoCoMo† (Maharana et al., 2024)	✓	□	×	×	×	×	×	×	✓	×	×	×	×
HaluMem‡ (Chen et al., 2026)	✓	×	×	×	×	×	✓	×	×	×	×	×	×
StreamBench (Wu et al., 2024)	✓	×	×	×	×	✓	✓	×	×	✓	×	×	×
LifelongAgentBench (Zheng et al., 2025)	✓	×	×	×	×	✓	✓	×	×	✓	×	×	×
Evo-Memory (Wei et al., 2025)	✓	×	×	×	✓	✓	✓	✓	×	×	✓	×	×
MemoryBench (Ai et al., 2025)	✓	□	×	✓	✓	×	×	×	✓	×	×	×	×
Mem2ActBench (Shen et al., 2026)	×	□	×	□	×	✓	□	□	□	□	□	□	□
CL-bench§ (Dou et al., 2026)	×	□	✓	□	✓	✓	□	□	□	□	□	□	□

✓: supported; ×: not supported; □: not applicable
† Also includes LongMemEval (Wu et al., 2025a), KnowMe-Bench (Wu et al., 2026), PersonaMem-v2 (Jiang et al., 2025), and DMR (Packer et al., 2024); ‡ Also includes AMemGym (Jiayang et al., 2026); § Also includes CONTEXTBENCH (Li et al., 2026) and CODE2BENCH (Zhang et al., 2026b)

baselines such as StreamICL (Wu et al., 2024) can match dedicated memory methods on this metric by activating LLM pretraining priors rather than accumulating genuine knowledge. As shown in Tables 6–9 (see Appendix A), StreamICL can match or exceed dedicated memory methods under offline and basic online metrics. Thus, distinguishing persistent memory methods from StreamICL requires generalization metrics, continual learning indicators (forgetting, forward transfer), and knowledge-conflict resolution measures—none of which existing benchmarks provide.

Context-learning benchmarks offer a complementary perspective. CL-bench (Dou et al., 2026), CONTEXTBENCH (Li et al., 2026), and CODE2BENCH (Zhang et al., 2026b) address an analogous discrimination problem within a single session: separating genuine context learning from mechanical retrieval. Mem2ActBench (Shen et al., 2026) extends this to memory-grounded tool calls, but remains a context-learning benchmark and does not resolve this discrimination problem. Long-term memory evaluation faces the same core challenge at a different timescale—across the full lifetime of an agent rather than within one context window. Our replay, transfer, and repair modes extend this evaluation philosophy to the long-term memory setting, providing the multi-dimensional protocol needed to rigorously assess persistent memory methods and distinguish them from in-context learning baselines. A detailed benchmark comparison is provided in Table 1.

For an extended discussion of memory benchmarks and methodologies, please refer to Appendix E and Appendix F.

3 AGENTMEMORYBENCH AND MEMS

AgentMemoryBench addresses a fundamental evaluation gap: *can existing memory methods maintain stable performance when system and personal tasks co-occur with blurred boundaries?* It comprises six datasets across four environment types (Table 2) and standardizes five complementary evaluation modes—offline, online, replay, transfer, and repair—to measure learning improvement, knowledge retention, forgetting, cross-task generalization, and conflict resolution. To complement the benchmark, we further propose **MEMs**, a multi-memory coordination framework that employs a lightweight trigger model to route retrieval and updates across specialized system and personal memory stores.

Table 2: Summary of AgentMemoryBench Datasets

Dataset	Environment Type	Memory Type	# Sub-tasks	Evaluation Metrics
DB (Liu et al., 2025)	Code-grounded	System	300	ASR, AS
OS (Liu et al., 2025)	Code-grounded	System	144	ASR, AS
KG (Liu et al., 2025)	Code-grounded	System	150	ASR, AS
ALFWorld (Shridhar et al., 2021)	Embodied	System	109	ASR, AS
WebShop (Yao et al., 2023)	Web-grounded	System	200	ASR, AS
LoCoMo (Maharana et al., 2024)	Dialogue-grounded	Personal	∼2000	F1, BLEU, LLM as J

Detailed construction protocols and task specifications are provided in Appendix G

3.1 Problem Formulation

Each **task** in AgentMemoryBench corresponds to a **dataset**; each dataset contains multiple **sub-tasks**—individual samples requiring single-turn or multi-turn agent–environment interactions.

To systematically evaluate memory systems across these datasets and evaluation modes, we establish a formal framework defining the agent-environment interaction protocol. This formulation specifies how memory is retrieved before each sub-task and how new experiences are transformed into persistent memory upon completion.

3.1.1 Single sub-task trajectory definition

We formalize memory evaluation in large language model (LLM) based agents as follows. Consider an agent executing a sub-task within an environmental state space S. The state evolves according to a stochastic transition model Ψ:

$$s_{t+1} \sim \Psi(s_{t+1}|s_t, a_t)$$

where a_t denotes the action at each time step t. At each step t, the agent receives an observation:

$$o_t = O(h_t, Q)$$

Here, h_t represents the interaction history visible during the current sub-task, typically the agent-environment trajectory. Q denotes sub-task requirements, including the system prompt, toolset T, and objective. O is the observation function that constructs the agent's current view by combining interaction history with task requirements. The agent selects actions following a policy:

$$a_t = \pi(o_t, m_t), a_t \in T$$

where m_t is the memory signal retrieved from system M_t:

$$m_t = R(M_t, o_t)$$

The retrieval operator R identifies relevant content based on the observation and returns a formatted signal, such as text snippets or summaries, for the LLM policy.

3.1.2 Memory system lifecycle

The memory lifecycle during sub-task k execution is examined below. The memory system $M_t \in \mathcal{M}$, where $\mathcal{M}$ is the space of permissible configurations, evolves through three operators. The memory retrieval operator R extracts relevant information at time t based on the observation and memory state M_t:

$$m_t = R(M_t, o_t), \quad t \geq 0$$

The memory formation operator F transforms informational artifacts ϕ_t generated after an action into memory candidates. These artifacts may include tool outputs, reasoning, plans, self-evaluations, or environmental feedback:

$$M_{t+1}^{form} = F(M_t, \phi_t), \quad t \geq 0$$

The memory evolution operator E integrates candidates into the repository. The evolved state M_{t+1} persists for subsequent decisions and sub-tasks:

$$M_{t+1} = E(M_{t+1}^{form}, M_t), \quad t \geq 0$$

Most studies (e.g., StreamBench (Wu et al., 2024), LifelongAgentBench (Zheng et al., 2025), Evo-Memory (Wei et al., 2025)) adopt a sub-task-level update hypothesis. Instead of step-wise invocation, R is called once before sub-task k. Upon completion, F and E are executed sequentially:

$$m_k = R(M_k, Q_k), \quad M_{k+1}^{form} = F(M_k, \phi_k), \quad M_{k+1} = E(M_{k+1}^{form}, M_k)$$

where M_k is the state at the onset of sub-task k, Q_k denotes requirements, and M_{k+1} is the state after completion. Term ϕ_k refers to artifacts produced at the conclusion, such as outcomes, feedback, or interaction history.

3.2 Evaluation Modes Of AgentMemoryBench

AgentMemoryBench supports five distinct evaluation modes: Offline, Online, Replay, Transfer, and Repair (Figure 3). These modes collectively provide a multi-dimensional assessment of memory system capabilities, ranging from terminal performance and learning dynamics to knowledge retention, generalization, and conflict resolution.

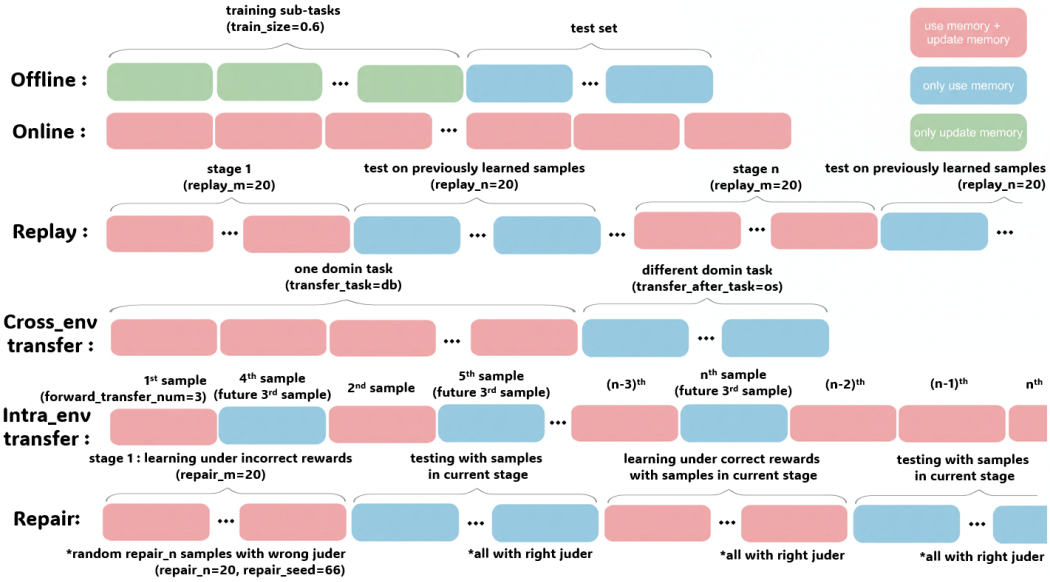

Figure 3: Evaluation modes and task pipelines of AgentMemoryBench.

Offline Mode follows the traditional train-test split setting. The agent first executes all training sub-tasks to build memory (invoking only memory formation and evolution operators), then performs on the test set using the constructed memory. This mode evaluates the final effectiveness of the memory system through metrics like Average Success Rate (**ASR**) and Average Step (**AS**) for system tasks, or **F1-score**, **BLEU-score**, and **LLM-as-Judge** for personal memory tasks.

Online Mode adopts a streaming setting where the agent processes sub-tasks sequentially and updates memory in real-time. After each sub-task, we record instantaneous performance to form complete learning curves, capturing dynamic transitions of knowledge accumulation, restructuring, and forgetting. The core metrics include Cumulative Success Rate (**CSR**) showing the learning trajectory, Learning Gain (**LG**) measuring improvement potential relative to future peak performance, and Stability Loss (**SL**) quantifying forgetting severity relative to historical peak. However, offline mode and single-task online mode cannot reveal the learning process or distinguish persistent memory methods from in-context learning baselines (as shown in Tables 6, 7, 8, and 9). Besides, for multi-task scenarios, we use weighted versions (**WCSR, WLG, WSL**) to aggregate performance across heterogeneous tasks. This mode is critical for observing how memory systems evolve over time, especially in mixed-task scenarios with blurred boundaries, as shown in Figure 2.

Replay Mode implements periodic testing to measure knowledge retention and anti-forgetting capability, serving as the core mechanism for quantifying backward transfer. The samples sequence is

Table 3: Online Evaluation Results on System Memory Tasks (Except WebShop)

Memory	DB		OS		KG		ALFWorld	
	ASR↑	AS↓	ASR	AS	ASR	AS	ASR	AS
zero-shot	49	2.7	36.1	2.59	24.7	10.81	17.4	18.28
streamICL	58.7	2.65	24.3	4.36	30.7	11.65	32.1	**16.43**
awmPro	53	3.13	37.06	2.57	24.7	11.13	20.2	18.06
mem0	50.3	3.16	**38.9**	2.52	22.7	11.05	7.3	18.59
MEMs	**59.7**	**2.43**	37.76	**2.46**	**36**	**10.74**	**33.9**	16.75

ASR: Average Success Rate; AS: Average Step

partitioned into multiple stages; after learning each stage, the agent is tested on previously learned samples using retrieval-only operations (without updating memory). The key metric is Forgetting Rate (**FR**), which measures performance degradation from the initial learning point to subsequent replay stages. Low FR indicates robust resistance to forgetting, while high FR signals catastrophic forgetting. This mode systematically evaluates whether memory systems can retain knowledge over extended periods rather than merely memorizing recent experiences.

Transfer Mode evaluates generalization capability through two distinct settings. Cross-environment transfer tests whether knowledge learned in one domain (e.g., DB task) can improve performance in a different domain (e.g., OS task) using Transfer Gain (**TG**) as the metric. Within-environment forward transfer measures the predictive boost for future samples after learning current ones, quantified by Forward Transfer Gain (**FTG**)—analogous to FR in Replay mode but focusing on forward impact. Together, these settings distinguish persistent memory methods—which extract abstract, reusable patterns—from in-context learning baselines that merely activate pre-training priors without accumulating transferable knowledge.

Repair Mode tests the robustness and self-correction capability of memory systems when facing erroneous feedback and knowledge conflicts. Each evaluation cycle involves four phases: (1) learning under incorrect rewards to contaminate memory, (2) testing with contaminated memory, (3) re-learning under correct rewards to repair memory, and (4) testing post-repair performance. Three normalized metrics characterize the system: Error Robustness (**ER**) measures resistance to erroneous learning, Repair Gain (**RG**) quantifies recovery capability from errors, and Net Recovery (**NR**) evaluates whether post-repair performance matches normal learning baselines. This mode is essential for validating whether memory systems possess active knowledge management mechanisms to identify and resolve conflicts, rather than passively accumulating all trajectories.

For detailed implementation specifications and metric formulations, please refer to Appendix H.

3.3 Formulation of MEMs

MEMs is defined by the tuple $(\mathcal{T}, M^{\text{sys}}, M^{\text{pers}})$, where $\mathcal{T}$ is a lightweight trigger model that routes memory operations; M^{sys} stores reusable execution patterns; and M^{pers} stores user preferences and dialogue context.

Retrieval. Before sub-task k, $\mathcal{T}$ selects active memory sources and aggregates their outputs:

$$S_{\text{ret}} = \mathcal{T}(Q_k), \quad S_{\text{ret}} \subseteq \{\text{sys, pers}\}, \quad m_k = \bigoplus_{s \in S_{\text{ret}}} R^s(M_k^s, Q_k)$$

Update. After sub-task k, $\mathcal{T}$ determines which stores to update, applying source-specific formation and evolution operators:

$$S_{\text{upd}} = \mathcal{T}(\phi_k), \quad S_{\text{upd}} \subseteq \{\text{sys, pers}\}, \quad M_{k+1}^s = E^s(F^s(M_k^s, \phi_k), M_k^s)$$

where F_{sys} extracts reusable workflows and F_{pers} extracts user facts.

4 Experiments

In this section, we systematically evaluate five memory methods using the AgentMemoryBench benchmark, focusing on three core research questions. **First**, can traditional offline evaluation (static

train-test split) adequately reflect the continual learning capabilities of agents, or does streaming evaluation provide additional value for memory systems (RQ1)? **Second**, can multi-dimensional evaluation modes (replay, transfer, and repair) rigorously distinguish persistent memory methods from in-context learning baselines that merely activate pre-training priors (RQ2)? **Third**, can a single memory system (either system memory or personal memory alone) meet the requirements of complex agents, or do real-world scenarios with fuzzy task boundaries demand multi-memory coordination such as MEMs (RQ3)?

Our experiments proceed in three stages corresponding to RQ1-RQ3. We begin by demonstrating the limitations of traditional evaluation through Offline and Online modes (Section 4.2). Three evaluation modes (Replay, Transfer, and Repair) then expose the essential differences between persistent memory methods and in-context learning baselines (Section 4.3). Finally, we verify the necessity of multi-memory coordination in simulated real-world mixed-task scenarios (Section 4.4). Detailed experimental setups, full result tables, and additional analyses are provided in Appendix A, B, C, and D.

4.1 Experiment settings

We benchmark five memory methods to comprehensively evaluate AgentMemoryBench across system and personal memory dimensions. All experiments use Qwen3-14B as the LLM backbone. The evaluated methods include: (1) **Zero-shot**, a baseline without any memory mechanism that reflects the fundamental instruction-following capability of LLMs; (2) **StreamICL** (Wu et al., 2024), a streaming in-context learning method that stores and retrieves complete interaction trajectories from successful tasks using vector databases; (3) **Mem0** (Chhikara et al., 2025), a scalable long-term memory system focused on user preference retention, employing hybrid storage with graph structures and vector databases; (4) **awmPro**, our modified implementation of Agent Workflow Memory (Wang et al., 2024), which records and retrieves workflow histories to extract standardized execution paths; and (5) **MEMs**, our proposed multi-memory coordination framework. *For detailed formulations of all methods (Zero-shot, StreamICL, Mem0, awmPro, and MEMs), please refer to Appendix I.*

4.2 Limitations of Offline Evaluation and Value of Streaming Evaluation

Offline evaluation gives a misleading picture: as shown in Tables 6 and 7 (Appendix A), streamICL ranks first or second on most system and personal memory tasks, outperforming the dedicated method awmPro and mem0. Online evaluation (Table 3) reveals a similarly counter-intuitive result: despite having no memory refinement mechanism, streamICL maintains competitive CSR across DB, KG, and ALFWorld. Do these gains reflect genuine persistent memory accumulation, or merely the activation of LLM pretraining priors? Standard online metrics alone cannot answer this question—motivating the three-probe evaluation in Section 4.3.

Table 4: Cross-Environment Transfer Evaluation Results

Memory	DB→OS		OS→DB	
	ASR↑	AS↓	ASR	AS
zero-shot	36.8	2.52	47.3	2.73
streamICL	33.3	2.92	**49.0**	2.79
awmPro	**41.1**	**2.5**	**47.7**	**2.48**
mem0	35.4	**2.45**	47.0	**2.55**

ASR: Average Success Rate; AS: Average Step; ↑: higher is better

4.3 Distinguishing Persistent Memory from In-Context Learning: Multi-Dimensional Evaluation

To address the questions in RQ1, we design three probes to distinguish persistent memory methods from in-context learning baselines via Transfer, Repair, and forward-transfer modes. Detailed experimental setups appear in Appendix B.

Table 5: LoCoMo-0 Repair Evaluation Results (Normalized Metrics)

Memory	ER↑ Full	NR↑ Full	RG↑ Full	ER Std	NR Std	RG Std
streamICL	-44.74%	-36.84%	**+7.90%**	-38.02%	-32.40%	**+5.64%**
mem0	**+5.19%**	**+6.49%**	+1.30%	**+8.43%**	**+8.43%**	0.00%

ER: Error Robustness; NR: Net Recovery; RG: Repair Gain

Cross-environment transfer reveals generalization. We test whether methods learn transferable knowledge by training on one environment and testing on another (DB→OS and OS→DB). Table 4 shows that awmPro is the only method outperforming zero-shot in both directions, demonstrating genuine knowledge abstraction as a persistent memory method. streamICL and mem0 fail to transfer—their apparent online improvements stem from activating domain-specific LLM pretraining priors rather than accumulating reusable patterns.

Repair mode exposes catastrophic fragility. Table 5 shows starkly divergent behavior under erroneous feedback on LoCoMo-0: mem0 maintains positive performance after error injection (ER: +5.19%) and exceeds its baseline after correction (NR: +6.49%), demonstrating that its LLM-driven memory management can resolve knowledge conflicts. streamICL collapses catastrophically (ER: -44.74%) and never recovers (NR: -36.84%), because erroneous trajectories are indiscriminately appended to the vector store and cannot be selectively removed.

Forward transfer reveals long-term divergence. Within-environment forward transfer on LoCoMo-0 (Figure 4) confirms the asymmetry: mem0 forms a snowball effect—refined memory continuously improves future performance (rising FTG)—while streamICL degrades after 80 samples as retrieval quality collapses under memory bloat (falling FTG). Persistent memory methods refine knowledge over time; in-context learning baselines accumulate noise.

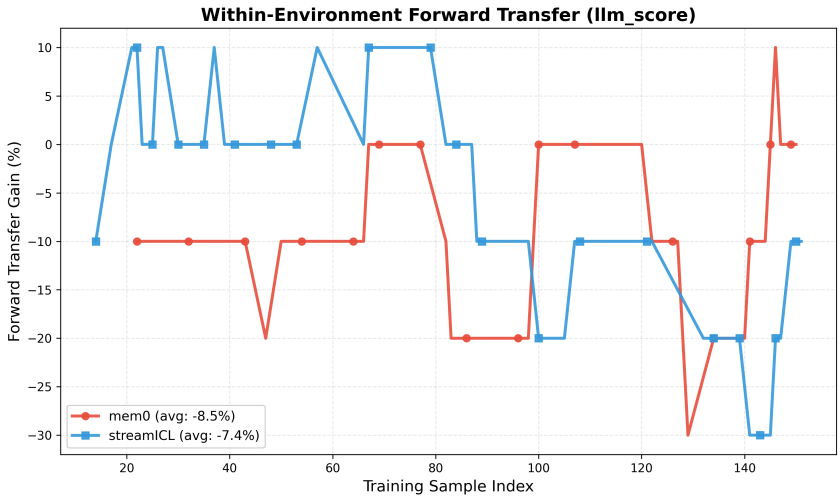

Figure 4: LoCoMo-0 within-environment forward transfer curves.

4.4 Multi-Memory Coordination in Scenarios with Fuzzy Task Boundaries

RQ1/RQ2 established that awmPro is a genuine persistent memory method for system tasks but fails on personal memory, while mem0 handles personal memory well but degrades to in-context learning behavior on system tasks. To test whether any single-memory method can handle both simultaneously, we construct a mixed task stream (DB+LoCoMo-4) where system and personal tasks alternate randomly. Detailed experimental setups are provided in Appendix C.

Figure 5 shows that all single-memory methods suffer architectural collapse during task switching: awmPro contaminates its workflow store with dialogue trajectories; mem0 misinterprets system task

execution as user preferences; streamICL fluctuates most severely due to absent memory management. MEMs resolves this through intelligent routing and memory isolation, maintaining stable upward trends throughout task switching.

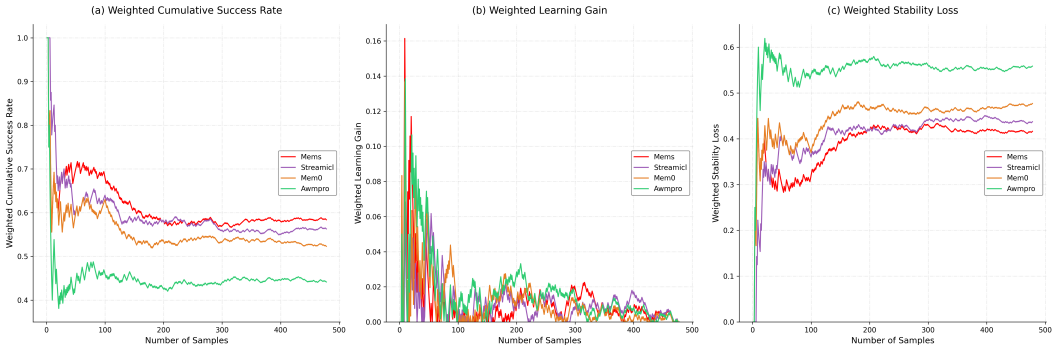

Figure 5: DB+LoCoMo-4 mixed task online learning curves. MEMs maintains stable upward trends while single-memory methods collapse at task boundaries.

5 Conclusion

We present AgentMemoryBench, the first benchmark to jointly evaluate system memory (a.k.a. experiential memory) and personal memory (a.k.a. factual memory) under a unified continual-learning framework with five complementary modes: offline, online, replay, transfer, and repair. Our experiments yield three interconnected findings. **First**, offline evaluation is systematically misleading: streamICL, an in-context learning baseline with no persistent memory management, rivals dedicated memory methods under static metrics— masking the underlying mechanism. **Second**, multi-dimensional probes expose a clear boundary: awmPro demonstrates genuine knowledge abstraction through successful cross-environment transfer, and mem0's LLM-driven management enables error recovery under the repair probe; by contrast, streamICL fails both probes, confirming its gains stem from activating pretraining priors rather than accumulating reusable knowledge. **Third**, no single-memory method survives mixed system–personal task streams: awmPro contaminates its workflow store with dialogue trajectories, mem0 misroutes system tasks as user preferences, and streamICL fluctuates most severely without memory management. MEMs resolves architectural collapse through a lightweight trigger model that routes retrieval and updates to specialized stores, maintaining stable learning curves throughout task switching.

These findings highlight two gaps that prior work has left unaddressed: the inability of static metrics to reveal learning dynamics, and the inadequacy of single-memory architectures in boundary-blurred real-world streams. We release AgentMemoryBench and MEMs as open resources to support reproducible, long-horizon agent memory research, and hope this work catalyzes standardized evaluation of persistent memory in increasingly capable agent systems.

References

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. Memorybench: A benchmark for memory and continual learning in llm systems, 2025. URL https://arxiv.org/abs/2510.17281.

Ding Chen, Simin Niu, Kehang Li, Peng Liu, Xiangping Zheng, Bo Tang, Xinchi Li, Feiyu Xiong, and Zhiyu Li. Halumem: Evaluating hallucinations in memory systems of agents, 2026. URL https://arxiv.org/abs/2511.03506.

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.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, Huaibing Xie, Jianglu Hu, Shaolei Wang, Weichao Wang, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Pluto Zhou, Tao Gui, Zuxuan Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. Cl-bench: A benchmark for context learning, 2026. URL https://arxiv.org/abs/2602.03587.

Howard Eichenbaum. Memory: Organization and control. *Annual Review of Psychology*, 68:19–45, jan 2017. doi: 10.1146/annurev-psych-010416-044131.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for llm agents through bayesian selection and contrastive refinement, 2025. URL https://arxiv.org/abs/2512.18950.

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, and Yafeng Deng. Evermemos: A self-organizing memory operating system for structured long-horizon reasoning, 2026a. URL https://arxiv.org/abs/2601.02163.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026b. URL https://arxiv.org/abs/2512.13564.

Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, Radha Poovendran, Gregory Wornell, Lyle Ungar, Dan Roth, Sihao Chen, and Camillo Jose Taylor. Personamem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory, 2025. URL https://arxiv.org/abs/2512.06688.

Hanqi Jiang, Junhao Chen, Yi Pan, Ling Chen, Weihang You, Yifan Zhou, Ruidong Zhang, Lin Zhao, Yohannes Abate, and Tianming Liu. Synapse: Empowering llm agents with episodic-semantic memory via spreading activation, 2026. URL https://arxiv.org/abs/2601.02744.

Cheng Jiayang, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. Amemgym: Interactive memory benchmarking for assistants in long-horizon conversations, 2026. URL https://arxiv.org/abs/2603.01966.

Han Li, Letian Zhu, Bohan Zhang, Rili Feng, Jiaming Wang, Yue Pan, Earl T. Barr, Federica Sarro, Zhaoyang Chu, and He Ye. Contextbench: A benchmark for context retrieval in coding agents, 2026. URL https://arxiv.org/abs/2602.05892.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. Memos: A memory os for ai system, 2025. URL https://arxiv.org/abs/2507.03724.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025. URL https://arxiv.org/abs/2308.03688.

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

Earl K. Miller and Jonathan D. Cohen. An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience*, 24:167–202, 2001. doi: 10.1146/annurev.neuro.24.1.167.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work?, 2022. URL https://arxiv.org/abs/2202.12837.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory, 2025. URL https://arxiv.org/abs/2509.25140.

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.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

Jerry W. Rudy, Joseph C. Biedenkapp, and Randall C. O'Reilly. Prefrontal cortex and the organization of recent and remote memories: an alternative view. *Learning & Memory*, 12(5):445–446, sep 2005. doi: 10.1101/lm.97905.

Yiting Shen, Kun Li, Wei Zhou, and Songlin Hu. Mem2actbench: A benchmark for evaluating long-term memory utilization in task-oriented autonomous agents, 2026. URL https://arxiv.org/abs/2601.19935.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2404.16789.

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.

Lukas Thede, Karsten Roth, Matthias Bethge, Zeynep Akata, and Tom Hartvigsen. Wikibigedit: Understanding the limits of lifelong knowledge editing in llms, 2025. URL https://arxiv.org/abs/2503.05683.

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

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. Larger language models do in-context learning differently, 2023. URL https://arxiv.org/abs/2303.03846.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2025. URL https://arxiv.org/abs/2511.20857.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung yi Lee. Streambench: Towards benchmarking continuous improvement of language agents, 2024. URL https://arxiv.org/abs/2406.08747.

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

Tingyu Wu, Zhisheng Chen, Ziyan Weng, Shuhe Wang, Chenglong Li, Shuo Zhang, Sen Hu, Silin Wu, Qizhen Lan, Huacan Wang, and Ronghao Chen. Knowme-bench: Benchmarking person understanding for lifelong digital companions, 2026. URL https://arxiv.org/abs/2601.04745.

Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From human memory to ai memory: A survey on memory mechanisms in the era of llms, 2025b. URL https://arxiv.org/abs/2504.15965.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023. URL https://arxiv.org/abs/2207.01206.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents, 2026. URL https://arxiv.org/abs/2601.01885.

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

Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, Siyuan Cheng, Ziwen Xu, Xin Xu, Jia-Chen Gu, Yong Jiang, Pengjun Xie, Fei Huang, Lei Liang, Zhiqiang Zhang, Xiaowei Zhu, Jun Zhou, and Huajun Chen. A comprehensive study of knowledge editing for large language models, 2024. URL https://arxiv.org/abs/2401.01286.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, 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, 2026a. URL https://arxiv.org/abs/2601.03192.

Zhe Zhang, Runlin Liu, Aishan Liu, Xingyu Liu, Xiang Gao, and Hailong Sun. Code2bench: Scaling source and rigor for dynamic benchmark construction, 2026b. URL https://arxiv.org/abs/2508.07180.

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.

Xiaoquan Zhi, Hongke Zhao, Likang Wu, Chuang Zhao, and Hengshu Zhu. Reinventing clinical dialogue: Agentic paradigms for llm enabled healthcare communication, 2025. URL https://arxiv.org/abs/2512.01453.

A EXPERIMENTAL DETAILS: LIMITATIONS OF OFFLINE EVALUATION AND VALUE OF STREAMING EVALUATION (RQ1)

Experimental Configuration: All experiments in RQ1 use the following unified configuration:

- **LLM Backbone**: Qwen3-14B
- **Random Seed**: 64, 65, and 66 (for reproducibility across all dataset shuffling and sampling operations)
- **Offline Mode Parameters**: Training samples constitute 60% of each dataset (train_size=0.6). For system memory tasks (DB, OS, KG, ALFWorld), samples are randomly shuffled with seed 66 and partitioned into train/test sets. For personal memory tasks (LoCoMo), all sessions serve as the training set to inject dialogue context into memory, and all questions serve as the test set to evaluate retrieval effectiveness.
- **Online Mode Parameters**: Sub-tasks are processed sequentially with memory retrieval (R operator) before each sub-task and memory update (F+E operators) after completion. Performance is recorded after each sub-task to form complete learning curves (CSR, LG, SL).

Surface Phenomena in Offline Mode: Tables 6 and 7 show the final performance of each method in traditional Offline mode, where different memory methods exhibit significant performance differences on system memory tasks (DB, OS, KG, ALFWorld) and personal memory tasks (LoCoMo-0).

Table 6: Offline Evaluation Results on System Memory Tasks (Except WebShop)

Memory	**DB**		**OS**		**KG**		**ALFWorld**	
	ASR↑	AS↓	ASR	AS	ASR	AS	ASR	AS
zero-shot	51.7	2.77	41.4	2.78	28.3	10.75	6.8	19.59
streamICL	**62.5**	**2.58**	43.1	3.28	**38.3**	10.77	**20.5**	**17.68**
awmPro	50	3.05	43.1	**2.47**	21.7	11.17	11.4	18.75
mem0	50	3.25	**50**	2.55	23.3	**10.6**	4.5	18.42

ASR: Average Success Rate; AS: Average Step

Table 7: Offline Evaluation Results on Personal Memory Task (only LoCoMo-0)

Memory	**single-hop**			**temporal**			**multi-hop**			**open-domain**		
	F↑	**B↑**	**J↑**	**F**	**B**	**J**	**F**	**B**	**J**	**F**	**B**	**J**
zero-shot	6.27	4.44	93.75	5.4	2.82	75.68	9.22	3.83	84.62	17.44	10.19	98.57
streamICL	**4.8**	**4.23**	53.12	6.09	**3.96**	**35.14**	6.55	**3.86**	69.23	13.17	8.47	57.14
awmPro	2.56	2.97	31.25	3.17	1.37	5.41	6.73	3.68	53.85	7.18	4.42	15.71
mem0	4.46	3.48	**56.25**	**6.25**	3.27	29.73	**7.27**	3.6	**92.31**	**15.26**	**8.66**	**71.43**

*In offline mode, all the sessions are provided to the zero-shot agent for LoCoMo tasks.

Table 8: Online Evaluation Results on System Memory Tasks (Except WebShop)

Memory	**DB**		**OS**		**KG**		**ALFWorld**	
	ASR↑	AS↓	ASR	AS	ASR	AS	ASR	AS
zero-shot	49	2.7	36.1	2.59	24.7	10.81	17.4	18.28
streamICL	58.7	2.65	24.3	4.36	30.7	11.65	32.1	**16.43**
awmPro	53	3.13	37.06	2.57	24.7	11.13	20.2	18.06
mem0	50.3	3.16	**38.9**	2.52	22.7	11.05	7.3	18.59
MEMs	**59.7**	**2.43**	37.76	**2.46**	**36**	**10.74**	**33.9**	16.75

ASR: Average Success Rate; AS: Average Step

Table 9: Online Evaluation Results on Personal Memory Task (average of LoCoMo-0&4)

Memory	single-hop			temporal			multi-hop			open-domain		
	F↑	B↑	J↑	F	B	J	F	B	J	F	B	J
zero-shot	9.52	7.28	87.35	5.45	3.61	74.17	7.63	4.30	73.9	21.09	13.54	95.55
streamICL	**10.28**	**9.19**	44.99	**6.11**	**4.11**	**42.40**	**9.99**	**6.99**	61.26	13.29	**8.78**	52.80
awmPro	5.62	5.43	28.56	2.7	2.11	14.77	7.40	4.64	48.63	7.6	4.69	32.69
mem0	6.46	5.60	49.40	4.81	3.37	31.6	7.48	4.43	78.3	13.29	8.26	54.99
MEMs	7.96	6.20	**60.39**	4.67	3.33	28.9	7.96	4.75	**75**	**13.62**	8.40	**59.94**

*In online mode, sessions are fed to the agent incrementally in temporal order; each session's QA pairs serve as sub-tasks processed sequentially.

Dynamic Revelation in Online Mode: We further evaluate each method in Online mode, recording complete learning curves. Figures 6, 7, 8, and 9 show the dynamic changes across three dimensions: Cumulative Success Rate (CSR), Learning Gain (LG), and Stability Loss (SL).

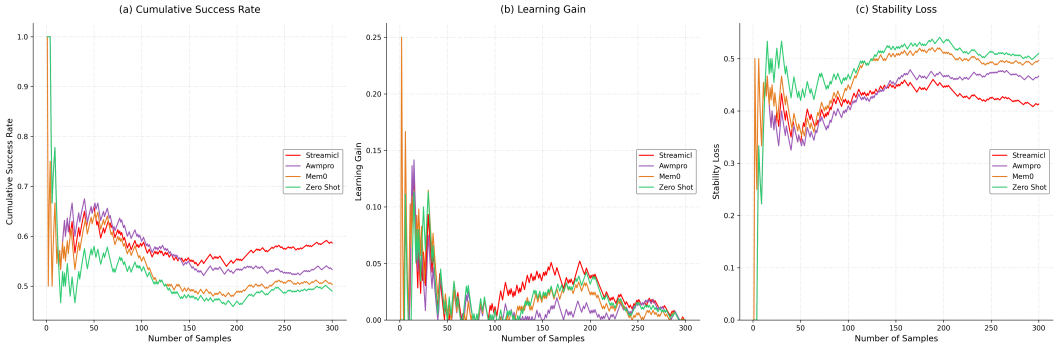

Figure 6: Online learning curves on DB task

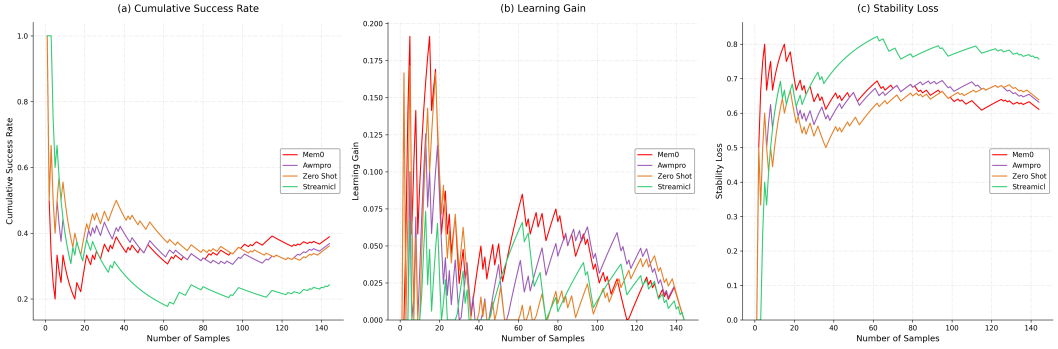

Figure 7: Online learning curves on OS task

Critical Questions Raised by Online Evaluation: From the Online learning curves, we observe several counter-intuitive phenomena that prompt deeper evaluation:

1. **Questions on System Memory Tasks**: On DB, OS, KG, and ALFWorld tasks, the learning curves of streamICL and mem0 appear to rise steadily, outperforming awmPro. However, these methods have obvious design flaws—streamICL only stores raw trajectories without memory refinement; mem0 incorrectly applies entity modeling to complete interaction trajectories (rather than focusing on user preferences). **Do their performance improvements come from persistent memory accumulation, or merely from activating semantic priors and format adaptation capabilities from LLM pretraining?** (Min et al., 2022; Wei et al., 2023)

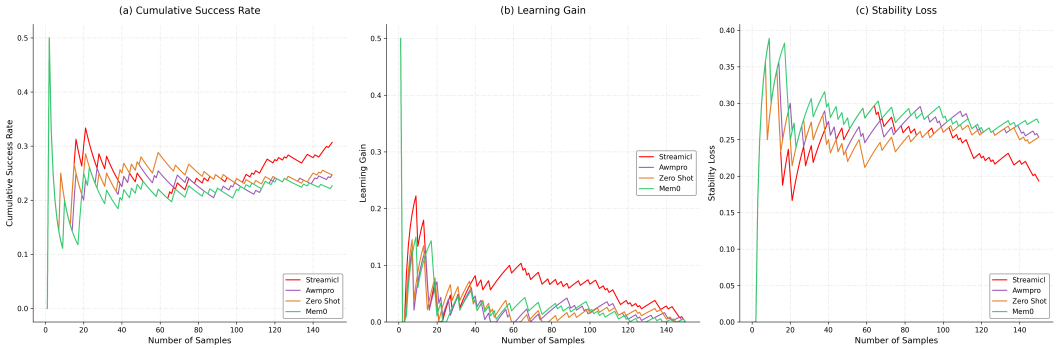

Figure 8: Online learning curves on KG task

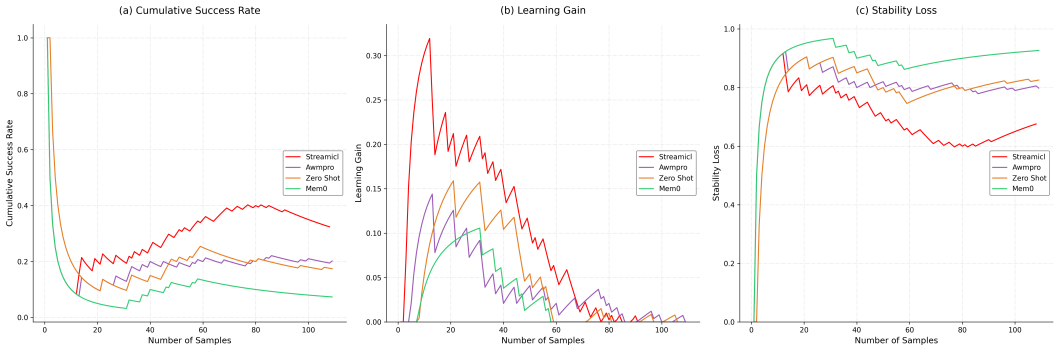

Figure 9: Online learning curves on ALFWorld task

2. **Questions on Personal Memory Tasks**: On LoCoMo tasks, streamICL's performance is comparable to the specially designed personal memory method mem0. **Is simple in-context learning truly sufficient for personal memory tasks? Or is Online evaluation still insufficient to distinguish persistent memory methods from in-context learning baselines?**

To answer these questions, we designed deeper evaluation experiments (Appendix B) to systematically distinguish persistent memory methods from in-context learning baselines through Replay, Transfer, and Repair modes.

B Experimental Details: Distinguishing Persistent Memory Methods from In-Context Learning Baselines (RQ2)

Experimental Configuration: All experiments in RQ2 use the following unified configuration:

- **LLM Backbone**: Qwen3-14B
- **Random Seed**: 64, 65, and 66 (for reproducibility across all dataset shuffling and sampling operations)
- **Replay Mode Parameters**: After every 20 learned samples (replay_m=20), we randomly sample 20 previously learned items (replay_n=20) for retrieval-only testing. Random sampling uses seed 66 (replay_seed=66) for reproducibility.
- **Transfer Mode Parameters**: For cross-environment transfer (DB→OS, OS→DB), agents first learn on the source task to build memory, then test on the target task using only retrieval. For within-environment forward transfer, we measure predictive improvement on samples 3 positions ahead (forward_transfer_num=3).

- **Repair Mode Parameters**: Samples are organized into repair groups with 20 samples per group (repair_m=20). Within each group, 20 samples receive inverted rewards (repair_n=20), selected with seed 66 (repair_seed=66). For LoCoMo tasks, 50% of QA pairs within each session receive inverted rewards (repair_size_locomo=0.5).

To systematically answer the critical questions raised in Appendix A, we designed three groups of deep evaluation experiments to reveal the essential differences between in-context learning baselines and persistent memory methods from four dimensions: **stability and generalization capability** (multi-task Online + cross-environment Transfer), **knowledge retention capability** (Replay), **robustness and repair capability** (Repair), and **forward transfer capability** (within-environment Transfer).

B.1 System Memory Tasks: Revealing streamICL and mem0 as In-Context Learning Baselines

Experimental Design Motivation: The Offline and Online results in Appendix A show that streamICL and mem0 outperform awmPro on system memory tasks. However, these methods have obvious design flaws—streamICL only stores raw trajectories, and mem0 incorrectly applies entity modeling to system task trajectories. To verify whether their performance improvements come from persistent memory accumulation, we designed the following experiments:

Experiment 1: Multi-Task Online Learning—Testing Stability

We conduct Online learning on a mixed DB+OS task stream, observing the learning curve stability of each method during task switching. As shown in Figure 10, streamICL exhibits significant performance regression on OS task samples, while mem0 and awmPro consistently maintain stable upward trends.

Key Finding: Performance regression indicates that streamICL's memory system lacks robustness to heterogeneous tasks—when task types switch frequently, simple similarity retrieval cannot stably provide effective memory, leading to performance fluctuations. This contrasts sharply with persistent memory methods: awmPro extracts reusable patterns through workflow abstraction, maintaining stable performance during task switching.

Experiment 2: Cross-Environment Transfer—Testing Generalization Capability

We further test two cross-environment transfer scenarios, DB→OS and OS→DB, to evaluate whether each method learns generalizable abstract knowledge. Table 12 in the main text shows that awmPro is the only method that outperforms zero-shot in both transfer directions.

Core Conclusion: Cross-environment transfer results reveal the essential deficiencies of streamICL and mem0 on system memory tasks—they rely on in-context learning rather than persistent memory accumulation. streamICL stores complete trajectories and uses similarity retrieval, essentially activating semantic priors and format adaptation capabilities from LLM pretraining (Min et al., 2022; Wei et al., 2023) rather than learning reusable task execution patterns; mem0 incorrectly treats system task trajectories as personal information for entity modeling, similarly unable to extract abstract knowledge. In contrast, awmPro extracts generalizable execution patterns through workflow induction, demonstrating genuine persistent memory capability in cross-environment transfer.

B.2 Personal Memory Tasks: Revealing the Shallow Memory Nature of streamICL

Experimental Design Motivation: The Offline results in Appendix A show that streamICL performs comparably to the specially designed mem0 on personal memory tasks (LoCoMo). To verify whether this "comparable performance" means streamICL truly possesses personal memory capability, we designed the following three experiments:

Experiment 3: Replay Learning—Testing Knowledge Retention Capability

We conduct Replay evaluation on LoCoMo-0, periodically testing the retention of learned samples. As shown in Table 10, mem0 and streamICL have comparable Average Success Rate (ASR) (53.94% vs. 55.92%) and both have low Forgetting Rate (FR) (6.67% vs. 1.67%).

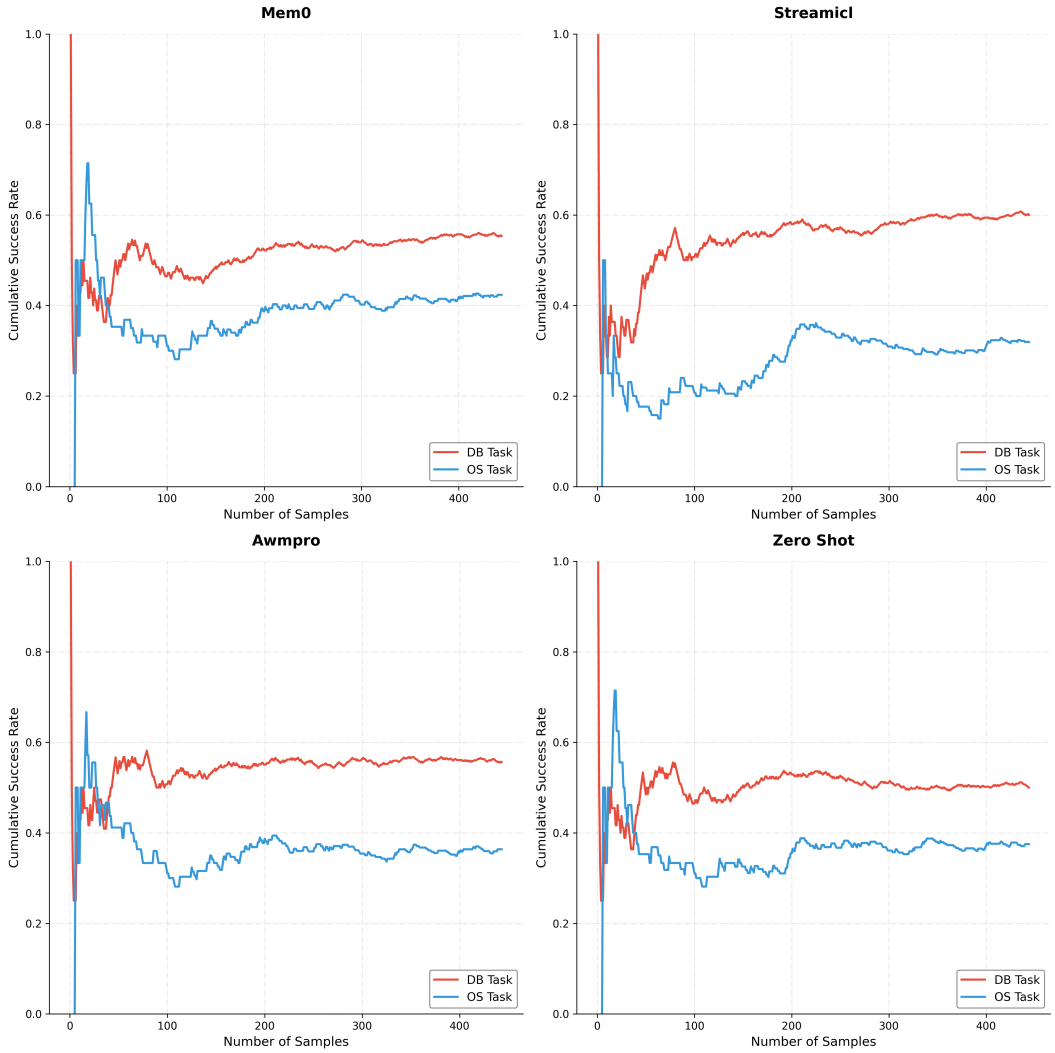

Figure 10: DB+OS multi-task Online learning curves.

Table 10: LoCoMo-0 Replay Evaluation Results

Memory	ASR↑	FR↓
streamICL	**55.92%**	**1.67%**
mem0	53.94%	6.67%

ASR: Average Success Rate; AS: Average Step

Initial Observation: Replay results seem to indicate that streamICL has good knowledge retention capability. However, this conclusion requires deeper evaluation to verify—does low forgetting rate mean genuine persistent memory, or merely mechanical storage of raw trajectories?

Experiment 4: Repair Learning—Testing Robustness and Repair Capability

We conduct Repair evaluation on LoCoMo-0, testing the memory system's resistance and repair capability under erroneous feedback. Table 11 in the main text shows that mem0 and streamICL exhibit starkly different performance in error robustness and net recovery.

Key Findings:

Table 11: LoCoMo-0 Repair Evaluation Results (Normalized Metrics)

Memory	ER↑ Full	NR↑ Full	RG↑ Full	ER Std	NR Std	RG Std
streamICL	-44.74%	-36.84%	**+7.90%**	-38.02%	-32.40%	**+5.64%**
mem0	**+5.19%**	**+6.49%**	+1.30%	**+8.43%**	**+8.43%**	0.00%

ER: Error Robustness; NR: Net Recovery; RG: Repair Gain

1. **Error Robustness**: mem0 maintains positive performance after erroneous learning (Full: +5.19%, Std: +8.43%), even slightly higher than the normal baseline; while streamICL's performance drops significantly after erroneous learning (Full: -44.74%, Std: -38.02%), indicating extreme fragility to erroneous feedback.
2. **Net Recovery**: mem0's post-repair performance even exceeds the normal baseline (Full: +6.49%, Std: +8.43%), indicating its memory management mechanism can identify and correct erroneous knowledge; while streamICL remains far below the normal baseline after repair (Full: -36.84%, Std: -32.40%), indicating erroneous knowledge has deeply contaminated the memory system and is difficult to fully repair.
3. **Repair Gain**: Although streamICL's repair gain appears higher (7.90% vs. 1.30%), this is because it drops severely after erroneous learning, leaving more room for repair; while mem0, due to strong error robustness, does not require significant repair.

Core Conclusion: The Repair experiment reveals a fatal flaw of streamICL—lacking a memory management mechanism, it cannot distinguish correct knowledge from erroneous knowledge and can only mechanically store all trajectories. When encountering erroneous feedback, erroneous trajectories are directly appended to the vector database, contaminating the entire memory system; during the repair phase, although correct trajectories are appended, erroneous trajectories still exist, causing retrieval to potentially recall both correct and erroneous memories, making performance difficult to fully recover. In contrast, mem0's LLM-driven memory management (ADD, UPDATE, DELETE, NONE) can identify knowledge conflicts and dynamically update memory, demonstrating genuine knowledge conflict resolution capability.

Experiment 5: Within-Environment Forward Transfer—Testing Knowledge Forward Transfer Capability

We conduct within-environment forward transfer evaluation on LoCoMo-0, testing the predictive improvement on future samples after learning current samples. Figure 11 (see main text Section 4.3) shows that mem0 and streamICL exhibit completely different forward transfer curves.

Key Findings:

1. **Early Stage (0-40 samples)**: mem0's Forward Transfer Gain (FTG) is relatively low, even slightly lower than streamICL. This is because the memory management mechanism requires time to accumulate and integrate knowledge in the early stages, with short-term effects inferior to directly storing complete trajectories.
2. **Mid-Stage Turning Point (40-130 samples)**: As memory accumulates, mem0's memory management system begins to take effect—removing redundant information through UPDATE and DELETE operations and integrating new knowledge through ADD operations, continuously improving memory quality, with the FTG curve beginning to rise.
3. **Late-Stage Divergence (130+ samples)**: mem0 forms a "snowball effect" similar to that mentioned in AWM (Wang et al., 2024)—high-quality refined memory makes new sample learning more efficient, with FTG continuing to rise; while streamICL, due to too many raw trajectories stored in the vector database, sees declining retrieval quality (increased noise), causing FTG to continuously regress after 80 samples.

Core Conclusion: The within-environment forward transfer experiment reveals the essential difference in long-term performance between in-context learning baselines and persistent memory methods. streamICL's mechanical storage strategy is effective in the short term (because few samples mean low retrieval noise), but as samples increase, lack of memory management leads to "memory

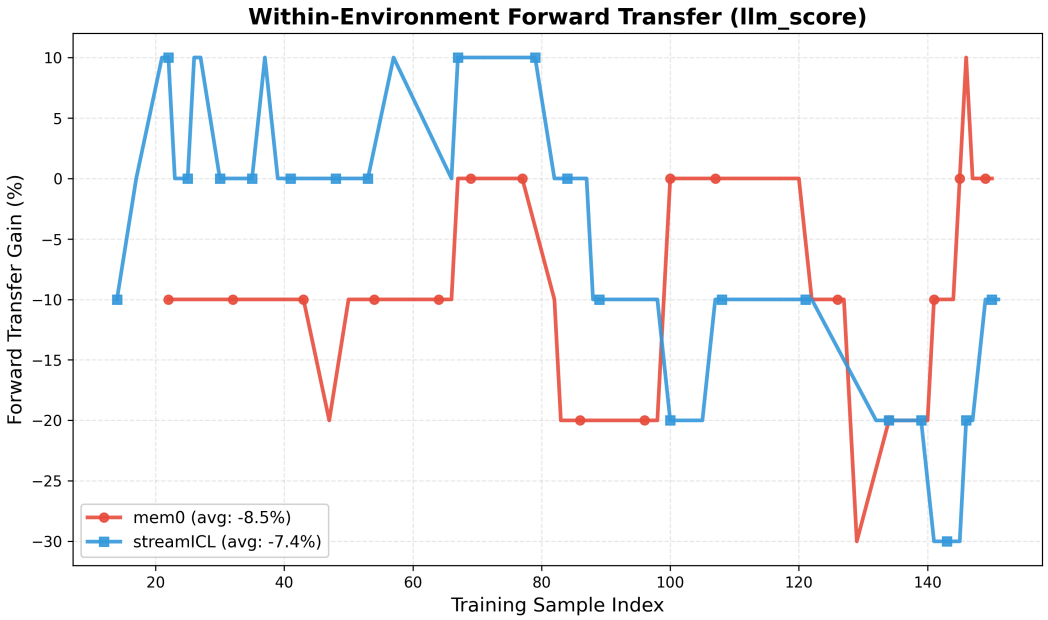

Figure 11: LoCoMo-0 within-environment forward transfer curves.

bloat"—the vector database fills with numerous redundant and low-quality trajectories, continuously deteriorating retrieval effectiveness. In contrast, mem0's memory management mechanism, through continuous refinement and integration, transforms raw trajectories into high-quality structured knowledge, demonstrating genuine forward transfer capability.

B.3 Comprehensive Conclusion: In-Context Learning Baselines vs. Persistent Memory Methods

Through three deep evaluation modes—Replay, Transfer (cross-environment + within-environment), and Repair—we systematically revealed the essential differences between in-context learning baselines and persistent memory methods:

StreamICL as an In-Context Learning Baseline:

- **System Memory Tasks**: Relies on "similarity retrieval of complete trajectories" to activate LLM pretraining priors rather than learning reusable patterns. Cross-environment transfer failure (Table 12) and performance regression in multi-task scenarios (Figure 10) prove its inability to extract abstract knowledge.
- **Personal Memory Tasks**: Relies on "mechanical storage of all trajectories" to achieve short-term effectiveness. Extremely low error robustness in Repair experiments (Table 11) and late-stage collapse in within-environment forward transfer (Figure 11) prove its lack of knowledge management capability, unable to handle knowledge conflicts and long-term accumulation.

The Persistent Memory Capability of Mem0 (limited to Personal Memory):

- Achieves knowledge conflict handling through LLM-driven memory management (ADD, UPDATE, DELETE), demonstrating extremely high error robustness and net recovery capability in Repair experiments (Table 11).
- Forms a "snowball effect" through continuous refinement and integration, with continuously improving late-stage performance in within-environment forward transfer (Figure 11).
- **But fails on System Memory tasks**: Incorrectly applies entity modeling to system task trajectories, with mediocre cross-environment transfer performance (Table 12).

Table 12: Cross-Environment Transfer Evaluation Results

Memory	DB→OS		OS→DB	
	ASR↑	AS↓	ASR	AS
zero-shot	36.8	2.52	47.3	2.73
streamICL	33.3	2.92	**49.0**	2.79
awmPro	**41.1**	**2.5**	**47.7**	**2.48**
mem0	35.4	**2.45**	47.0	**2.55**

ASR: Average Success Rate; AS: Average Step; ↑: higher is better

The Persistent Memory Capability of AwmPro (limited to System Memory):

- Extracts generalizable execution patterns through workflow induction, the only method comprehensively outperforming zero-shot in cross-environment transfer (Table 12).
- **But fails on Personal Memory tasks**: Lacks modeling of user preferences, with poor performance on LoCoMo tasks (Table 7).

Core Insight: A single memory mechanism may demonstrate persistent memory capability in its specialized domain but degrades to in-context learning behavior or even fails in non-specialized domains. This provides the theoretical foundation for the multi-memory coordination experiments in Appendix C.

C Experimental Details: Multi-Memory Coordination in Scenarios with Fuzzy Task Boundaries (RQ3)

Experimental Configuration: All experiments in RQ3 use the following unified configuration:

- **LLM Backbone**: Qwen3-14B
- **Random Seed**: 64, 65, and 66 (for reproducibility of task shuffling in mixed streams)
- **Mixed Task Stream Parameters**: System memory tasks (DB) and personal memory tasks (LoCoMo-4) are randomly interleaved with seed 66 to simulate fuzzy task boundaries. Sub-tasks are processed sequentially in Online mode with memory retrieval (R operator) before each sub-task and memory update (F+E operators) after completion.
- **Evaluation Metrics**: Cumulative Success Rate (CSR), Learning Gain (LG), and Stability Loss (SL) are recorded after each sub-task to form complete learning curves, capturing dynamic performance changes during task type switching.
- **Task Weights for Multi-task Aggregation**: Following the weight calculation method described in Appendix H.2 and based on the task statistics in Table 14, we compute the task weights for the DB+LoCoMo-4 mixed scenario using descending rank (higher rank value = higher priority):
  - **DB+LoCoMo-4**: DB has 300 samples (size rank 1, more) and 49.0% difficulty (diff rank 2, harder). LoCoMo-4 has 178 samples (size rank 2, fewer) and 56.18% difficulty (diff rank 1, easier). Combined ranks: $r_{\text{DB}} = 1 + 2 = 3$, $r_{\text{LoCoMo-4}} = 2 + 1 = 3$. Normalized weights: $w_{\text{DB}} = \frac{3}{3+3} = 0.5$ and $w_{\text{LoCoMo-4}} = \frac{3}{3+3} = 0.5$.

Experimental Motivation: The experiments in Appendix B revealed the fundamental limitations of single memory mechanisms—awmPro demonstrates persistent memory capability on system memory tasks but fails on personal memory tasks; mem0 performs excellently on personal memory tasks but degrades to in-context learning on system memory tasks. This phenomenon of "effective in specialized domains, ineffective in non-specialized domains" causes serious problems in real applications—real-world scenarios often have fuzzy task boundaries, with system tasks (such as code writing) and personal interactions (such as daily conversations) appearing randomly alternately.

Core Question: In real-world scenarios with fuzzy task boundaries and randomly switching task types, are single memory mechanisms still effective? Can multi-memory system coordination (such as MEMs) solve this problem?

Experimental Design: We designed a mixed task stream—**DB+LoCoMo-4**. In this mixed scenario, system memory tasks (DB) and personal memory tasks (LoCoMo-4) appear randomly alternately, precisely simulating the complex work scenarios in real applications where "task distributions are discrete and interaction processes are continuous."

Experimental Results: Figure 5 shows that MEMs significantly outperforms all single memory methods in the mixed scenario.

Key Findings:

1. **Performance Collapse of Single Memory Methods**:
   - **awmPro**: Performance drops significantly in mixed scenarios. When tasks switch from DB to LoCoMo, awmPro's workflow memory cannot provide effective support (because LoCoMo doesn't need workflows), causing performance to fall back to zero-shot level; worse, awmPro incorrectly induces LoCoMo dialogue trajectories as "workflows" and stores them, contaminating system memory and also affecting subsequent DB task performance.
   - **mem0**: Performs slightly better than awmPro in mixed scenarios but still has obvious problems. When tasks switch from LoCoMo to DB, mem0 incorrectly treats DB task trajectories as "user preferences" for entity modeling, causing memory quality to decline; simultaneously, erroneous memories from DB tasks also interfere with LoCoMo task retrieval.
   - **streamICL**: Shows the most performance fluctuation in mixed scenarios. Due to lack of memory management, the vector database simultaneously stores DB trajectories and LoCoMo trajectories, often recalling incorrect types of memories during retrieval (e.g., recalling LoCoMo trajectories during DB tasks), causing severe performance degradation.
2. **Multi-Memory Coordination Advantages of MEMs**:
   - **Intelligent Routing Mechanism**: The trigger model intelligently decides which memory system to call based on task features. In DB tasks, only system memory (workflows) is called; in LoCoMo tasks, only personal memory (user preferences) is called, avoiding memory contamination and retrieval noise.
   - **Memory Isolation and Specialization Maintenance**: System memory and personal memory are maintained separately, each maintaining high-quality memory in its specialized domain. DB task workflows don't contaminate personal memory, and LoCoMo dialogue history doesn't interfere with system memory.
   - **Stable Learning Curves**: MEMs maintains stable upward trends during task switching (Figure 5), with Cumulative Success Rate (CSR) reaching 58.7% in DB+LoCoMo-4, significantly outperforming the best single memory method (mem0: 51.8%).

Core Conclusion: Experimental results verify the core hypothesis we proposed in the Introduction—**in real-world scenarios with fuzzy task boundaries, single memory mechanisms cannot simultaneously meet the dual needs of system task execution and personal interaction; multi-memory system coordination is necessary to achieve true dynamic adaptability**. MEMs successfully solves the performance collapse problem of single memory methods in mixed scenarios through intelligent routing and memory isolation via a lightweight trigger model, providing an effective solution for building agent memory systems suitable for real-world applications.

D Experimental Details: Ablation Study on Memory Insertion Position (RQ4)

Experimental Configuration: All experiments in RQ4 use the following unified configuration:

- **LLM Backbone**: Qwen3-14B
- **Random Seed**: 64, 65, and 66 (for reproducibility across all dataset shuffling and sampling operations)
- **Evaluation Datasets**: DB, OS, and LoCoMo-0 tasks in Online mode

- **Insertion Strategies**: Two prompt insertion positions are compared: (1) In-between (System Prompt → Memory → User Question) and (2) Last (System Prompt → User Question → Memory)

Experimental Motivation: Where should retrieved memory be inserted in the prompt? Existing work typically inserts memory after the system prompt and before the user question (in-between), or after the user question (last). However, the impact of different insertion positions on performance has not been systematically studied.

Experimental Design: We compare two insertion strategies on three datasets—DB, OS, and AlfWorld:

- **In-between**: System Prompt → Memory → User Question
- **Last**: System Prompt → User Question → Memory

Experimental Results: As shown in Figure 12, the In-between strategy outperforms the Last strategy on all datasets.

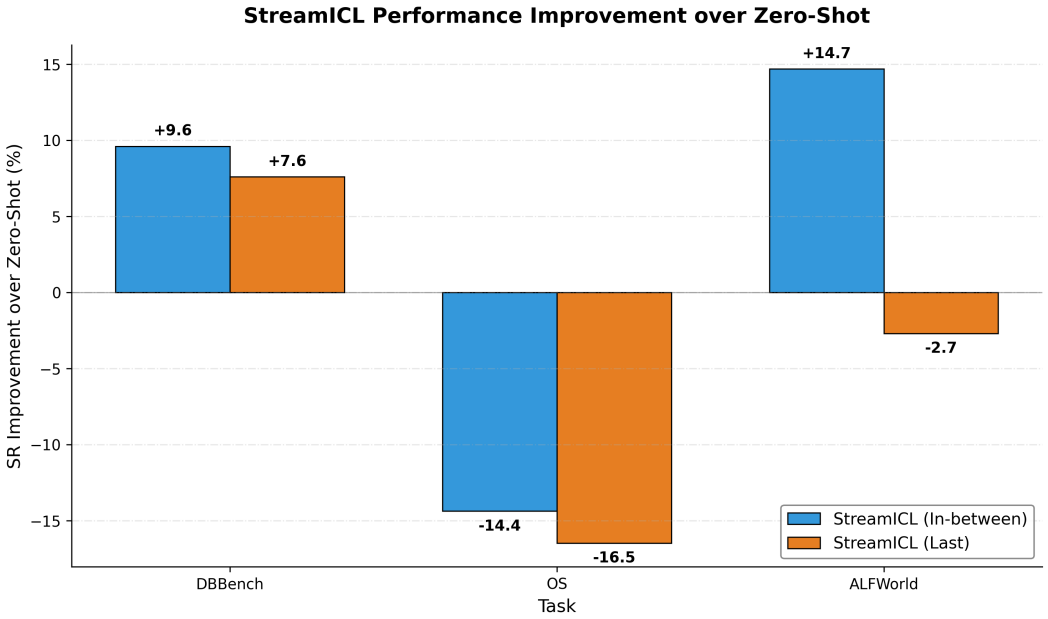

Figure 12: Comparison of memory insertion positions.

Explanation: The advantage of the In-between strategy lies in conforming to the LLM's attention mechanism—placing memory between the system prompt and user question allows the LLM to simultaneously attend to system instruction and memory context when processing the user question, forming a natural cognitive flow of "instruction → memory → question." In contrast, the Last strategy places memory at the end, requiring the LLM to "backtrack" to the previous question to understand the role of memory, increasing cognitive load and causing performance degradation.

Core Conclusion: Memory should be inserted between the system prompt and user question (in-between strategy) to maximize the enhancement effect of memory. This finding provides clear engineering guidance for the practical deployment of memory systems.

E Memory Benchmarks for Agents: Extended Survey

Related work on benchmarks for agent memory focuses on two primary dimensions: the evaluation paradigm and the categorization of memory types.

Streaming evaluation, introduced by StreamBench (Wu et al., 2024), is a paradigm where data and tasks are presented incrementally. In this setting, the model must complete learning and assess-

ment immediately upon receiving each batch of the stream, rather than undergoing centralized training followed by a unified test. This approach mirrors the sequential task evaluation commonly used in continual learning (CL) (Shi et al., 2024). The significance of streaming evaluation lies in its ability to quantify performance gains over time, memory retention, forgetting effects, generalization, and knowledge conflict resolution. However, a substantial portion of existing memory benchmarks still lacks support for streaming evaluation. While some benchmarks, such as LifelongAgentBench (Zheng et al., 2025) and Evo-Memory (Wei et al., 2025), have begun implementing stream-based assessments for system memory, they lack a comprehensive suite of continual learning metrics, such as backward transfer (forgetting rate) and forward transfer (Shi et al., 2024), as well as indicators for knowledge conflict resolution, such as the correction success rate used in knowledge editing (Thede et al., 2025).

Agent memory is further categorized into personal memory and system memory (Wu et al., 2025b). The former focuses on personalized user information, while the latter retains data related to task execution. System memory benchmarks evaluate an agent's ability to extract experience from past cases and internalize information over the long term, with the goal of increasing success rates in subsequent tasks. Conversely, personal memory benchmarks assess the ability to remember user preferences—such as dietary restrictions, schedules, and personal attributes—to enhance the interaction experience. Although some recent studies, such as MemoryBench (Ai et al., 2025), attempt to address both personal memory (referred to as declarative or factual memory) and system memory (referred to as procedural or experiential memory), they rely on non-streaming evaluation methods. This creates a gap in temporal assessment and fails to provide evaluation modes capable of distinguishing persistent memory methods from in-context learning baselines that merely accumulate context within the current window.

E.1 Non-stream Evaluation and Personal Memory Benchmarks

LOCOMO (Maharana et al., 2024) comprises multiple session-based and cross-session reasoning tasks designed to evaluate the long-term memory of agents within ultra-long dialogues. The dataset includes 10 extended conversations, each averaging approximately 300 turns and 9,000 tokens, requiring the agent to retain user preferences, key facts, and dialogue history across sessions with a focus on causal and temporal reasoning. Similarly, LongMemEval (Wu et al., 2025a) is tailored for multi-session user-assistant interactions, covering five core capabilities: information extraction, multi-session reasoning, temporal reasoning, knowledge updating, and abstention from unanswerable questions. DMR, introduced by MemGPT (Packer et al., 2024), focuses on the long-term memory consistency of conversational agents, specifically testing their ability to maintain and recall critical user facts across multiple chat sessions. In contrast to the precise extraction of distant information emphasized by the aforementioned benchmarks, PersonaMem-v2 (Jiang et al., 2025) focuses on the long-term memory of implicit user preferences and personalized adaptation. Utilizing 1,000 distinct user personas and over 300 daily scenarios, it constructs interaction histories up to 128k tokens. This benchmark requires agents to implicitly extract preferences from task-oriented dialogues, manage privacy protection, and handle "forgetting requests," thereby testing implicit preference reasoning and dynamic adaptation within long contexts.

These five benchmarks primarily employ accuracy and recall metrics—such as ROUGE-L, BLEU, and NDCG@k—alongside LLM-as-judge scoring. KnowMe-Bench (Wu et al., 2026) further adopts a hierarchical scoring system that evaluates entity accuracy at the factual level, reasoning validity at the logical level, and internal-external mapping precision at the insight level. All these frameworks are classified as personal memory benchmarks because their evaluation objectives center on the long-term retention and adaptation of user-specific information. Whether focusing on explicit facts (LOCOMO, DMR), implicit preferences (PersonaMem-v2), interaction details (LongMemEval), or internal motivations derived from autobiographical narratives (KnowMe-Bench), these efforts aim to align agent interactions with individual user needs. Consequently, they lack assessment of system-task execution data. Furthermore, these benchmarks do not support the streaming evaluation of personal memory and lack evaluation modes—such as replay and repair—that can distinguish persistent memory methods from in-context learning baselines like streamICL (Wu et al., 2024).

E.2 Stream Evaluation and Personal Memory Benchmarks

HaluMem (Chen et al., 2026) focuses on the operation-level hallucination assessment of memory systems, specifically targeting the full lifecycle of personal memory, including extraction, updating, and question-answering. It detects four categories of hallucinatory behaviors—fabrication, error, conflict, and omission—when a memory system processes personalized user information. Based on the HaluMem-Medium and HaluMem-Long datasets, which contain approximately 15,000 memory points and 3,500 QA pairs spanning a 10-to-20-year dialogue timeline, the benchmark emphasizes ensuring the reliability and consistency of personalized memory through fine-grained tasks. While HaluMem utilizes standard metrics such as accuracy, recall, and LLM-as-judge scoring, it introduces additional operation-level indicators, such as memory extraction accuracy and update hallucination rates. HaluMem is classified as a personal memory benchmark because its evaluation objectives and data are anchored in personalized user information; all memory points consist of user-specific identity traits, life events, and interpersonal relationships. Its primary goal is to guarantee the reliability of personalized memory by controlling hallucinations, thereby ensuring that agent interactions align closely with individual user needs. However, similar to other personal memory benchmarks, HaluMem lacks evaluation modes to distinguish persistent memory methods from in-context learning baselines, and it fails to provide comprehensive continual learning metrics or knowledge conflict resolution indicators to measure the quality of memory methods across the temporal dimension.

AMemGym (Jiayang et al., 2026) is an advanced interactive personal memory benchmark designed for on-policy evaluation of memory-driven personalization in long-horizon conversations. It addresses the limitations of static off-policy evaluation in traditional benchmarks by grounding free-form interactions in structured data: predefined user profiles (sampled from 100K personas), state evolution trajectories (capturing dynamic changes in user preferences, habits, and plans over multiple periods), and state-dependent evaluation questions. Leveraging LLM-simulated users to role-play with consistent structured states, AMemGym generates high-fidelity, scalable interaction data that reflects the true conversational consequences of an assistant's memory choices. Classified as a personal memory benchmark, its evaluation is centered on user-specific latent states (e.g., dietary preferences, schedule changes, life events) and their dynamic retention across extended dialogue turns. AMemGym introduces a comprehensive metric suite: overall question-answering accuracy (capturing personalization and memory integration), normalized memory scores (isolating memory performance from reasoning capabilities by normalizing against random and perfect-memory baselines), and diagnostic metrics (decomposing failures into write, read, and utilization errors to pinpoint memory bottlenecks). Additionally, it supports agent self-evolution by providing environmental feedback to refine memory management policies. Despite its innovations in interactive and dynamic evaluation, AMemGym shares similar limitations with HaluMem: it lacks evaluation modes to distinguish persistent memory methods from in-context learning baselines, and while it assesses temporal memory retention across conversation periods, it does not incorporate specialized continual learning metrics (e.g., backward transfer, knowledge conflict correction rates) for rigorous measurement of long-term memory quality over time.

E.3 Stream Evaluation and System Memory Benchmarks

StreamBench (Wu et al., 2024) is the inaugural benchmark focusing on the continuous improvement of LLM agents via streaming, evaluating their ability to learn from input-feedback sequences. It serializes static datasets into quasi-real-time data streams encompassing Text-to-SQL, Python programming, tool-use, medical diagnosis, and open-domain QA. Agents are required to receive task instances, generate outputs, and update external memory components immediately upon receiving binary feedback. However, its tasks are limited to single-turn interactions, whereas AgentMemoryBench supports multi-turn interaction sequences of up to 20 turns.

LifelongAgentBench (Zheng et al., 2025) is the first streaming benchmark specifically designed to assess the lifelong learning capabilities of LLM agents, focusing on skill reuse and transfer across three interactive environments: MySQL databases, Ubuntu operating systems, and SPARQL knowledge graphs. The benchmark constructs 22 SQL skill categories, 29 Bash command categories, and 7 KG query categories to evaluate how agents accumulate operational experience. Nevertheless, neither StreamBench nor LifelongAgentBench quantifies knowledge forgetting or backward transfer over time. Furthermore, they support only intra-environment transfer rather than cross-environment transfer or multi-task online learning—all of which are core features of AgentMemoryBench.

Evo-Memory (Wei et al., 2025) emphasizes the self-evolution of memory, evaluating the retrieval, synthesis, and dynamic updating of memory within multi-task streams. Similar to AgentMemoryBench, it supports both single-turn and multi-turn tasks (e.g., household instruction following and scientific experimentation), requiring agents to refine task-execution strategies through a "retrieve-synthesize-evolve" loop. While Evo-Memory supports multi-task online learning and cross-environment transfer, it lacks support for intra-environment transfer and replay learning modes. Crucially, it considers only a single memory type: system memory.

These three benchmarks are classified as system memory benchmarks as they focus exclusively on the retention and reuse of experience data generated during task execution to improve future success rates. They lack assessment of user preferences or personal attributes. While they utilize task-specific performance metrics (e.g., terminal Success Rate, average steps) and cumulative performance curves, they lack replay and repair evaluation modes that can distinguish persistent memory methods from in-context learning baselines. AgentMemoryBench addresses this gap by introducing a comprehensive suite of metrics, including backward transfer (forgetting rate), forward transfer (Shi et al., 2024), and knowledge conflict resolution indicators such as the correction success rate (Thede et al., 2025).

E.4 Non-stream Evaluation and Hybrid Memory Benchmarks

MemoryBench (Ai et al., 2025) is the first benchmark to explicitly cover both personal and system memory evaluation, breaking the limitation of single-memory assessment. Likely the classification in (Wu et al., 2025b), MemoryBench categorizes memory into declarative and procedural types. Declarative memory corresponds to personal memory, anchoring user-specific information such as dialogue history, preferences, and task contexts to test the agent's ability to optimize interaction experiences. Procedural memory aligns with system memory, evaluating the retention of general experience from task execution to improve future success rates. The benchmark integrates 11 cross-domain datasets spanning open-domain, legal, and academic fields, supporting four task formats: Long-in Short-out (LiSo), Short-in Long-out (SiLo), Long-in Long-out (LiLo), and Short-in Short-out (SiSo). For example, multi-turn dialogue understanding in datasets like DialSim and Locomo serves personal memory evaluation, while legal document generation in LexEval depends on system memory.

Despite these innovations, MemoryBench operates under a non-streaming evaluation paradigm. Training data and feedback logs (simulated via an LLM-as-user approach) are pre-generated, and the model undergoes offline or stepwise off-policy training followed by a one-time assessment on a disjoint test set. Its evaluation metrics rely on dataset-native scores (e.g., F1, ROUGE-L) consolidated into a single 1–10 score via LLM-as-Judge, lacking temporal indicators unique to continual learning. Consequently, MemoryBench exhibits two primary deficiencies: first, the absence of streaming evaluation precludes dynamic assessment of memory retention, forgetting effects, and knowledge transfer; second, it lacks evaluation modes such as replay and repair that can distinguish persistent memory methods from in-context learning baselines like streamICL (Wu et al., 2024).

E.5 Context-Learning Benchmarks

Context-learning benchmarks evaluate how well a model acquires and applies new knowledge from within a single session's context window, without relying on pre-trained internal knowledge. This is fundamentally different from AgentMemoryBench's focus: we evaluate how agents accumulate, retain, and transfer knowledge across sessions through persistent external memory. While Mem2ActBench (Shen et al., 2026), CL-bench (Dou et al., 2026), CONTEXTBENCH (Li et al., 2026), and CODE2BENCH (Zhang et al., 2026b) each advance single-session context utilization, none addresses the cross-session persistent memory evolution that real-world deployments require.

Mem2ActBench: Focused on Memory-Driven Grounding, but Limited to Retrieval. Mem2ActBench designs tasks where the model must extract "new information" (e.g., user preferences or task constraints) from fragmented, long-term dialogue chains and ground them as parameters for tool calls (e.g., budget or flight preferences). While it functions as a context-learning benchmark, its limitations are twofold: (i) *Metric Deficiency*: It relies on task completion metrics like Tool Accuracy (TA), parameter F1, and BLEU-1. It cannot distinguish whether a model has truly understood the relationship between parameters and user needs or is simply performing

"pseudo-learning" via mechanical string matching; (ii) *Static Paradigm*: It does not support streaming inputs or iterative memory optimization, failing to simulate the incremental accumulation of knowledge found in real-world interactions.

CL-bench: Comprehensive Evaluation of Complex In-Context Learning. CL-bench stands as a representative benchmark in this field, focusing on how models learn from complex, uncontaminated contexts such as new domain knowledge or rule systems. It provides a robust answer to the "true learning vs. mechanical patching" problem through multi-dimensional metrics: (i) *Error Distribution*: Categorizes failures into context-ignorance, misuse, or formatting errors to locate the root of the learning failure. (ii) *Sensitivity and Effort*: Evaluates stability across context lengths (up to 65,000 tokens) and compares performance across different reasoning effort levels to verify deep comprehension. (iii) *Sequential Dependency*: With over 50% of tasks requiring results from previous steps, it forces models to demonstrate cross-step knowledge reuse.

CONTEXTBENCH: Process-Oriented Evaluation of Code Context Retrieval. As a code-specific context-learning benchmark, CONTEXTBENCH targets the "black-box" limitation of outcome-driven evaluations by focusing on the process of context retrieval in coding agents. It effectively distinguishes genuine context learning from mechanical exploration through two core designs: (i) *Gold Context Anchoring*: Expert annotators label the critical code regions (gold contexts) necessary for resolving each issue, establishing a precise benchmark for evaluating retrieval relevance; (ii) *Fine-Grained Process Metrics*: By tracking the agent's full retrieval trajectory, it computes recall, precision, and F1 scores at three granularities (file, block, line), complemented by process dynamics metrics (efficiency, redundancy, usage drop). These metrics expose whether the agent genuinely retrieves and utilizes key context or merely achieves task success through blind trial-and-error, thus solving the "true learning vs. mechanical patching" dilemma in code-related context learning.

CODE2BENCH: Rigorous Validation of Code Generation Generalization. Focused on code-generating LLMs, CODE2BENCH addresses the core challenge of distinguishing "memorization vs. true generalization" through a "Dual Scaling" philosophy: (i) *Dynamic Contamination-Resistant Sourcing*: It extracts functions exclusively from GitHub commits post-dating the evaluated models' knowledge cutoff, ensuring tasks are provably unseen and eliminating memorization of pre-trained data; (ii) *High-Rigor Property-Based Testing (PBT)*: Instead of relying on limited example-based tests, it generates hundreds of structured inputs (including edge cases) and enforces 100% branch coverage, exposing "near-perfect failures" (solutions passing 98% of tests but failing subtle edge cases) that conventional benchmarks miss. Additionally, it classifies tasks into Self-Contained (SC, pure algorithmic reasoning) and Weakly Self-Contained (WSC, API application), enabling targeted evaluation of distinct learning capabilities. Together, these designs ensure that the benchmark measures true code generation generalization (genuine learning) rather than mechanical pattern matching or memorization.

The Cost of Missing Hybrid Evaluation. The absence of a hybrid memory benchmark is not merely an evaluation gap—it has concrete consequences for the field. Methods that attempt to bridge system and personal memory, such as MemGen (Zhang et al., 2025), which unifies heterogeneous memory representations via machine-native latent token sequences, cannot be rigorously validated: no existing benchmark supports the mixed-scenario streams in which such coordination is actually needed. Without a testbed that interleaves system tasks and personal interactions with blurred boundaries, the cross-type coordination capability of general memory approaches remains unmeasured. AgentMemoryBench is designed to close this gap.

E.6 Annotation Guide for Table 1

Table 1 evaluates nine benchmarks across 13 dimensions using three symbols: ✓ (fully supported), × (not supported), and □ (not applicable by design). We explain each dimension and the rationale for □ assignments below. eftab:benchmark-comparison evaluates nine benchmarks across 13 dimensions using three symbols: ✓ (fully supported), $imes$ (not supported), and □ (not applicable by design). We explain each dimension and the rationale for □ assignments below.

Memory Eval. Whether the benchmark targets *persistent external memory*—storage, retrieval, and update of knowledge across sessions or sub-tasks. ✓ denotes a memory benchmark; × denotes a

context-learning benchmark, which evaluates within-session context utilization rather than persistent memory.

CL Metrics. Whether the benchmark reports continual learning metrics beyond terminal task performance, specifically Forgetting Rate (backward transfer) and/or Forward Transfer Gain. □ is assigned to benchmarks that operate under purely offline or single-session paradigms—including both offline memory benchmarks (e.g., LoCoMo, MemoryBench) and context-learning benchmarks—where temporal dynamics are absent by design and such metrics are structurally inapplicable. **Distinguish Learn.** Whether the evaluation protocol can distinguish persistent memory methods from in-context learning baselines. For memory benchmarks: ✓ denotes full distinguishability via transfer, replay, and repair probes; × denotes no distinguishing capability (offline-only or no multi-dimensional probes). For context-learning benchmarks: ✓ denotes the ability to distinguish genuine context learning from mechanical pattern matching within a session (e.g., CL-bench); × denotes no such distinguishing capability.

Hybrid Memory. Whether the benchmark simultaneously evaluates both system memory (experiential/procedural, related to task execution) and personal memory (factual/declarative, related to user preferences). □ is assigned to context-learning benchmarks for which this categorization is not applicable.

Multi-turn. Whether individual *sub-tasks* require multiple rounds of agent–environment interaction to reach completion—for example, iterative tool invocations across steps in database or operating-system tasks. Single-turn question answering (e.g., LoCoMo sub-tasks each consist of answering one question) does not qualify, regardless of whether the overall benchmark spans many sessions.

Tool-use. Whether sub-tasks require the agent to invoke external tools, such as code execution engines, SQL databases, or web APIs.

Single-task Online and **Multi-task Online.** Whether the benchmark supports streaming evaluation within a single task type or across interleaved task types respectively, where the agent processes sub-tasks sequentially and updates memory in real time. □ is assigned to context-learning benchmarks, for which online streaming across sessions is structurally inapplicable.

Offline. Whether the benchmark supports static train-test split evaluation, where memory is built from a training set and assessed on a disjoint test set.

Intra-env Transfer, **Cross-env Transfer**, **Replay**, and **Repair** correspond directly to AgentMemoryBench's four advanced evaluation modes (Section H). □ is assigned to benchmarks that do not support online evaluation (as all four modes require temporal learning dynamics) and to context-learning benchmarks. × is assigned to online memory benchmarks that lack these specific modes.

F Memory Methodologies for Agents: Extended Survey

Memory mechanisms can be categorized into two primary classes based on their functional focus: system memory methods and personal memory methods.

System memory methods focus on extracting reusable knowledge from task execution experiences. SYNAPSE (Jiang et al., 2026) integrates episodic memory from interactions with abstract semantic concepts by constructing a unified episodic-semantic graph and a spreading activation mechanism, which dynamically highlights relevant knowledge subgraphs to enhance reliability in complex temporal and multi-hop reasoning tasks. AgeMem (Yu et al., 2026) utilizes a unified tool-like interface and a three-stage progressive reinforcement learning strategy, enabling large language models (LLMs) to autonomously perform operations such as storage, retrieval, and updating of long-term and short-term memory. This approach learns collaborative management strategies from multi-stage task experiences to improve adaptation and context utilization in long-term reasoning. MACLA (Forouzandeh et al., 2025) builds an external hierarchical program memory that tracks program reliability through Bayesian selection and refines preconditions and action sequences via contrastive learning. It extracts interpretable reusable programs from massive trajectories to improve sample efficiency and generalization in long-horizon tasks. Agent Workflow Memory (AWM) (Wang et al., 2024) records and retrieves complete workflow histories to extract standardized execution paths and key nodes from repetitive tasks, reducing redundant decision steps and improving consistency by solidifying procedural experiences into reusable templates. Reasoning-

Bank (Ouyang et al., 2025) focuses on distilling generalizable high-level reasoning strategies from both successful and failed task experiences to construct a specialized reasoning memory framework. MemRL (Zhang et al., 2026a) applies non-parametric reinforcement learning to episodic memory to achieve self-evolution based on runtime environmental feedback. By decoupling the stable reasoning of a frozen LLM from the plasticity of external memory, it distinguishes high-value strategies from semantic noise. It constructs a structured memory bank of intent-experience-utility (Q-value) triplets and employs a two-stage retrieval mechanism involving semantic similarity recall and value-aware selection, while updating memory utility via Bellman updates during runtime.

These methods share the common characteristic of centering on task execution experience, transforming it into reusable knowledge through structural modeling, abstraction, and intelligent retrieval. However, they generally assume clear task boundaries and explicit task types. In hybrid scenarios where personal and system memory tasks are intertwined, these methods struggle with flexible memory scheduling and lack the dynamic adaptability required for fuzzy task boundaries.

Personal memory methods prioritize user preference retention and context understanding during long-term dialogues. Mem0 (Chhikara et al., 2025) provides a scalable long-term memory system that dynamically captures and integrates key information through an extraction-update pipeline, supporting graph-structured storage to model complex semantic relationships and maintain consistency in cross-session reasoning. MemGPT (Packer et al., 2024) adopts an operating-system-inspired hierarchical memory architecture, partitioning memory into a main context and an external context. It implements information paging via function calls to efficiently manage user facts and dialogue history, supporting long-document analysis and multi-session interactions. ZEP (Rasmussen et al., 2025) builds a memory layer service based on temporal knowledge graphs, using a hierarchical structure of episodic, semantic entity, and community subgraphs to dynamically integrate unstructured conversation data with structured business data. MemOS (Li et al., 2025) introduces a memory operating system to manage heterogeneous memory types, including parametric, active, and plaintext memory, using MemCube as the core unit for lifecycle scheduling and cross-platform migration. EverMemOS (Hu et al., 2026a) constructs a self-organizing memory system that mimics the biological memory cycle through stages of MemCell generation, semantic integration into MemScenes, and reconstructive recall. While effective in their respective domains, these five methods are restricted to the personal memory dimension and cannot support the transfer of task execution experiences. Evaluation of these methods typically relies on personal memory benchmarks such as Locomo (Maharana et al., 2024), LongMemEval (Wu et al., 2025a), and DMR (Packer et al., 2024), which focus on user preferences and temporal reasoning but fail to assess performance in system task execution or hybrid scenarios.

Taken together, system memory methods excel within task-execution domains but assume clear task boundaries and cannot adapt to user-specific context; personal memory methods model user preferences effectively but cannot generalize procedural experience across tasks. Neither type alone is sufficient when system tasks and personal interactions are interleaved—the precise condition that real-world deployments impose. MEMs addresses this by retaining both specialized stores and introducing a lightweight trigger model as a coordination layer, enabling each store to remain structurally optimized for its domain while the router decides when and where to retrieve and update.

G Dataset Details

This appendix provides detailed specifications for the six datasets in AgentMemoryBench, including dataset statistics, construction protocols, and task-level characteristics. For a summary overview of all datasets, see Table 2 in the main text.

G.1 System Memory Tasks

Code-Grounded Environments:

- **Database (DB)** (Liu et al., 2025): Evaluates SQL database manipulation skills. Agents interact with SQLite databases to execute queries, insert/update records, and perform data analysis. Each sub-task requires deriving correct SQL commands from natural language instructions. Dataset contains 300 sub-tasks requiring multi-turn code execution.

- **Operating System (OS)** (Liu et al., 2025): Assesses terminal shell command execution capabilities. Agents navigate file systems, manage processes, and manipulate system configurations through bash commands. Tasks require learning command-line patterns and system operation strategies. Dataset contains 144 sub-tasks.
- **Knowledge Graph (KG)** (Liu et al., 2025): Examines reasoning and querying over large-scale knowledge graphs. Agents construct SPARQL queries or traversal algorithms to extract multi-hop relational information. Tasks demand understanding graph structure and query optimization. Dataset contains 150 sub-tasks.

Embodied Environment:

- **ALFWorld** (Shridhar et al., 2021): Focuses on commonsense reasoning and task execution within embodied household scenarios. Agents perform tasks such as "place a clean pan on the dining table," requiring spatial reasoning and action planning. Tasks evaluate behavioral pattern learning from historical actions. Dataset contains 109 sub-tasks spanning diverse household objectives.

Web-Grounded Environment:

- **WebShop** (Yao et al., 2023): Simulates online purchasing workflows in e-commerce scenarios. Agents search for products matching user requirements, compare options, and complete purchases. Tasks require summarizing search strategies and selection logic from past shopping experiences. Dataset contains 200 sub-tasks.

G.2 Personal Memory Task

Dialogue-Grounded Environment:

- **LoCoMo** (Maharana et al., 2024): Emphasizes long-term conversational memory retention. The dataset comprises 10 extended dialogues (locomo-0 through locomo-9), each averaging 20 sessions and 200 questions. Sessions contain background information (user preferences, context, key facts) that must be retained across turns. Each question serves as a sub-task requiring retrieval of relevant session information to generate accurate responses. Total: 10 dialogues $\times$ $\sim$200 QA pairs = $\sim$2000 sub-tasks across all dialogues.

G.3 Task-Level Characteristics

Task vs. Sub-task Definition: Each dataset corresponds to a *task* (e.g., DB task = DB dataset). Each task contains multiple *sub-tasks*—individual samples requiring single-turn or multi-turn agent-environment interactions. For system memory tasks, each sub-task is a standalone problem instance. For LoCoMo, each question within a dialogue constitutes a sub-task, while sessions provide the memory context.

Multi-turn Interaction: System memory sub-tasks (DB, OS, KG, ALFWorld, WebShop) require iterative exchanges between agents and environments. Agents must invoke tools (e.g., SQL executor, bash shell), observe feedback, and refine actions until task completion or maximum steps are reached. Personal memory sub-tasks (LoCoMo) are single-turn: the agent retrieves relevant memory and generates a response to each question in one step.

Evaluation Metrics: System memory tasks use Average Success Rate (ASR) and Average Steps (AS) computed across all sub-tasks. Personal memory task (LoCoMo) uses F1-score (measuring answer overlap with gold references), BLEU-score (evaluating fluency), and LLM-as-Judge (assessing semantic correctness via Qwen2.5-7B-Instruct evaluation).

Sampling and Reproducibility: All datasets support seeded random shuffling to enable reproducible evaluation across five modes (offline, online, replay, transfer, repair). For LoCoMo tasks, session-based scheduling preserves temporal ordering across sessions while allowing intra-session question shuffling.

H EVALUATION MODES: DETAILED SPECIFICATIONS

This appendix provides detailed implementation specifications and metric formulations for each of the five evaluation modes in AgentMemoryBench. Table 13 summarizes all evaluation modes and their metrics.

H.1 OFFLINE MODE

Offline Mode represents a traditional non-streaming evaluation paradigm that bifurcates the sample set into distinct training and testing phases. The agent first executes all sub-tasks within the training set to update its memory—invoking only the memory formation (F) and memory evolution (E) operators—without recording performance metrics. Subsequently, the agent performs tasks in the testing set by exclusively retrieving memory—invoking only the memory retrieval (R) operator—to assess terminal performance.

For system memory tasks (e.g., DB, OS, KG, ALFWorld, WebShop), samples are shuffled and partitioned according to a predefined ratio (e.g., $train\_size = 0.6$). For personal memory tasks (e.g., LoCoMo), we adopt the configuration established by Mem0 (Chhikara et al., 2025), utilizing all sessions as the training set to inject dialogue context into memory and all questions as the test set to evaluate retrieval effectiveness.

Evaluation Metrics:

- **System Memory Tasks**: Effectiveness is measured by performance on the test set:
  - Average Success Rate (ASR): The task success rate on the test set, measuring the capacity of the memory system to facilitate task completion;
  - Average Step (AS): The average number of execution steps on the test set, measuring the efficiency gains provided by the memory system.
- **Personal Memory Tasks**: Retrieval quality is assessed via multi-dimensional metrics:
  - F1-score: The harmonic mean of precision and recall for the generated answers;
  - BLEU-score: The n-gram overlap between generated responses and ground-truth references;
  - LLM as Judge: Utilizing large language models to evaluate the semantic accuracy and completeness of the answers.

While this mode effectively evaluates the terminal performance of a memory system, it remains static and cannot capture dynamic transitions during the learning process.

H.2 ONLINE MODE

Online Mode (Streaming Evaluation) adopts a streaming learning paradigm where the agent updates its memory in real-time during the execution of a sub-task sequence. By recording performance variations after each sub-task, we observe the dynamic processes of knowledge accumulation, restructuring, and forgetting. Given a task sequence $\mathcal{T} = \{k_1, k_2, \ldots, k_N\}$, memory is retrieved (invoking the R operator) before each sub-task k_i and updated (invoking the F and E operators) upon completion:

$$M^0 \xrightarrow[\text{R+F+E}]{k_1} M^1 \xrightarrow[\text{R+F+E}]{k_2} M^2 \cdots \xrightarrow[\text{R+F+E}]{k_N} M^N$$

This mode captures the complete learning curve of the agent, representing a core innovation of AgentMemoryBench. Furthermore, we support multi-task online learning and reproducible task shuffling (shuffle with seed). During evaluation, task types can alternate randomly—mixing heterogeneous tasks such as SQL generation (DB), shell command execution (OS), and daily dialogue (LoCoMo). This accurately simulates complex real-world scenarios where task boundaries are blurred and task distributions shift continuously, ensuring that methods performing well on this benchmark remain effective in practical deployments, as shown in Figure 2.

Table 13: Summary of Evaluation Modes and Performance Metrics

	Offline	Online	Replay	Transfer	Repair
Metrics	**Sys**: ASR, AS **Pers**: F1, BLEU, LLM as J	**Single**: CSR, LG, SL **Multi**: WCSR, WLG, WSL	ASR, FR	**Cross**: TG **Intra**: FTG	RG, ER, NR (Full & Std)

Evaluation Metrics: The Online mode utilizes three core metrics to quantify learning dynamics across single-task and multi-task scenarios:

1. Single-task Online Learning:

- **Cumulative Success Rate (CSR)**: We record the instantaneous success rate P_i after each sample k_i to form a learning curve $\{P_1, P_2, \ldots, P_N\}$. An upward trend indicates effective learning, while fluctuations or declines suggest forgetting or knowledge conflicts.
- **Learning Gain (LG)**: This measures the potential improvement space of current performance relative to future peak performance. At time step t, it is defined as:

$$\text{LG}_t = P_{\max}^{\text{future}} - P_t$$

  where P_t is the average success rate after t samples and $P_{\max}^{\text{future}} = \max_{i \geq t} P_i$ is the future maximum success rate from time step t onwards (including current). $\text{LG}_t = 0$ indicates current performance has reached the future optimum (no room for improvement), $\text{LG}_t > 0$ indicates remaining improvement potential, and smaller LG_t indicates the learning system is approaching its performance ceiling.
- **Stability Loss (SL)**: This quantifies the degradation from the historical peak to measure the forgetting effect. At time step t, it is defined as:

$$\text{SL}_t = P_{\max}^{t} - P_t$$

  where $P_{\max}^{t} = \max_{i \leq t} P_i$ is the historical maximum success rate. A larger SL_t signifies more severe forgetting.

2. Multi-task Online Learning: For multi-task scenarios, we aggregate performance across different tasks using task weights $w_{\mathcal{T}}$ to ensure fairness and comparability.

Task Weight Calculation: To ensure balanced contribution across heterogeneous tasks, we compute task weights by comprehensively considering both quantity and difficulty dimensions using descending order ranking (higher rank value indicates higher priority). Given the task set $\mathcal{D}$, for each task $\mathcal{T} \in \mathcal{D}$:

1. Assign quantity rank $r_{\text{size}}(\mathcal{T})$: tasks with fewer samples receive higher rank values.
2. Assign difficulty rank $r_{\text{diff}}(\mathcal{T})$: tasks with lower baseline success rates (harder tasks) receive higher rank values.
3. Calculate the combined rank: $r(\mathcal{T}) = r_{\text{size}}(\mathcal{T}) + r_{\text{diff}}(\mathcal{T})$.
4. Normalize to obtain task weights:

$$w_{\mathcal{T}} = \frac{r(\mathcal{T})}{\sum_{\mathcal{T}' \in \mathcal{D}} r(\mathcal{T}')}$$

   ensuring $\sum_{\mathcal{T}} w_{\mathcal{T}} = 1$.

This balanced weighting strategy ensures that when tasks exhibit complementary characteristics (e.g., one task has fewer samples while the other is more challenging), their contributions are naturally equilibrated through the combined ranking mechanism.

Table 14 presents the sample size and baseline difficulty (zero-shot success rate) for each task.

Table 14: Task statistics: sample size and baseline difficulty (zero-shot success rate in system memory tasks; mem0 success rate in personal memory tasks).

Task	Sample Size	Baseline Success Rate (%)
DB	300	49.0
OS	144	36.1
KG	150	24.7
AlfWorld	109	17.4
LoCoMo-0	152	50.0
LoCoMo-4	178	56.18

- **Weighted Cumulative Success Rate (WCSR)**:

$$\text{WCSR}_t = \sum_{\mathcal{T} \in \mathcal{D}} w_{\mathcal{T}} \cdot P_t^{\mathcal{T}}$$

  where $\mathcal{D}$ is the set of all involved tasks and $\sum_{\mathcal{T}} w_{\mathcal{T}} = 1$.
- **Weighted Learning Gain (WLG)**:

$$\text{WLG}_t = \sum_{\mathcal{T} \in \mathcal{D}} w_{\mathcal{T}} \cdot (P_{\max}^{\text{future},\mathcal{T}} - P_t^{\mathcal{T}})$$

  where $P_{\max}^{\text{future},\mathcal{T}} = \max_{i \geq t} P_i^{\mathcal{T}}$ is the future maximum success rate of task $\mathcal{T}$ from time step t onwards (including current), measuring the overall improvement potential relative to the future optimal state in multi-task scenarios. Smaller WLG_t indicates the learning system is approaching its performance ceiling in multi-task settings.
- **Weighted Stability Loss (WSL)**:

$$\text{WSL}_t = \sum_{\mathcal{T} \in \mathcal{D}} w_{\mathcal{T}} \cdot (P_{\max}^{t,\mathcal{T}} - P_t^{\mathcal{T}})$$

  This measures the aggregate degree of forgetting in multi-task environments.

Together, these metrics characterize the learning ability (LG/WLG), retention capacity (SL/WSL), and dynamic adaptability (CSR/WCSR curve morphology) of the memory system over time.

H.3 Replay Mode

Replay Mode (Replay Learning) represents a periodic testing paradigm designed to assess the agent's ability to retain previously acquired knowledge and resist forgetting. It serves as the core mechanism for quantifying Backward Transfer. The task sequence is partitioned into multiple replay stages, where each stage consists of learning new samples followed by testing on a subset of previously learned samples.

- **Stage Partitioning Mechanism**: For system memory tasks, a test is triggered after every m new samples are learned (via R+F+E operators); the test involves randomly sampling n previously learned items for retrieval-only evaluation (R). For personal memory tasks (e.g., LoCoMo), each session constitutes a natural stage. After completing the learning phase for a session, the agent is tested on a random selection of QA pairs from that session.
- **Memory Contamination Control**: To ensure the integrity of the evaluation, test samples only invoke the retrieval operator (R) and bypass the formation (F) and evolution (E) operators. This prevents the testing process itself from altering the memory state.
- **Immediate Test**: Following the learning of each training sample, an immediate test is performed (using the R operator) to measure short-term memory effectiveness. By comparing immediate test results with subsequent replay results, the degree of forgetting can be precisely quantified.

Evaluation Metrics:

- **Average Success Rate (ASR)**: The mean performance across all replay stages, reflecting the overall retention capacity of the memory system.
- **Forgetting Rate (FR)**: This metric quantifies the performance degradation of a sample from its initial learning point to subsequent replay stages. For K replay stages, the forgetting rate for a sample s learned in stage j and re-tested in stage k ($k > j$) is defined as:

$$FR_k^{(j)}(s) = \frac{P_{\text{immediate}}^{(j)}(s) - P_{\text{replay}_k}^{(j)}(s)}{P_{\text{immediate}}^{(j)}(s)} \times 100\%$$

where: $P_{\text{immediate}}^{(j)}(s)$ is the short-term memory performance immediately after learning. $P_{\text{replay}_k}^{(j)}(s)$ is the long-term memory performance during the k-th replay stage.

The aggregate Forgetting Rate (FR) is calculated by averaging across all samples and all subsequent replay intervals:

$$FR = \frac{1}{K-1} \sum_{j=1}^{K-1} \frac{1}{|\mathcal{S}_j|} \sum_{s \in \mathcal{S}_j} \frac{1}{K-j} \sum_{k=j+1}^{K} FR_k^{(j)}(s)$$

where $\mathcal{S}_j$ denotes the set of samples learned in stage j. A low FR (approaching 0%) indicates robust resistance to forgetting, while a high FR suggests catastrophic forgetting.

This mode provides a systematic evaluation of backward knowledge transfer, serving as a pivotal paradigm in continual learning research.

H.4 TRANSFER MODE

Transfer Mode (Transfer Learning) evaluates the agent's generalization capability by applying acquired knowledge to novel scenarios. We design two distinct transfer settings:

Cross-Environment Transfer: Assesses the ability to transfer knowledge across different task domains. The agent first learns on a source task $\mathcal{T}_{\text{source}}$ (e.g., DB tasks), evolving its memory from M^0 to M^{source}. It is then tested on a target task $\mathcal{T}_{\text{target}}$ (e.g., OS tasks) by retrieving only from the source memory (invoking the R operator). The transfer gain is defined as:

$$\Delta_{\text{cross}} = P_{\text{transfer}}(\mathcal{T}_{\text{target}}) - P_{\text{zero-shot}}(\mathcal{T}_{\text{target}})$$

where P_{transfer} denotes the performance using source memory and $P_{\text{zero-shot}}$ is the baseline. This scenario verifies whether the memory system extracts abstract reusable knowledge, such as reasoning strategies or tool-use patterns.

Within-Environment Forward Transfer: Evaluates the predictive capability for future samples within the same environment after learning a current sample. Given a sequence $\{k_1, k_2, \ldots, k_N\}$, after learning sample k_i (invoking R + F + E), the agent immediately tests the N_{forward}-th future sample $k_{i+N_{\text{forward}}}$ using the current memory M^i (invoking only the R operator).

Evaluation Metrics:

1. Cross-Environment Transfer:

- **Transfer Gain (TG)**: Quantifies the improvement provided by source memory on the target task.

$$\text{Transfer Gain} = P_{\text{transfer}}(\mathcal{T}_{\text{target}}) - P_{\text{zero-shot}}(\mathcal{T}_{\text{target}})$$

A positive value indicates successful knowledge transfer, while a value ≤ 0 suggests no effect or negative transfer.

2. Within-Environment Forward Transfer:

To mirror the Forgetting Rate (FR) used in Replay mode, we introduce the Forward Transfer Gain (FTG):

- **Forward Transfer Gain (FTG)**: Measures the predictive boost for future samples after learning the current one. For a sample learned at position i, the FTG for a future sample at $i + N_{\text{forward}}$ is:

$$\text{FTG}_i = \frac{P^{(i)}_{\text{forward}}(k_{i+N_{\text{forward}}}) - P_{\text{immediate}}(k_{i+N_{\text{forward}}})}{P^{(i)}_{\text{immediate}}(k_i)} \times 100\%$$

  where:
  - $P^{(i)}_{\text{immediate}}(k_i)$ is the immediate performance on the current sample (self-learning effect).
  - $P^{(i)}_{\text{forward}}(k_{i+N_{\text{forward}}})$ is the performance on the future sample using memory M^i (forward prediction effect).
  - $P_{\text{immediate}}(k_{i+N_{\text{forward}}})$ is the immediate performance of the future sample after its own learning (baseline performance).

  $\text{FTG}_i > 0$ indicates that the memory acquired from learning the current sample positively transfers to future samples, enhancing predictive capability for future tasks; $\text{FTG}_i \approx 0$ indicates no benefit to future samples; $\text{FTG}_i < 0$ indicates negative transfer where current learning interferes with future task execution.

This mode systematically evaluates knowledge generalization and forward transfer, complementing the backward transfer (resistance to forgetting) assessed in Replay mode to provide a complete profile of the memory system's temporal transfer characteristics.

H.5 Repair Mode

Repair Mode (Repair Learning) tests the self-correction capabilities of the memory system when faced with erroneous feedback and knowledge conflicts. The sample set is organized into multiple repair groups, within which n samples are randomly selected as inverted samples (correct outcomes are labeled as failures, and failures as successes). Each group executes through four sequential phases:

Phase 1 - Erroneous Learning (wrongJudge): The agent learns all m samples under incorrect reward signals (invoking R + F + E operators). Inverted samples receive inverted rewards while others receive correct ones, resulting in a contaminated memory M_{wrong}.

Phase 2 - Erroneous Testing (wrongJudgeTest): All samples are tested using correct rewards (invoking only the R operator) to evaluate the performance of the contaminated memory. Memory is not updated during this phase; M_{wrong} is used solely for retrieval-augmented decision-making.

Phase 3 - Corrective Repair (rightJudge): The agent re-learns all samples under correct rewards (invoking R + F + E operators), revising the contaminated memory into M_{right}. This phase allows the system to identify and rectify previously stored erroneous knowledge.

Phase 4 - Repair Testing (rightJudgeTest): All samples are tested again with correct rewards (invoking only the R operator) to assess post-repair performance.

Evaluation Metrics:

The Repair mode employs three core normalized metrics to comprehensively evaluate the error tolerance and repair capability of the memory system. All metrics are normalized relative to the baseline performance P_{online} from normal online learning to eliminate the influence of different dataset baselines:

1. Repair Gain (RG): Measures the recovery capability of the memory system from erroneous learning

$$\text{Repair Gain}_{\text{Full}} = \frac{P_{\text{rightJudgeTestFull}} - P_{\text{wrongJudgeTestFull}}}{P_{\text{online}}} \times 100\%$$

$$\text{Repair Gain}_{\text{Standard}} = \frac{P_{\text{rightJudgeTestStandard}} - P_{\text{wrongJudgeTestStandard}}}{P_{\text{online}}} \times 100\%$$

The Full metric evaluates the repair effect across all samples, while the Standard metric focuses specifically on the inverted samples. Higher Repair Gain indicates stronger repair capability.

2. Error Robustness (ER): Measures the resistance of the memory system to erroneous learning

$$\text{Error Robustness}_{\text{Full}} = \frac{P_{\text{wrongJudgeTestFull}} - P_{\text{online}}}{P_{\text{online}}} \times 100\%$$

$$\text{Error Robustness}_{\text{Standard}} = \frac{P_{\text{wrongJudgeTestStandard}} - P_{\text{online}}}{P_{\text{online}}} \times 100\%$$

where P_{online} is the final cumulative success rate under normal online learning mode (baseline performance). Error Robustness closer to 0% indicates the memory system maintains high performance even after erroneous learning, demonstrating strong error resistance; Error Robustness < 0 indicates performance degradation due to erroneous learning, with more negative values indicating more severe degradation.

3. Net Recovery (NR): Measures the degree of performance recovery relative to normal learning after repair

$$\text{Net Recovery}_{\text{Full}} = \frac{P_{\text{rightJudgeTestFull}} - P_{\text{online}}}{P_{\text{online}}} \times 100\%$$

Net Recovery $\approx 0\%$ indicates complete recovery to normal learning levels; Net Recovery > 0 indicates post-repair performance exceeds the normal baseline (the memory system learns from errors); Net Recovery < 0 indicates incomplete repair, remaining below the normal baseline.

These metrics collectively characterize the **error tolerance** (Error Robustness), **repair capability** (Repair Gain), and **recovery completeness** (Net Recovery) of the memory system. This represents the first conflict-correction evaluation mechanism introduced in a memory benchmark, validating whether the memory system can identify and rectify erroneous knowledge. It serves as a critical paradigm for assessing memory **robustness** and **knowledge conflict resolution capability**.

I Baseline Methods: Detailed Specifications

We formally define the following baseline methods, which represent distinct design paradigms for memory mechanisms.

I.1 Zero-shot

The zero-shot approach operates without any memory mechanism. The agent generates actions based solely on current observations:

$$a_t = \pi(o_t, \perp)$$

where $\perp$ denotes a null memory signal, indicating that no memory retrieval is performed. This method reflects the fundamental instruction-following capability of the LLM and serves as the performance lower bound.

I.2 StreamICL

StreamICL (Wu et al., 2024): This streaming in-context learning method utilizes a vector database to store and retrieve complete interaction trajectories of historically successful tasks. When executing sub-task k, the retrieval operator $R^{streamICL}$ retrieves the $top-k$ most similar experiences from $M_k^{streamICL}$ based on the specific requirements Q_k:

$$m_k = R^{streamICL}(M_k^{streamICL}, Q_k)$$

Upon the successful completion of sub-task k, the formation operator $F^{streamICL}$ formats the complete trajectory, and the evolution operator $E^{streamICL}$ appends it to the vector database.

I.3 MEM0

Mem0 (Chhikara et al., 2025): Mem0 is a scalable long-term memory system primarily focused on user preference retention. The memory system M^{mem0} employs a hybrid storage architecture combining graph structures and vector databases. The graph structure uses entities as nodes and relations as edges to build a structured knowledge graph, while the vector database handles semantic retrieval and re-ranking. During sub-task k, the retrieval operator R^{mem0} first obtains candidate memories via semantic search and then filters and re-ranks them using the graph structure to obtain $top - k$ relevant items.

Upon successful task completion, Mem0 utilizes a lightweight LLM (gpt-4o-mini) for memory management across two stages:

- **Memory Formation (F^{mem0})**: The LLM extracts user-related structured facts from the dialogue history ϕ^k to generate memory candidates. Simultaneously, it identifies entities and relations to update the graph.
- **Memory Evolution (E^{mem0})**: The LLM performs four operations (ADD, UPDATE, DELETE, NONE) on the candidates and existing M_k to achieve conflict resolution, deduplication, and integration.

While Mem0 excels at maintaining long-term user preferences, it lacks the capacity to accumulate procedural task execution experience.

I.4 AWMPRO

awmPro (Modified from Agent Workflow Memory (Wang et al., 2024)): This workflow-based approach improves execution efficiency by recording and retrieving workflow histories. The workflow memory M^{awm} stores structured workflows in a vector database. During sub-task k, the retrieval operator R^{awm} retrieves the $top - k$ most similar workflows based on query Q_k.

After a successful task, awmPro updates its memory via a two-stage process:

- **Memory Formation (Workflow Induction, F^{awm})**: A lightweight LLM (Qwen2.5-7B-Instruct) distills reusable workflow descriptions W_k from the full trajectory h_k using specific prompts, generalizing execution steps into natural language patterns.
- **Memory Evolution (Workflow Management, E^{awm})**: A hybrid of vector retrieval and LLM judgment is used for management:
  - Vector Retrieval: For each new workflow $w \in W_k$, it identifies the $top - K$ similar workflows $\{w'_1, w'_2, \ldots, w'_K\}$ in M_k^{awm}.
  - LLM Judgment: A lightweight LLM (Qwen2.5-7B-Instruct) compares w with the similar set and executes one of the four operations (ADD, UPDATE, DELETE, NONE) to produce the updated memory: $M_{k+1}^{awm} = E^{awm}(M_k^{awm}, W^k)$.

This method focuses on accumulating execution experience and workflow reuse but lacks maintenance of user-specific preferences.

I.4.1 WORKFLOW INDUCTION PROMPT

The workflow induction process uses the following prompt to extract reusable workflows from trajectories:

```
Given a completed trajectory, your task is to extract reusable
workflow(s) from it.  Each workflow should be a commonly-reused
sub-routine.  Do not generate similar or overlapping workflows.
Each workflow should have at least two steps.

OUTPUT FORMAT (strictly follow this format):    workflow_name Given
that [context description], this workflow [what it does].  [step 1
description with {variable} placeholders] [step 2 description with
{variable} placeholders] ...

IMPORTANT: - Each workflow must start with " workflow_name"
(use snake_case for names) - The next line should describe the
context and purpose - Then list the steps, using {variable-name}
for parameterizable values - Extract multiple workflows if the
trajectory contains different reusable patterns - Use clear,
descriptive variable names in {curly_braces}
```

I.4.2 WORKFLOW MANAGEMENT PROMPT

The workflow management process uses the following prompt template to decide operations (ADD, UPDATE, DELETE, NONE) for each new workflow:

```
You are a smart workflow memory manager. You control the workflow
memory of a system. You can perform four operations: (1) ADD, (2)
UPDATE, (3) DELETE, and (4) NONE.

Based on the existing workflows and new workflows, you need to
decide which operation to perform for EACH new workflow.

Operation Guidelines:

1. ADD: Use when the new workflow does not exist in the existing
workflows. - The new workflow describes a different task or
pattern - Generate a descriptive workflow name as the ID (e.g.,
"filter_table_by_condition", "update_user_profile") - [IMPORTANT]
If existing workflows list is EMPTY, ALL new workflows must be ADD
operations

2. UPDATE: Use when the new workflow is similar to an existing
workflow but provides more details or improvements. - The new
workflow covers the same task but with better description or
additional steps - Use the exact numeric ID from the existing
workflow (e.g., "0", "1", "2") - Include the "old_memory" field
with the original workflow text - [IMPORTANT] NEVER use UPDATE if
the workflow ID does not exist in the existing workflows list

3. DELETE: Use when the new workflow contradicts an existing
workflow. - The new workflow provides information that conflicts
with an existing workflow - Use the exact numeric ID from the
existing workflow - This operation is rare for workflow management

4. NONE: Use when the new workflow already exists in the existing
workflows. - The new workflow is essentially the same as an
existing workflow - No changes needed

[CRITICAL RULES] - You MUST return one operation for
EACH new workflow - For ADD operations: ID must be a
descriptive_workflow_name (snake_case), NOT "ADD" or a number - For
UPDATE/DELETE operations: ID must be the exact numeric ID from
existing workflows (e.g., "0", "1", "2"), NOT a descriptive name
- If existing workflows is empty or "None", ALL operations must
be ADD - NEVER use UPDATE/DELETE with IDs that don't exist in the
existing workflows list

Existing workflows (format: "ID: <number>"): {existing_workflows}

New workflows: {new_workflows}

Return JSON only (no other text).
```

The LLM compares each new workflow with the top-K similar existing workflows retrieved via vector search, and returns decisions in JSON format with fields: `id`, `text`, `event`, and `old_memory` (for UPDATE operations).

I.5 MEMs: Multi-Memory Methodology

MEMs (Multi-Memory Methodology): This multi-memory coordination framework addresses the limitations of single-memory approaches in hybrid scenarios. MEMs maintains two specialized memory stores—System Memory (M^{sys}) and Personal Memory (M^{pers})—and employs a lightweight trigger model $\mathcal{T}$ as a metacognitive router to coordinate them.

Retrieval Phase: During the execution of sub-task k, the trigger model $\mathcal{T}$ analyzes the specific requirements Q_k to determine which memory systems to activate. It can select system memory, personal memory, both, or neither:

$$S_{\text{retrieve}} = \mathcal{T}(Q_k), \quad S_{\text{retrieve}} \subseteq \{\text{sys}, \text{pers}\}$$

The agent retrieves from selected sources and generates actions based on the integrated memory signals:

$$m_k = \bigoplus_{s \in S_{\text{retrieve}}} R(M_s, Q_k), \quad a_t = \pi(o_t, m_k)$$

where $\bigoplus$ denotes memory signal aggregation.

Update Phase: Upon completion of task k, the trigger model evaluates the interaction trajectory h^k to decide which memory systems require updating:

$$S_{\text{update}} = \mathcal{T}(h_k), \quad S_{\text{update}} \subseteq \{\text{sys}, \text{pers}\}$$

For each $s \in S_{\text{update}}$, the memory is updated via formation and evolution operators:

$$M_s^{k+1} = E_s(F_s(M_s^k, \phi_k), M_s^k)$$

where F_s and E_s are source-specific (e.g., F_{sys} extracts workflows, while F_{pers} extracts user facts).

Trigger Model Implementation: The trigger model is implemented as a lightweight LLM (Qwen2.5-7B-Instruct) that performs two-stage decision-making:

1. **Retrieval Decision**: Before task execution, the trigger analyzes task requirements Q_k to determine memory retrieval needs. It classifies the task type (system-task vs. personal-interaction) based on linguistic features and task context.
2. **Update Decision**: After task completion, the trigger analyzes the trajectory h^k to identify which knowledge should be stored. System memory is updated when procedural patterns are detected (e.g., repeated action sequences, tool-use workflows); personal memory is updated when user-specific information appears (e.g., preferences, facts, dialogue context).

Memory Isolation and Specialization: By maintaining separate memory stores and intelligent routing, MEMs achieves three key advantages: (i) *Memory Purity*: System and personal memories remain isolated, preventing contamination from heterogeneous task types; (ii) *Specialization Maintenance*: Each memory system maintains high quality in its domain through specialized formation and evolution operators; (iii) *Dynamic Adaptability*: The trigger model enables flexible memory coordination in mixed-task scenarios where task boundaries are blurred.

This architecture mimics the functional coordination of the human prefrontal cortex, which manages diverse memory systems such as hippocampus-dependent declarative memory and striatum-dependent procedural memory (Rudy et al., 2005; Eichenbaum, 2017; Miller & Cohen, 2001), enabling genuine multi-memory synergy in real-world agent applications.

I.5.1 TRIGGER MODEL PROMPTS

The trigger model employs two specialized prompts for different decision stages:

Retrieval Decision Prompt: Before task execution, the trigger model analyzes the task query to determine which memory sources to retrieve from:

```
You are a memory source selector. Given the task query, decide
which memory source(s) should be retrieved from.

Available memory sources: - system_memory: Stores task execution
workflows and experiences. Use for tasks requiring procedural
knowledge, tool usage patterns, or execution strategies. -
personal_memory: Stores user preferences and conversation
history. Use for tasks requiring personalization or user-specific
information.

Task query: {query}

You must return your decision in JSON format: { "sources":
["system_memory"] | ["personal_memory"] | ["system_memory",
"personal_memory"], "reasoning": "brief explanation" }
```

Update Decision Prompt: After task completion, the trigger model analyzes the interaction trajectory to determine which memory sources should be updated:

```
You are a memory source selector. Given the task interaction
trajectory, decide which memory source(s) should be updated.

Available memory sources: - system_memory: Stores task execution
workflows and experiences. Update when the trajectory contains
procedural patterns, repeated action sequences, or tool-use
workflows. - personal_memory: Stores user preferences and
conversation history. Update when the trajectory contains
user-specific information such as preferences, facts, or dialogue
context.

Task interaction trajectory: {history}

You must return your decision in JSON format: { "sources":
["system_memory"] | ["personal_memory"] | ["system_memory",
"personal_memory"], "reasoning": "brief explanation" }
```