MEMAGENT: LEARNING TO MANAGE HETEROGENEOUS MEMORY PROVIDERS FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Current agents remain largely stateless across tasks, limiting their ability to continually improve from prior interactions and making memory essential for long-horizon agentic behavior. Existing memory methods seek to reuse past experience, but most rely on a single memory representation (*e.g.*, trajectories, reflections, skills, structured knowledge) whose effectiveness varies across task distributions. Rethinking this design space, we evaluate 13 memory methods and find that no single method generalizes across benchmarks, revealing the potential of *managing* heterogeneous memory providers. We formulate agent memory as a *routing* problem in which a memory agent decides which memory provider to retrieve from, whether to inject short-term memory, and which providers should store the resulting experience. Based on this perspective, we propose **MemAgent**, featuring a content-aware routing architecture and a training-data synthesis pipeline. The routing architecture combines content-aware probing before retrieval, short-term memory gating during execution, and selective multi-provider storage, while the training pipeline synthesizes phase-specific supervision for routing decisions. Across GAIA, WebWalkerQA, and xBench-DS, MemAgent improves average accuracy by 10.0% and outperforms the best individual memory method. These gains come with less than 0.3% routing overhead and a 12% reduction in average task steps.

1 INTRODUCTION

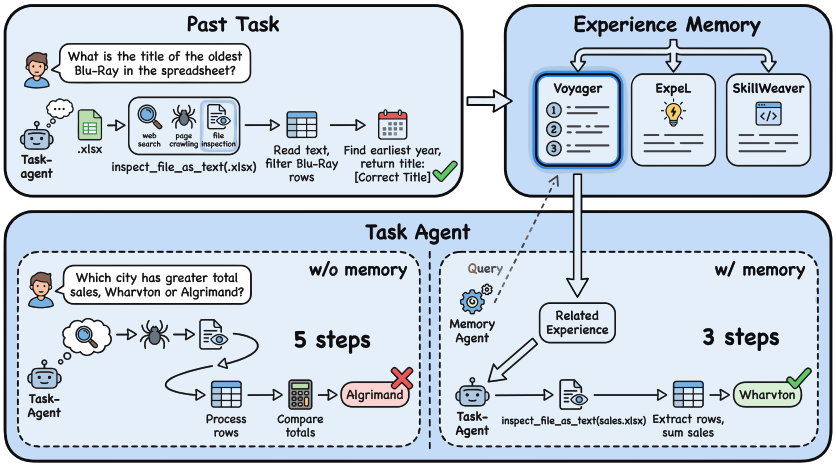


Figure 1: The task agent interacts with tools to solve tasks, while experiences from past tasks are stored in heterogeneous memory providers. By routing retrieval to the appropriate provider, the memory agent helps the task agent solve the new task correctly in fewer steps.

LLM agents have rapidly extended beyond single-turn question answering into long-horizon tasks such as web research, multimodal reasoning, and tool-augmented problem solving (Yao et al., 2023;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Xu et al., 2025; Shi et al., 2026b; Li et al., 2025b). Despite operating in increasingly complex environments, most agents remain largely *stateless across tasks*: each invocation begins from scratch, with no mechanism to reuse prior experience or learned strategies accumulated over time. This statelessness is a fundamental obstacle to continual improvement and limits an agent's ability to amortize effort across related tasks (Xu et al., 2026; Ouyang et al., 2026). Memory has thus emerged as an essential mechanism for sustained agentic behavior, prompting a rich line of work on how to capture, represent, and reuse past interactions (Du et al., 2026; Long et al., 2026; Li et al., 2025a; Shu et al., 2026).

Existing memory-augmented agents primarily differ in *how* they represent past experience: summarized trajectories (Wang et al., 2024), reflected insights (Shinn et al., 2023), structured knowledge (Tang et al., 2025), workflow templates (Wang et al., 2025c), procedural scripts (Fang et al., 2026), or executable skills (Zheng et al., 2025). Although each method reports performance gains on its target benchmark, these designs are typically proposed and evaluated in isolation. Because each approach commits to a single memory representation, a central question remains open: *does any single representation remain consistently effective across different task distributions?*

To answer this question, we unify 13 representative memory methods under a common interface, evaluating each as a distinct *memory provider* across three benchmarks: GAIA, WebWalkerQA, and xBench DeepSearch. Our results show strong cross-benchmark heterogeneity (see rethinking in Sec. 3); no single method universally excels, and top performers on one benchmark frequently underperform elsewhere. Crucially, a *hindsight oracle* (which considers a task solved if any provider succeeds) outperforms the best individual provider by over 18%. This untapped potential suggests that agent memory is less about finding a single optimal format and more about solving a routing problem over heterogeneous memory providers. Providers differ not only in the content they store but also in how they index and retrieve it, *e.g.*, by surface similarity, structural abstraction, or composable insights. Therefore, provider choice should account for the actual retrieved content and each provider's transfer bias rather than relying only on surface-level task similarity.

Motivated by this observation, we formulate agent memory as a *routing problem* over a heterogeneous provider pool (see Fig. 1). For each task, a dedicated memory agent $\mathcal{A}_M$ makes three decisions across the memory lifecycle: which long-term provider to retrieve from before execution (BEGIN), whether to inject short-term working memory during execution (IN), and which providers should store the resulting trajectory afterward (END). Unlike static tool routing, these decisions depend not only on provider descriptions but also on the execution context in which each decision is made.

To realize this concept, we propose MemAgent, a framework comprising two core components. The first is a **content-aware routing architecture** that spans the full memory lifecycle. At the BEGIN phase, a lightweight *probe* previews each provider before routing. During the IN phase, *short-term memory gating* injects working-memory summaries only when they are likely to assist the next action. At the END phase, *selective multi-provider storage* writes trajectories exclusively to providers where they will likely be useful, thereby enhancing future retrieval and preventing memory pollution. The second component is a **training-data synthesis pipeline**. This pipeline generates supervision for routing decisions through forced exploration and phase-specific reward signals, enabling reward-filtered supervised fine-tuning of the memory agent $\mathcal{A}_M$.

Across GAIA, WebWalkerQA, and xBench-DS, MemAgent improves average accuracy by 10.0% over the baseline and beats every single-provider method on all three benchmarks. It also improves efficiency, reducing average task steps by 12% while adding less than 0.3% routing overhead.

In summary, our main contributions are threefold:

- We rethink agent memory through a cross-benchmark study of heterogeneous memory providers, showing that no single provider consistently dominates across tasks. Moreover, we propose a graph-structured long-term memory provider and a short-term working-memory provider.
- We formulate memory as a three-phase routing problem (BEGIN/IN/END) and propose MemAgent, a framework comprising a content-aware routing architecture and a training-data synthesis pipeline to supervise routing decisions without ground-truth labels.
- Across GAIA, WebWalkerQA, and xBench-DS, MemAgent improves average accuracy by 10.0% over the baseline and outperforms every single-provider method. Meanwhile, it reduces average task steps by 12% while adding less than 0.3% routing overhead.

2 Related Work

Memory has become a central mechanism for enabling long-horizon agentic behavior, but recent work spans a broad and fragmented design space (Yu et al., 2026a; Xiang et al., 2026). Prior surveys organize agent memory along dimensions such as functional role (*e.g.*, working, factual, and experiential memory) and lifecycle (formation, retrieval, and refinement) (Hu et al., 2026b; Zhang et al., 2025b). We group prior work into three directions.

Working memory and context management. To actively manage context windows, MEM1 (Zhou et al., 2026b) maintains a compact internal state under a fixed budget, while AgeMem (Yu et al., 2026b) formulates memory operations as RL-trained actions. For long-horizon web and deep-research tasks, agents either reflectively reorganize their workspaces between steps (ARC (Yao et al., 2026), IterResearch (Chen et al., 2026)) or periodically distill trajectories into compact reasoning states (ReSum (Wu et al., 2025c), AgentFold (Ye et al., 2026)).

Long-term and reusable memory. Prior work studies cross-task memory in several forms, including summarized trajectories (Wang et al., 2024), reflected successes and failures (Zhao et al., 2024), importance-weighted memory pools (Park et al., 2023), semantically indexed retrieval (Zhong et al., 2024), structured external memory (Wang et al., 2025a), and utility-aware retrieval (Zhang et al., 2026c); MemoryAgentBench (Hu et al., 2026a) further broadens how such systems are evaluated. A related line stores reusable procedures or skills instead of raw trajectories, as in ReMe (Cao et al., 2026), Memento-Skills (Zhou et al., 2026a), and MemSkill (Zhang et al., 2026b). These works demonstrate the value of long-term reuse, but typically retain one fixed representation.

Learning or evolving the memory design itself. Beyond improving fixed memory systems, recent works also attempt to automate memory design. For instance, ALMA (Xiong et al., 2026) meta-learns memory designs as executable programs, whereas MemEvolve (Zhang et al., 2026a) evolves memory architectures across a pool of providers. Automated agent design frameworks (Hu et al., 2025; Shang et al., 2025) treat memory components as one axis within a broader search space for agent architectures. These approaches are compatible with our observation that different memory designs exhibit complementary behavior, although they seek a better unified design for the system as a whole.

Most prior work improves a single memory system or a unified memory controller. In contrast, MemAgent treats memory heterogeneity as an opportunity: instead of proposing another memory format or globally optimized design, it learns an inference-time routing policy over heterogeneous long-term providers, together with short-term memory gating and selective multi-provider storage.

3 Rethinking Agent Memory

Recent memory-augmented agents primarily differ in how they represent past experiences, utilizing formats such as experience libraries (Zhao et al., 2024), workflow templates (Wang et al., 2025c), procedural scripts (Fang et al., 2026), knowledge graphs (Zhang et al., 2025a), and agent skills (Zheng et al., 2025). Although each method demonstrates gains on its target benchmark, it remains unclear whether any single memory paradigm performs consistently across tasks.

The illusion of a universal memory. To test this assumption, we evaluate 13 representative memory methods, each used as the agent's sole memory system, on three diverse benchmarks: GAIA (general reasoning), WebWalkerQA (web navigation), and xBench-DS (deep search).

Fig. 2 shows strong cross-benchmark heterogeneity. AWM leads on GAIA ($+6.1\%$) but slightly hurts WebWalkerQA (-0.6%), whereas ExpeL leads on WebWalkerQA ($+7.6\%$) but is only mid-tier on GAIA. No method is best on more than one benchmark. Different tasks favor different memory representations, so simply designing a "better" single memory is unlikely to close the gap.

The routing opportunity. Given the observed cross-benchmark complementarity, routing emerges as a promising solution. A hindsight oracle, which considers a task solved if any of the 13 single-provider systems succeeds, achieves an average accuracy of 89.6% (Table 1). This far exceeds the best individual provider (71.0%), further supported by the fact that 10 of the 13 providers uniquely solve

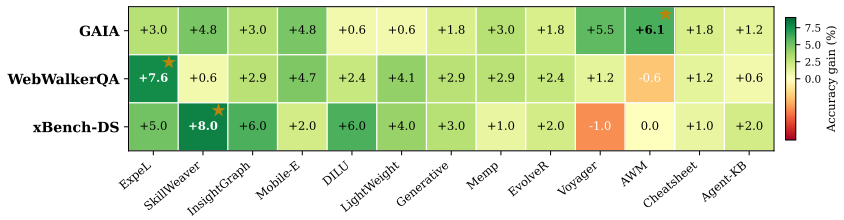


Figure 2: Cross-benchmark gains of 13 memory methods over the no-memory baseline. No single method dominates across all three benchmarks. Stars mark the best method on each benchmark.

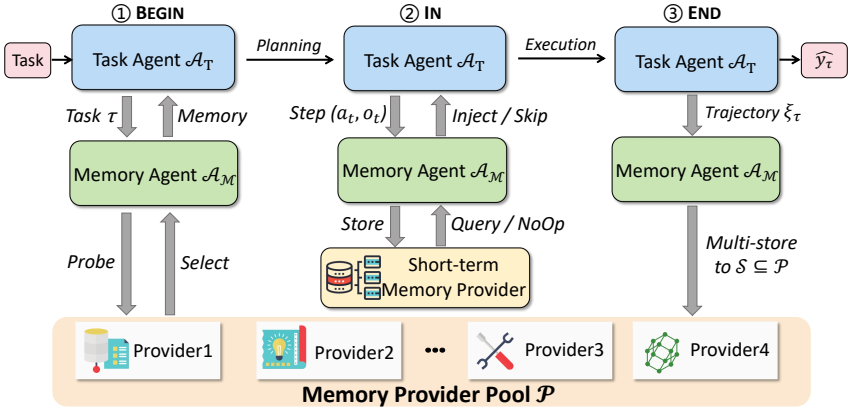


Figure 3: For each task, the memory agent $\mathcal{A}_M$ makes three routing decisions. **(I) BEGIN:** It previews all providers in $\mathcal{P}$, selects one, and injects the retrieved long-term memory into the task agent $\mathcal{A}_T$'s prompt. **(II) IN:** At each step, the short-term memory provider summarizes cross-step state from tool interactions (a_t, o_t), and $\mathcal{A}_M$ decides whether to inject the summary or skip. **(III) END:** After task completion, $\mathcal{A}_M$ selects a subset $\mathcal{S} \subseteq \mathcal{P}$ to store the trajectory ξ_τ for future reuse.

at least one task. These results expose a substantial opportunity for selecting among existing memory structures on a per-task basis. As illustrated in Appendix D, providers expose different representations and retrieval contracts; selection should therefore consider the actual retrieved content and each provider's transfer bias rather than task surface alone. This shift, however, introduces a credit-assignment challenge: task-level outcomes fail to indicate which provider or phase is responsible, because storage mechanisms receive only delayed feedback.

4 METHODOLOGY

We introduce MemAgent, a framework in which a memory agent manages diverse memory providers for a task agent. It comprises two components: (1) a content-aware routing architecture (Sec. 4.2), and (2) a data synthesis pipeline that provides routing supervision without ground-truth labels (Sec. 4.3). MemAgent learns a per-decision policy to route, inject, and store memories via the appropriate provider throughout task execution. Fig. 3 illustrates the overall architecture.

4.1 PROBLEM FORMULATION

Consider a task agent $\mathcal{A}_T$ that solves multi-step agentic tasks. At each task τ, the agent executes a trajectory $\xi_\tau = (s_0, a_0, o_0, \ldots, s_T, a_T)$ of states, actions, and observations, producing a binary outcome $y_\tau \in \{0, 1\}$ (correct or not). We augment $\mathcal{A}_T$ with a memory agent $\mathcal{A}_M$ that makes three types of memory decisions during the lifecycle of ξ_τ: (i) **BEGIN** (d_{begin}): before execution starts, $\mathcal{A}_M$ selects a long-term memory provider $p \in \mathcal{P}$ to retrieve relevant cross-task experience, which is injected into $\mathcal{A}_T$'s system prompt prior to s_0; (ii) **IN** ($d_{\text{in}}^{(t)}$): at each step t during ξ_τ, $\mathcal{A}_M$ decides whether to inject structured short-term memory or skip injection; (iii) **END** (d_{end}): after ξ_τ completes, $\mathcal{A}_M$ selects a subset of providers $\mathcal{S} \subseteq \mathcal{P}$ to store ξ_τ for future retrieval by other tasks.

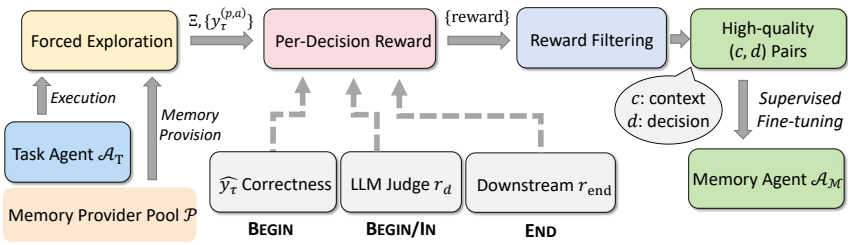


Figure 4: **Training data synthesis pipeline for the memory agent.** Forced exploration enumerates provider-action combinations for each task, producing trajectories Ξ with binary outcomes $\{y_\tau^{(p,a)}\}$. Supervision signals are derived from task correctness (for BEGIN), LLM-as-Judge attribution (for BEGIN and IN), and downstream reuse (for END). Finally, reward filtering retains high-quality context-decision pairs (c, d) for the supervised fine-tuning of $\mathcal{A}_M$.

We formulate the memory agent as a policy $\mathcal{A}_M(d \mid c)$ over a decision space $\mathcal{D}$ given context $c \in \mathcal{C}$ (*e.g.*, task state), trained to maximize the expected success rate $\max_{\mathcal{A}_M} \mathbb{E}_\tau[y_\tau]$.

To leverage memory diversity across varying tasks, MemAgent maintains a pool of K heterogeneous memory providers. These providers share a unified interface with two standard operations: **retrieving** relevant memories and **storing** task trajectories as reusable experiences. This abstraction decouples the routing policy from provider internals, allowing $\mathcal{A}_M$ to interact with diverse memory paradigms while remaining agnostic to their underlying implementations. In addition, we contribute a short-term memory provider for within-task working memory, and a graph-structured memory provider, InsightGraph. All providers are introduced in Table 6 (Appendix A.1).

4.2 CONTENT-AWARE ROUTING

The memory agent $\mathcal{A}_M$ maps a phase-specific decision context c to the routing actions.

BEGIN phase: content-aware probing. Routing based solely on the task description q and static provider profiles is unreliable (Fioresi et al., 2026); what matters is whether a provider *actually holds* relevant content. Therefore, we augment the BEGIN context c with *probe results*. Prior to the routing decision, $\mathcal{A}_M$ issues a lightweight parallel retrieval against all providers, obtaining a short snippet preview of each provider's most relevant memory. These previews are injected without their retrieval scores. Since providers use incompatible scoring scales, exposing scores would teach the model a spurious "pick the highest-scored provider" shortcut rather than forcing it to reason about semantic relevance. To mitigate positional bias, candidate providers are randomly shuffled.

IN phase: short-term memory gating. To determine whether to inject short-term memory at step t, the context c includes q, the step index t, a compact summary of the current short-term memory state, and the task agent's recent actions (*i.e.*, recent tool calls and observations from ξ_τ). This allows $\mathcal{A}_M$ to judge whether the recalled state would materially aid the next step.

END phase: multi-provider storage. To facilitate the selection of providers for storing the completed trajectory, the context c comprises a trajectory summary, the binary task outcome, and provider-specific descriptions detailing what each would extract and retain.

4.3 TRAINING DATA SYNTHESIS

The central challenge of training $\mathcal{A}_M$ is the absence of ground-truth routing labels. We address this by synthesizing supervision through three stages: forced exploration, per-decision reward scoring, and reward-filtered data construction.

Forced exploration. For each training task τ, we systematically iterate through all providers by forcing $\mathcal{A}_M$ to route to each $p_i \in \mathcal{P}$ in turn, while additionally varying the IN-phase action (QUERY or NOOP). The task agent $\mathcal{A}_T$ executes normally in every run; only the memory routing is overridden. This process yields a trajectory set $\Xi = \{\xi_\tau^{(p,a)} : \tau \in \mathcal{T}_{\text{train}},\ p \in \mathcal{P},\ a \in \mathcal{A}\}$ with corresponding recorded outcomes $\{y_\tau^{(p,a)}\}$. By observing how the *same* task performs under *every* provider, we better control for task difficulty and obtain comparisons across providers.

Per-decision reward. We derive fine-grained per-decision rewards from three complementary sources: current-task correctness, LLM-judge attribution, and downstream reuse.

Signal 1: Task correctness (BEGIN). For task τ, let $\mathcal{P}_\tau^+ = \{p : y_\tau^{(p,\cdot)} = 1\}$ denote the set of providers that led to correct outcomes. A BEGIN decision routing to $p \in \mathcal{P}_\tau^+$ receives positive reward. We further weight by discriminability: task correctness carries the strongest signal when $\mathcal{P}_\tau^+$ is a proper non-empty subset of $\mathcal{P}$, *i.e.*, the task separates effective providers from ineffective ones.

Signal 2: LLM-as-Judge for causal attribution (BEGIN/IN). Task correctness provides a binary signal but cannot quantify *how much* a particular retrieval or injection contributed to y_τ. We employ an LLM judge (Zheng et al., 2023) that examines each decision's context (what was retrieved or injected) alongside the task agent's subsequent actions (whether the memory was actually utilized). The judge scores multiple attribution dimensions (Appendix B.2), which we aggregate into a scalar reward:

$$r_d \;=\; \big\langle \mathbf{w}^{(\phi)}, \hat{\mathbf{s}} \big\rangle \;=\; \sum_{i=1}^{K_\phi} w_i^{(\phi)}\, \hat{s}_i, \quad \hat{s}_i = \begin{cases} s_i / s_{\max}, & \text{dim. } i \uparrow \\ 1 - s_i / s_{\max}, & \text{dim. } i \downarrow \end{cases}, \quad \mathbf{w}^{(\phi)} \in \Delta^{K_\phi - 1} \tag{1}$$

where $\phi \in \{\text{BEGIN}, \text{IN}\}$ denotes the decision phase with K_ϕ judge dimensions, $s_i \in \{0, \ldots, s_{\max}\}$ is the raw judge score, and $\uparrow$ / $\downarrow$ indicate positive/negative dimensions (*e.g.*, Utilization vs. Noise). The phase-specific weight vector $\mathbf{w}^{(\phi)} \in \Delta^{K_\phi - 1}$ satisfies $w_i^{(\phi)} \geq 0$ and $\sum_i w_i^{(\phi)} = 1$.

Signal 3: Downstream reward (END). For storage decisions, we define the reward based on actual downstream reuse, as a storage choice is valuable only if the reused memory *also* contributes to the success of the retrieving task. Let ξ_τ be a trajectory stored in provider p, and let $\mathcal{T}_{>\tau}^{(p)} = \{\tau' > \tau : \text{RETRIEVED}(\tau', p) \cap \text{STORED}(\xi_\tau, p) \neq \emptyset\}$ denote the set of subsequent tasks where the BEGIN retrieval from p contains items originating from ξ_τ. We combine *hit coverage* and *downstream success rate* into a single scalar reward:

$$r_{\text{end}}(\xi_\tau, p) = \tfrac{1}{2} \min\Big(1,\ \frac{|\mathcal{T}_{>\tau}^{(p)}|}{H}\Big) + \tfrac{1}{2} \cdot \frac{\big|\{\tau' \in \mathcal{T}_{>\tau}^{(p)} : y_{\tau'} = 1\}\big|}{\max(|\mathcal{T}_{>\tau}^{(p)}|,\ 1)} \tag{2}$$

where $\text{STORED}(\xi_\tau, p)$ denotes the memory items written by trajectory ξ_τ to p, $\text{RETRIEVED}(\tau', p)$ the items retrieved by task τ' from p, and H is a saturation constant that prevents a single heavily-reused trajectory from dominating the reward.

These three signals are complementary: task correctness defines the *search space boundary* (only y_τ=1 decisions enter the training pool), LLM-as-Judge provides *quality gradation* within this space, and downstream reward provides *ground-truth storage labels*.

Reward-filtered SFT data construction. We construct SFT data from the scored decisions using phase-specific filtering. During the BEGIN phase, task-level correctness acts as the primary filter via $\mathcal{P}_\tau^+$: (i) if $\mathcal{P}_\tau^+$ is a proper non-empty subset of $\mathcal{P}$, we keep all decisions routing to $p \in \mathcal{P}_\tau^+$; (ii) if all providers succeed ($\mathcal{P}_\tau^+ = \mathcal{P}$), we retain the top-$k$ decisions ranked by judge reward r_d; and (iii) if all fail ($\mathcal{P}_\tau^+ = \emptyset$), we drop the task. Decisions with $r_d = 0$, indicating empty or irrelevant retrievals, are also discarded. For the IN phase, a threshold $r_d \geq \epsilon_{\text{IN}}$ retains high-quality QUERY and NOOP decisions. Similarly, the END phase retains decisions where $r_{\text{end}} \geq \epsilon_{\text{END}}$. Each training example is a (prompt, completion) pair comprising the decision context c and the corresponding structured routing decision, matching the space defined in Sec. 4.2.

SFT training. We fine-tune the $\mathcal{A}_M$ via LoRA (Hu et al., 2022), minimizing cross-entropy loss exclusively on structured decision tokens (Ouyang et al., 2022) by masking prompt tokens. To prevent data leakage from shared contexts, we perform a task-level train-validation split, ensuring all BEGIN/IN/END decisions from the same task remain grouped. The end-to-end inference pipeline is summarized in Algorithm 1.

5 EXPERIMENTS

In this section, we first describe our experimental setup. Then, we present our main results. We also provide ablation studies and discussions for a thorough analysis.

5.1 Experimental Setup

Benchmarks. We evaluate on three agentic benchmarks requiring multi-step tool use and reasoning: GAIA (Mialon et al., 2024) (across three difficulty levels), WebWalkerQA (Wu et al., 2025b) (multi-turn web-navigation queries), and xBench-DS (Chen et al., 2025) (planning and tool-use tasks). Dataset details are in Appendix A.2.

Agent framework. We use Flash-Searcher (Qin et al., 2026), a single-agent deep research framework as the task agent, equipped with four tools: `WebSearch` (search engine queries), `CrawlPage` (URL content extraction), `VisualInspector` (image analysis), and `AudioInspector` (audio transcription). The task agent is powered by GPT-5-mini via API. The memory agent is Qwen3.5-4B (Team, 2026) fine-tuned with LoRA (r=16, α=32, dropout= 0.05; 0.84% trainable).

Training configuration. Following MemEvolve (Zhang et al., 2026a), training uses 53 GAIA Level-1 tasks and 120 synthetically generated multi-hop tasks from TaskCraft (Shi et al., 2026a). Per-decision reward scoring uses GPT-5-mini as the LLM-as-Judge. SFT training uses learning rate 2×10^{-5}, cosine schedule, warmup ratio 0.1, effective batch size 64, and maximum sequence length 4,096.

Evaluation protocol. Storage accumulated during forced exploration is seeded into evaluation runs, ensuring that the memory agent has access to a memory pool at inference time. Evaluation follows an online protocol, where the experiential memory base is updated on-the-fly as the agent system processes a continuous stream of tasks. The task agent is capped at 15 steps per task on GAIA and 40 steps on WebWalkerQA and xBench-DS, applied uniformly across all methods.

Baselines. We compare MemAgent against different memory methods. (i) **No Memory** runs the task agent without any memory augmentation. (ii) **Random Routing** replaces the memory agent with random routing decisions. (iii) **MemAgent w/o training** directly employs the Qwen3.5-4B model. (iv) **Individual Memory Methods** evaluate each of the 13 long-term memory providers in isolation. (v) **PromptRoute** replaces the SFT-trained memory agent with GPT-5-mini, utilizing a prompt that includes per-provider statistics, routing rules, and representative cases mined from forced-exploration trajectories (see Appendix C.2). (vi) **MemAgent (5 providers)** restricts our agent to a curated subset (ExpeL, AWM, SkillWeaver, Voyager, and InsightGraph) selected for cross-benchmark complementarity, thereby isolating the effect of the provider pool size. (vii) **Oracle** chooses the successful provider for each task when available, defining the upper bound for routing. (viii) **All-Concat** retrieves from all memory providers at the BEGIN phase and concatenates their outputs. (ix) **All-Fuse** performs the same memory retrieval but uses an LLM to consolidate the retrieved content into a single memory note.

5.2 Main Results

Table 1 presents the results across all three benchmarks. MemAgent achieves consistent gains over the no-memory baseline, averaging a 10.0% improvement overall. Crucially, MemAgent surpasses every single-provider baseline on all benchmarks: +4.9% on GAIA (vs. AWM), +2.3% on WebWalkerQA (vs. ExpeL), and +1.0% on xBench (vs. SkillWeaver). While different individual methods dominate specific benchmarks, MemAgent consistently outperforms them all. This validates that learned routing effectively captures complementary strengths across the provider pool. On GAIA, these improvements remain consistent across all difficulty levels. To further disentangle the sources of these gains, we examine two controlled variants that isolate the roles of routing and of training. Replacing the memory agent with a random policy (Random Routing) fails to recover MemAgent's performance, indicating that the improvements arise from *learned* rather than arbitrary routing. MemAgent w/o training likewise falls short of MemAgent, demonstrating that the model's instruction-following ability alone is insufficient and that training is essential for effective routing.

PromptRoute. We investigate whether routing knowledge can be distilled into a prompt instead of learned via SFT. PromptRoute replaces the SFT-trained agent with a frozen LLM, optimizing its prompt using a Decision Knowledge Base (DKB). This DKB extracts performance statistics, routing rules, and representative examples from forced exploration trajectories for inference-time injection. PromptRoute averages 70.9% across benchmarks (+5.2% over the no-memory baseline), proving the efficacy of LLM-mined knowledge. However, MemAgent still outperforms it, confirming that SFT captures finer-grained routing distinctions than prompt-based retrieval alone. We view LLM-mined prompt optimization as a promising direction for future work.

Table 1: Accuracy (%) on GAIA, WebWalkerQA, and xBench-DS. Gray deltas show percentage-point changes vs. the no-memory baseline. GAIA results are detailed by difficulty levels ($L1$–$L3$). **Bold** and underline denote the best and second-best memory methods.

Method	GAIA				WebWalkerQA	xBench-DS	Avg.
	Overall	Level 1	Level 2	Level 3			
No Memory	58.8	73.6	54.7	42.3	69.4	69.0	65.7
Individual memory methods							
Agent-KB (Tang et al., 2025)	60.0 +1.2	73.6	54.7	50.0	70.0 +0.6	71.0 +2.0	67.0 +1.3
LightWeight (Zhang et al., 2026a)	59.4 +0.6	67.9	61.6	34.6	73.5 +4.1	73.0 +4.0	68.6 +2.9
ExpeL (Zhao et al., 2024)	61.8 +3.0	73.6	61.6	38.5	77.1 +7.6	74.0 +5.0	71.0 +5.2
SkillWeaver (Zheng et al., 2025)	63.6 +4.8	71.7	66.3	38.5	70.0 +0.6	77.0 +8.0	70.2 +4.5
Voyager (Wang et al., 2024)	64.2 +5.5	77.4	64.0	38.5	70.6 +1.2	68.0 −1.0	67.6 +1.9
InsightGraph (Ours)	61.8 +3.0	69.8	59.3	53.8	72.4 +2.9	75.0 +6.0	69.7 +4.0
DILU (Wen et al., 2024)	59.4 +0.6	69.8	61.6	30.8	71.8 +2.4	75.0 +6.0	68.7 +3.0
Generative (Park et al., 2023)	60.6 +1.8	75.5	58.1	38.5	72.4 +2.9	72.0 +3.0	68.3 +2.6
Memp (Fang et al., 2026)	61.8 +3.0	73.6	61.6	38.5	72.4 +2.9	70.0 +1.0	68.1 +2.3
Cheatsheet (Suzgun et al., 2026)	60.6 +1.8	69.8	59.3	46.2	70.6 +1.2	70.0 +1.0	67.1 +1.4
AWM (Wang et al., 2025c)	64.8 +6.1	75.5	64.0	46.2	68.8 −0.6	69.0 +0.0	67.5 +1.8
EvolveR (Wu et al., 2026)	60.6 +1.8	66.0	65.1	34.6	71.8 +2.4	71.0 +2.0	67.8 +2.1
Mobile-E (Wang et al., 2025b)	63.6 +4.8	81.1	62.8	30.8	74.1 +4.7	71.0 +2.0	69.6 +3.9
Routing methods							
Random Routing	58.2 −0.6	67.9	58.1	38.5	72.4 +3.0	71.0 +2.0	67.2 +1.5
MemAgent w/o training	63.6 +4.8	71.7	67.4	34.6	70.6 +1.2	70.0 +1.0	68.1 +2.4
MemAgent (5 providers)	63.6 +4.8	77.4	61.6	42.3	72.9 +3.5	73.0 +4.0	69.8 +4.1
PromptRoute	65.5 +6.7	77.4	64.0	46.2	71.2 +1.8	76.0 +7.0	70.9 +5.2
All-Concat	62.4 +3.6	79.2	61.6	30.8	74.1 +4.7	73.0 +4.0	69.8 +4.1
All-Fuse	63.6 +4.8	83.0	60.5	34.6	75.3 +5.9	74.0 +5.0	71.0 +5.3
MemAgent (Ours)	**69.7** +10.9	**83.0**	**67.4**	**50.0**	**79.4** +10.0	**78.0** +9.0	**75.7** +10.0
Upper bound							
Oracle (13 providers)	87.3 +28.5	94.3	87.2	73.1	89.4 +20.0	92.0 +23.0	89.6 +23.8

Table 2: Training-pipeline ablation on GAIA. Accuracy (%); Δ = drop relative to the full model.

Configuration	Accuracy (%)				Δ
	Ov.	L1	L2	L3	
Base (Judge all, no probe, no filtering)	60.6	75.5	61.6	26.9	–
w/o BEGIN correctness filtering	62.4	79.2	60.5	34.6	+7.3
w/o Probe score removal	63.0	77.4	64.0	30.8	+6.7
w/o IN threshold filtering	63.6	79.2	61.6	38.5	+6.1
w/o END downstream reward	66.7	75.5	**68.6**	42.3	+3.0
MemAgent (Full)	**69.7**	**83.0**	67.4	**50.0**	–

Table 3: Inference-architecture ablation on GAIA. Accuracy (%); selective stores to chosen providers, all-store to all 13 providers.

Begin	In	End	Accuracy (%)			
			Ov.	L1	L2	L3
✗	✗	✗	58.8	73.6	54.7	42.3
✗	✓	✗	63.0	77.4	64.0	30.8
✓	✓	All-store	65.5	81.1	64.0	38.5
✓	✓	**Selective**	**69.7**	**83.0**	**67.4**	**50.0**

Provider pool size. MemAgent (5 providers) restricts routing to a curated subset of five complementary providers. This variant achieves 69.8% overall, trailing the full 13-provider system. Although it outperforms the no-memory baseline and rivals most single-provider methods, this gap underscores the importance of provider diversity. Excluded providers (*e.g.*, LightWeight, Memp, DILU, and Mobile-E) evidently offer complementary strengths that the full routing model successfully exploits.

Retrieving from all providers. Retrieving and combining memories from every provider can introduce redundant, irrelevant, or conflicting context. The all-provider baselines in Table 1 test whether broad coverage alone can replace selective routing, and support selecting provider-specific content rather than indiscriminate aggregation.

5.3 Analysis and Discussion

Ablation Studies. We conduct two systematic ablation studies to isolate the contributions of the training data synthesis pipeline and the three-phase inference architecture. Table 2 presents a leave-one-out ablation on GAIA. Compared to a *Base* configuration using heuristic LLM-as-Judge rewards (without probing or filtering), our four components yield a combined +9.1% improvement. BEGIN correctness filtering is the most impactful (+7.3%); using ground-truth outcomes rather than proxy scores is critical for harder tasks (boosting L3 from 34.6% to 50.0%). Removing probe scores (+6.7%)

Table 5: **Inference overhead of MemAgent on GAIA.** Injection sizes count memory tokens added to each task-agent prompt. Routing latency is MemAgent's extra wall-clock cost per task.

Component	Per-task cost	Notes
Memory injection (tokens inserted into context):		
BEGIN (long-term retrieval)	∼370 tokens	median 268, max 2.2K
IN (short-term, per step)	∼210 tokens	3.8 steps/task triggered
Total injected per task	∼1.2K tokens	<0.5% of task tokens
Routing cost (MemAgent-specific):		
Probe (13 providers, parallel)	∼75 ms	one-time at BEGIN
Memory agent inference (4B)	∼80 ms/decision	∼9 decisions/task
Total routing latency	∼1.2 s/task	<0.3% of task time
Task efficiency:		
Avg. steps (No Memory → MemAgent)	9.1 → 8.0 steps	−12%

prevents the agent from exploiting incompatible metric scales (*e.g.*, raw counts vs. cosine similarities) to naively select the highest-scoring provider. IN-phase threshold filtering curbs short-term memory over-querying, which otherwise degrades long-horizon performance. Finally, the downstream END reward (+3.0%) optimizes storage decisions by measuring actual memory reuse.

Table 3 ablates the three-phase inference by varying whether long-term memory is retrieved, whether short-term memory is injected, and whether storage is selective or indiscriminate. The IN-only configuration (rule-based short-term memory injection without long-term retrieval or storage) achieves a 4.2 % gain over the baseline, confirming the value of within-task working memory. The force-all-store ablation, which stores every trajectory across all providers, achieves only 65.5%. This occurs because indiscriminate storage pollutes the retrieval pools with low-relevance memories. This highlights the necessity of selective END-phase routing; without it, the noise from stored low-quality memories offsets the benefits of long-term memory.

Task-Agent Transfer. We replace GPT-5-mini with GPT-5.6 Luna and transfer the memory agent without retraining. Table 4 shows that MemAgent outperforms both the no-memory setting and the best fixed provider overall and at every difficulty level, indicating that learned routing remains effective across task-agent backbones. Appendix C.1 further examines the effect of router capacity.

Table 4: Zero-shot transfer to GPT-5.6 Luna on GAIA.

Method	Overall	L1	L2	L3
No Memory	67.9	77.4	68.6	46.2
AWM (best provider)	69.7	77.4	68.6	57.7
MemAgent (zero-shot)	**71.5**	**79.2**	**69.8**	**61.5**

Inference Overhead. A natural concern is the computational cost of memory-augmented inference; Table 5 details this per-task overhead on GAIA. The volume of memory text injected into the task agent's context is minimal, totaling ∼1.2K tokens per task (less than 0.5% of the task agent's average tokens per task). While injection overhead is inherent to all memory-augmented approaches, MemAgent's injection size remains moderate compared to 13 other providers, such as ExpeL at ∼970 tokens (see Appendix C.4). MemAgent's unique routing cost involves parallel probing and inference via a 4B-parameter model. This step adds only 1.2 seconds per task, or less than 0.3% of average latency. Crucially, memory augmentation actually *reduces* the overall number of task steps. MemAgent requires an average of 8.0 steps compared to 9.1 for the no-memory baseline (a 12% reduction). This efficiency gain is most pronounced on harder tasks, where L3 steps drop from 12.3 to 10.7, as retrieved memories help the agent avoid unproductive exploration.

6 CONCLUSION

In this paper, we revisit agent memory from the perspective of heterogeneity and introduce MemAgent, a three-phase routing framework. Its content-aware architecture and data-synthesis pipeline supervise long-term retrieval, short-term memory use, and selective storage across providers. Across three benchmarks, MemAgent improves average accuracy by 10.0% and outperforms single-provider baselines while reducing task steps with minimal routing overhead. Overall, our results support adaptive, content-aware orchestration throughout the memory lifecycle.

AI USE STATEMENT

Generative AI tools were used for language editing, including grammar, spelling, word choice, and formatting assistance. They were not used to generate research ideas, design experiments, analyze experimental results, or determine scientific claims. All AI-assisted edits were reviewed by the authors, who take full responsibility for the final content. The use of language models as components of the proposed method and evaluation pipeline is described explicitly in the main paper and appendix.

ETHICS STATEMENT

This work does not involve human subjects or the collection of private user data. The evaluated agents operate on public benchmark tasks and web content. Persistent agent memory may retain sensitive or unreliable information in real deployments; practical systems should therefore apply access control, data minimization, provenance tracking, and deletion mechanisms before storing retrieved content.

REPRODUCIBILITY STATEMENT

The complete memory-provider pool, training-data synthesis procedure, routing prompts, hyperparameters, evaluation protocol, and additional analyses are documented in the appendix. The main paper reports the task agent, memory-agent backbone, tools, step budgets, and benchmark settings used for all comparisons. The code has been open-sourced.

REFERENCES

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of ACL*, 2026.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Wayne Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, Kuan Li, Minpeng Liao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Iterresearch: Rethinking long-horizon agents with interaction scaling. In *ICLR*, 2026.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, et al. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations. *arXiv preprint arXiv:2506.13651*, 2025.

Yiming Du, Baojun Wang, Yifan Xiang, Zhaowei Wang, Wenyu Huang, Boyang Xue, Bin Liang, Xingshan Zeng, Fei Mi, Haoli Bai, et al. Memory-t1: Reinforcement learning for temporal reasoning in multi-session agents. In *ICLR*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of ACL*, 2026.

Joseph Fioresi, Parth Parag Kulkarni, Ashmal Vayani, Song Wang, and Mubarak Shah. Learning to share: Selective memory for efficient parallel agentic systems. In *ICML*, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *ICLR*, 2025.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. In *ICLR*, 2026a.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2026b.

Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. Cam: A constructivist view of agentic memory for llm-based reading comprehension. In *NeurIPS*, 2025a.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In *EMNLP*, 2025b.

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. In *ICLR*, 2024.

Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory. In *ICLR*, 2026.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *ICLR*, 2024.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *ICLR*, 2026.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *ACM UIST*, 2023.

Tianrui Qin, Qianben Chen, Sinuo Wang, Xinyu Yang, Kang Zhu, He Zhu, Dingfeng Shi, Xinxin Liu, Ge Zhang, Jiaheng Liu, Xitong Gao, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Flash-searcher: Fast and effective web agents via DAG-based parallel execution. In *ICLR*, 2026.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *ICLR*, 2025.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, et al. Taskcraft: Automated generation of agentic tasks. In *ICLR*, 2026a.

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context llm agents. In *ICLR*, 2026b.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Yiheng Shu, Saisri Padmaja Jonnalagedda, Xiang Gao, Bernal Jiménez Gutiérrez, Weijian Qi, Kamalika Das, Huan Sun, and Yu Su. Remem: Reasoning with episodic memory in language agent. In *ICLR*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *EACL*, 2026.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025a.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *ICML*, 2025c.

Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao Ma, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. DiLu: A knowledge-driven approach to autonomous driving with large language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=OqTMUPuLuC.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhenglin Wang, Zhengwei Tao, Ding-Chu Zhang, Zekun Xi, Xiangru Tang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webdancer: Towards autonomous information seeking agency. In *NeurIPS*, 2025a.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, et al. Webwalker: Benchmarking llms in web traversal. In *ACL*, 2025b.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. From interactions to principles: Experience-driven self-distillation for evolving llm agents. In *ICML*, 2026.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Ding-Chu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025c.

Keyi Xiang, Zeyu Feng, Zhuoyi Lin, Yueming Lyu, Shi Boyuan, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Flowsearcher: Synthesizing memory-guided agentic workflows for web information seeking. In *ICLR*, 2026.

Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to continually learn via meta-learning agentic memory designs. In *ICLR Workshop on Memory for LLM-Based Agentic Systems*, 2026.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Wenlin Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, and Tong Xu. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. In *ICLR*, 2026.

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

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

Yilun Yao, Shan Huang, Elsie Dai, Zhewen Tan, Zhenyu Duan, Shousheng Jia, Yanbing Jiang, and Tong Yang. Arc: Active and reflection-driven context management for long-horizon information seeking agents. In *Findings of ACL*, 2026.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Siheng Chen, and Yong Jiang. AgentFold: Long-horizon web agents with proactive context folding. In *ICLR*, 2026.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. In *ICLR*, 2026a.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *ACL*, 2026b.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. In *NeurIPS*, 2025a.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. In *ICML*, 2026a.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026b.

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

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 2025b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *AAAI*, 2024.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *NeurIPS*, 2023.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *AAAI*, 2024.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, et al. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026a.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=XY8AaxDSLb.

APPENDIX CONTENTS

A Implementation Details 15
- A.1 Memory Provider Pool 15
- A.2 Dataset Details 15
- A.3 Training Details 16
- A.4 Evaluation Details 16
- A.5 Provider Implementation Parameters 16

B Method Details 18
- B.1 Inference Pipeline 18
- B.2 LLM-as-Judge Reward Design 18
- B.3 InsightGraph 20
- B.4 Short-Term Memory Provider 21
- B.5 Decision Prompt Format and Output Schema 22

C Further Analyses 26
- C.1 Router Capacity 26
- C.2 PromptRoute: Decision Knowledge Base 26
- C.3 Routing Behavior 27
- C.4 Injection Overhead 29

D Why Do Memory Providers Exhibit Complementary Behavior? 29
- D.1 Three Representational Families and Their Failure Modes 29
- D.2 A Qualitative Lens: Task Surface and Transfer Requirements 30
- D.3 Case Studies: Complementary Transfer Patterns in Storage 31
- D.4 Implications 32

A Implementation Details

This section outlines the memory provider pool, dataset, training, and evaluation details for our experiments, followed by the implementation parameters for each provider.

A.1 Memory Provider Pool

Table 6 summarizes the comprehensive provider pool utilized by MemAgent. This pool comprises 13 heterogeneous long-term providers spanning exploration memory, experience learning, workflow induction, procedural memory, principle evolution, and graph-structured memory, alongside a short-term memory provider designed for within-task state aggregation. Notably, we introduce two custom providers to this framework: InsightGraph, a graph-structured long-term provider, and a rule-based short-term memory provider. Comprehensive implementation details for these novel components are available in Appendix B.3 and Appendix B.4.

Table 6: **Memory providers in MemAgent.** The pool comprises 13 heterogeneous long-term providers spanning diverse memory paradigms, plus a short-term memory provider for within-task state aggregation. Storage Strategy: the specific information extracted and saved from past trajectories; Retrieval Mechanism: the method used to query relevant memories for new tasks.

Provider	Memory Paradigm	Storage Strategy	Retrieval Mechanism
Voyager (Wang et al., 2024)	Exploration memory	Refined trajectory summaries with embeddings	Semantic search over task similarity
ExpeL (Zhao et al., 2024)	Experience learning	Dual-path: insights from all outcomes + refined trajectories from successes	Hybrid insight + trajectory retrieval
Generative (Park et al., 2023)	Scored memory pool	Task queries + trajectory summaries	Vector retrieval + LLM importance scoring
DILU (Wen et al., 2024)	Query-trajectory mapping	Query embeddings + trajectory summaries	Semantic search on query similarity
AWM (Wang et al., 2025c)	Workflow templates (SOPs)	LLM-induced abstract workflow patterns	Vector search over workflow templates
Mobile-E (Wang et al., 2025b)	Keyword-indexed tips	Tips (text) + Shortcuts (structured sequences)	Keyword-based retrieval
Cheatsheet (Suzgun et al., 2026)	Global best practices	Continuously updated cheatsheet of rules	LLM-curated synthesis
SkillWeaver (Zheng et al., 2025)	Procedural code memory	Trajectory → generalized Python functions	Skill matching → dynamic tool wrapping
Agent-KB (Tang et al., 2025)	Structured knowledge base	Full workflow decomposition into planning & experience fields	Student/Teacher guidance injection
Memp (Fang et al., 2026)	Procedural scripts	Process script induction / self-correction	Script similarity matching
EvolveR (Wu et al., 2026)	Evolving principles	Principle extraction + semantic merge/prune	Principle retrieval with examples
LightWeight (Zhang et al., 2026a)	Dual-layer factual memory	Strategic (planning) + operational (tool usage) extraction	Semantic search over both layers
InsightGraph (Ours)	Knowledge graph	LLM-extracted insight nodes with semantic edges	Hybrid search + graph expansion
Short-Term Memory (Ours)	Structured working memory	Rule-based extraction of facts, URLs, searches	Template-based sectioned output

A.2 Dataset Details

The training set comprises 173 tasks: 53 GAIA Level-1 tasks and 120 synthetic multi-hop tasks from TaskCraft (Shi et al., 2026a). Each TaskCraft task is a two-hop question covering domains such as academic research, financial reports, and scientific papers. These tasks require tool combinations like (web search, web search) or (image analysis, web search) (Liu et al., 2024; Wu et al., 2025a). An example is as follows:

> *"According to the January 9, 2025 special feature on Artsy, what did Clare Lilley forecast regarding diversity and craft in the art world?"* [web → web]

Following MemEvolve (Zhang et al., 2026a), the 53 GAIA L1 tasks are used both to populate the memory providers during forced exploration and to evaluate downstream reuse. We therefore treat GAIA L1 as an experience-reuse setting rather than a strict held-out generalization test. The router

itself is supervised only on provider selection, not on gold answers, and this protocol is intended to mimic deployment scenarios in which the memory pool has already been populated through prior interactions. By contrast, the remaining 112 GAIA L2/L3 tasks and all WebWalkerQA and xBench-DS tasks are disjoint from training and serve as held-out generalization benchmarks.

A.3 Training Details

We instantiate the memory agent with Qwen3.5-4B (Team, 2026) and fine-tune it with LoRA (r=16, α=32, dropout= 0.05) applied to all attention and MLP projections, yielding 0.84% trainable parameters. SFT minimizes cross-entropy loss on completion tokens only (prompt tokens are masked):

$$\mathcal{L}_{\text{SFT}} = -\frac{1}{N}\sum_{i=1}^{N} \log P_\theta(y_i \mid x_i) \tag{3}$$

where y_i are the completion tokens and x_i the corresponding input contexts. Training uses learning rate 2×10^{-5} with cosine annealing and 10% linear warmup, weight decay 0.01, effective batch size 64, maximum sequence length 4,096 tokens. We train for 3 epochs and select the final checkpoint via validation on a held-out subset of decisions. The downstream reward (Eq. (2)) uses saturation constant $H = 5$, and the END-phase SFT filter uses threshold $\epsilon_{\text{END}} = 0.5$.

A.4 Evaluation Details

Task agent and tools. The task agent is a single-agent deep research framework (Flash-Searcher (Qin et al., 2026)) powered by GPT-5-mini via API, equipped with four tools: `WebSearch`, `CrawlPage`, `VisualInspector`, and `AudioInspector`. At each step the agent either issues one tool call or emits a final answer. Within each benchmark, every method (no-memory, individual providers, random routing, PromptRoute, MemAgent) uses the same budget. If the step budget is reached before a final answer, the agent is prompted to answer from its context; the same correctness judge evaluates this output.

Correctness judging. For GAIA and WebWalkerQA, GPT-5-mini determines correctness by comparing the agent's final answer against the ground truth. This comparison relies on a strict equivalence rubric, requiring the exact same entity, an identical numeric value (regardless of formatting), or a semantically equivalent short phrase. For xBench-DS, we follow the benchmark's original protocol, employing Gemini-2.5-Flash as the equivalence judge. In all scenarios, the evaluating model remains completely independent of both the task agent and the memory agent to prevent self-grading artifacts.

A.5 Provider Implementation Parameters

All providers share a unified interface (`retrieve` and `store`) but differ in their internal representations and retrieval mechanisms. Table 7 summarizes these parameters. To prevent redundant loading, the 11 providers utilizing embeddings share a single 384-dimensional `all-MiniLM-L6-v2` encoder through a GPU-locked singleton.

Four providers require LLM calls during retrieval: Agent-KB for query refinement and guidance synthesis, LightWeight for memory selection and synthesis, Generative for per-candidate importance scoring (which falls back to cosine similarity if an LLM is unavailable), and Cheatsheet for curator synthesis. Conversely, Memp and EvolveR invoke LLMs exclusively during storage. Their retrieval paths rely entirely on cosine similarity search over embeddings without any generation step. Furthermore, all 13 providers necessitate LLM calls during storage to perform trajectory summarization, insight extraction, or skill generation.

Summary. For most providers, storage costs intentionally dominate retrieval costs because storage LLM calls are amortized across numerous subsequent retrievals. Additionally, the lightweight probe stage used for content-aware BEGIN routing completely bypasses the retrieval LLM for all 13 providers. By relying solely on embedding or keyword lookups, this stage ensures that probe latency remains bounded, regardless of a provider's full retrieval cost.

Table 7: Implementation parameters for all 13 long-term memory providers. "Emb.": uses a dense embedding model on the retrieval path; "LLM-R": requires an LLM call on the full retrieval path; k: retrieval top-k, where 3→1 denotes three dense candidates reranked by an LLM to one, and 2+2 denotes two tips plus two shortcuts. All 13 providers additionally invoke an LLM during storage. The content-aware BEGIN probe stage is embedding/keyword-only for all providers (no LLM).

Provider	**Emb.**	k	**LLM-R**	**Key Parameters**
Voyager (Wang et al., 2024)	✓	1	✗	Pure semantic search; returns 6-sentence max trajectory summary with hierarchical distillation
ExpeL (Zhao et al., 2024)	✓	3	✗	Hybrid search: TF-IDF 0.3, semantic 0.7; dual-path insight + trajectory retrieval
Generative (Park et al., 2023)	✓	3→1	✓	Semantic search returns 3 candidates; LLM importance scoring selects top-1; <150 words output
DILU (Wen et al., 2024)	✓	1	✗	Pure semantic search on query embeddings; ultra-concise step format (<150 words)
AWM (Wang et al., 2025c)	✓	1	✗	Pure embedding search; abstract workflow templates with entity generalization; <200 words
Mobile-E (Wang et al., 2025b)	✗	2+2	✗	Jieba keyword tokenization; 2 tips + 2 shortcuts; structured action sequences
Cheatsheet (Suzgun et al., 2026)	✓	1	✓	Global cheatsheet; LLM curator synthesizes from similar trajectories; <200 words per update
SkillWeaver (Zheng et al., 2025)	✗	3	✗	Keyword-based skill matching; safety validation blocks `exec/eval/import`; returns source code
Agent-KB (Tang et al., 2025)	✓	3	✓	Hybrid search weights: TF-IDF 0.5, semantic 0.5; structured fields: planning, experience
Memp (Fang et al., 2026)	✓	1	✗	Procedural scripts (<100 words) + concrete steps (<150 words); self-correction on failure
EvolveR (Wu et al., 2026)	✓	1	✗	Principle retrieval; merge threshold 0.8; prune threshold 0.3; max 1 pos + 1 neg example
LightWeight (Zhang et al., 2026a)	✓	3	✓	Dual-layer: max 30 strategic + 30 operational memories; short-term buffer size 20
InsightGraph (Ours)	✓	5	✗	Graph expansion (see Appendix B.3); 500-char output cap

B METHOD DETAILS

This section describes the end-to-end inference pipeline, the LLM-as-Judge reward design used for training data synthesis, the two memory providers we contribute (InsightGraph and the short-term memory provider), and the full decision prompt templates and output schemas.

B.1 INFERENCE PIPELINE

Algorithm 1 summarizes the end-to-end inference pipeline. For each task, the memory agent first probes all providers in parallel, routes to one for BEGIN retrieval, decides QUERY/NOOP for short-term memory at every step, and finally selects a subset of providers to store the completed trajectory.

Algorithm 1 MemAgent Inference Pipeline

Require: Task description q; memory agent $\mathcal{A}_M$; provider pool $\mathcal{P}$; task agent $\mathcal{A}_T$
Ensure: Task answer $\hat{y}_\tau$

— BEGIN: long-term memory routing —
$\{\text{probe}_i\} \leftarrow \text{PARALLELPROBE}(\mathcal{P}, q)$ ▷ snippet previews from all providers
$p^* \leftarrow \mathcal{A}_M(q,\ \text{SHUFFLE}(\{\text{probe}_i\}))$ ▷ content-aware provider selection
Inject p^*.RETRIEVE(q) into $\mathcal{A}_T$'s system prompt
— IN: short-term memory gating —
$\mathcal{M}_{\text{stm}} \leftarrow \emptyset$
for $t = 0, 1, \ldots, T$ **do**
  $\mathcal{M}_{\text{stm}} \leftarrow \text{UPDATESTM}(\mathcal{M}_{\text{stm}},\ a_{<t},\ o_{<t})$ ▷ extract facts, URLs, searches
  $d^{(t)} \leftarrow \mathcal{A}_M(q,\ t,\ \mathcal{M}_{\text{stm}},\ \{(a_i, o_i)\}_{i<t}^{\text{recent}})$ ▷ QUERY or NOOP
  if $d^{(t)} = \text{QUERY}$ **and** $\mathcal{M}_{\text{stm}} \neq \emptyset$ **then**
    Inject FORMAT($\mathcal{M}_{\text{stm}}$) into $\mathcal{A}_T$'s prompt
  end if
  $(a_t, o_t) \leftarrow \mathcal{A}_T$.STEP($t$) ▷ break if a_t is terminal
end for
$\hat{y}_\tau \leftarrow \mathcal{A}_T$'s final answer
— END: selective multi-store —
$\mathcal{S} \leftarrow \mathcal{A}_M(y_\tau,\ \text{SUMMARIZE}(\xi_\tau))$ ▷ select provider subset
for each $p_j \in \mathcal{S}$ **do** p_j.STORE(ξ_τ)
end for

B.2 LLM-AS-JUDGE REWARD DESIGN

The per-decision reward model uses an LLM judge to evaluate each memory routing decision along phase-specific attribution dimensions. All dimensions are scored on a 0–5 integer scale and normalized to $[0, 1]$ before aggregation. Dimensions marked with $\downarrow$ are *inverted* (higher raw scores correspond to lower rewards). Table 8 summarizes all dimensions and their weights.

Scoring rubrics. Each dimension uses a 6-level rubric (0–5). Table 9 provides representative anchors across all phases. The general principle is consistent: a score of 0 indicates no effect (memory ignored, no progress, no missed information), 2–3 marks a borderline case, and 5 represents an unambiguous strong signal. For inverted dimensions (Noise, Missed Opportunity), the anchoring is reversed so that 0 corresponds to the best outcome.

Aggregation. For each decision, the scalar reward is computed as

$$r_d \;=\; \langle \mathbf{w}^{(\phi)}, \hat{\mathbf{s}} \rangle \;=\; \sum_{i=1}^{K_\phi} w_i^{(\phi)}\, \hat{s}_i, \quad \hat{s}_i = \begin{cases} s_i / s_{\max}, & \text{dim. } i \uparrow \\ 1 - s_i / s_{\max}, & \text{dim. } i \downarrow \end{cases}, \quad \mathbf{w}^{(\phi)} \in \Delta^{K_\phi - 1} \tag{4}$$

where $s_{\max} = 5$, $\phi \in \{\text{BEGIN}, \text{IN-QUERY}, \text{IN-NOOP}\}$ indexes the decision phase, K_ϕ is the number of scoring dimensions for phase ϕ, and $\mathbf{w}^{(\phi)}$ lies on the $(K_\phi - 1)$-simplex so that $r_d \in [0, 1]$. The judge outputs a JSON object with scores and a brief reasoning string. Parsing uses direct JSON decoding with a regex fallback for robustness.

Table 8: LLM-as-Judge scoring dimensions and weights by decision phase. Dimensions marked ↓ are inverted before aggregation: $\hat{s}_i = 1 - s_i/s_{\max}$.

Phase	Dimension	Weight	Dir.	Description
BEGIN	Utilization	0.6	↑	Did the task agent reference or use information from the retrieved memory in its subsequent actions?
	Relevance	0.4	↑	How relevant was the retrieved memory to the specific task?
IN (QUERY)	Helpfulness	0.4	↑	Did the agent use information from the injected short-term memory?
	Progress	0.4	↑	Did the injection help advance the task?
	Noise	0.2	↓	Did the injection distract or confuse the agent?
IN (NOOP)	Appropriateness	0.5	↑	Was skipping retrieval a reasonable choice given the current state?
	Progress	0.3	↑	Did the task continue to make progress without memory retrieval?
	Missed Opportunity	0.2	↓	Did skipping retrieval cause the agent to miss important information?

Table 9: Representative rubric anchors for selected dimensions. All dimensions follow a 0–5 integer scale. Dimensions marked ↓ are inverted before aggregation.

Phase	Dimension	0 (lowest)	2–3 (borderline)	5 (highest)
BEGIN	Utilization	Memory completely ignored	Some overlap, likely coincidental	Agent directly built upon specific memory content
BEGIN	Relevance	Completely irrelevant domain	Somewhat related, not specifically useful	Perfectly targeted, critical information
IN (Query)	Helpfulness	No reference to injected content	Minor overlap, uncertain attribution	Agent directly leveraged specific injected details
IN (Query)	Noise ↓	No distraction	Some off-topic exploration triggered	Agent clearly confused, pursued wrong direction
IN (NoOp)	Appropriateness	STM had highly relevant content that should have been retrieved	Borderline; STM has marginal content	STM empty or clearly irrelevant; NoOp optimal
IN (NoOp)	Missed Opp. ↓	Nothing valuable missed	Missed non-critical information	Missed critical information; retrieval would have been breakthrough

Judge context windows. To control cost and focus the judge's attention, input fields are truncated: task description (500 char for BEGIN, 300 for IN), retrieved memory content (800 char), subsequent actions (2000 char for BEGIN), and before/after actions (400 char each for IN).

B.3 INSIGHTGRAPH: ADAPTIVE GRAPH MEMORY

InsightGraph is a graph-structured long-term memory provider designed to address two limitations of flat-list retrieval: the absence of relational structure between co-occurring strategies, and the inability to perform multi-hop associative recall. We describe its representation, retrieval mechanism, and maintenance protocol below.

B.3.1 GRAPH REPRESENTATION

Memories are organized as a typed knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{V}$ stores a concise, actionable insight extracted from a successful trajectory (*e.g.*, *"For spreadsheet tasks, call inspect_file_as_text rather than attempting web download"*), together with metadata linking it to its source task. Nodes are deduplicated using content-based signatures.

Edges capture two complementary relations. *Same-task* edges (w=1.0) link insights co-occurring within one trajectory, encoding the intuition that strategies applied together to a single task are likely jointly relevant. *Similar-concept* edges link semantically related insights *across* trajectories: after a new node is inserted, its sentence embedding is compared against all existing nodes, and bidirectional edges are created for pairs exceeding a cosine similarity threshold $\tau_{\text{sim}} = 0.55$, with initial weight set to the similarity score. Each edge additionally maintains usage and success counters (n_e, k_e) for online weight adaptation.

B.3.2 RETRIEVAL

Retrieval operates at the BEGIN phase and combines sparse lexical matching with dense semantic search and graph expansion.

Seed selection. Given a task query q, a hybrid score blends TF-IDF and embedding similarity:

$$s(v, q) = 0.2 \cdot \cos(\text{TF-IDF}_q, \text{TF-IDF}_v) + 0.8 \cdot \cos(\mathbf{e}_q, \mathbf{e}_v) \tag{5}$$

The dense term dominates because task queries are typically short and benefit from semantic generalization, while the sparse term handles exact entity matches (tool names, domain keywords). The top-5 nodes form the seed set $\mathcal{S}$.

Graph expansion. Seed scores are propagated to neighbors to surface associatively related insights that would be missed by direct retrieval:

$$s(u, q) \leftarrow \max\big(s(u, q),\ s(v, q) \cdot w_{\text{eff}}(e_{v \to u}) \cdot \gamma\big), \quad v \in \mathcal{S} \tag{6}$$

with decay $\gamma = 0.7$ and a maximum of 3 neighbors per seed. Single-hop expansion is sufficient in practice: the graph remains relatively sparse (∼50–200 nodes after typical training), and deeper expansion surfaces tangentially related content.

Composition. Nodes scoring below $\tau_{\min} = 0.22$ are discarded. The top-3 survivors are concatenated into a guidance text. No LLM call is made on the retrieval path, keeping latency negligible.

B.3.3 ADAPTIVE EDGE WEIGHTS

Edge weights reflect empirical co-utility, defined as the frequency with which traversing an edge leads to a successful task outcome. This approach ensures that weights do not rely solely on the initial semantic similarity estimate. Specifically, we blend a semantic prior w_0 with a Bayesian quality term as follows:

$$w_{\text{eff}}(e) = \alpha_e \cdot w_0(e) + (1 - \alpha_e) \cdot \frac{k_e + 1}{n_e + 2}, \qquad \alpha_e = \frac{1}{1 + n_e/\kappa} \tag{7}$$

The quality term employs Laplace smoothing with a Beta$(1, 1)$ prior. It starts at 0.5 for unused edges and converges to the empirical success rate k_e/n_e as evidence accumulates. A half-life parameter,

$\kappa = 5$, controls the blending coefficient α_e. This coefficient transitions from being dominated by the prior during a cold start ($\alpha_e \approx 1$) to being dominated by experience after approximately 15 uses ($\alpha_e \approx 0.25$). This dual-phase behavior prevents the premature pruning of promising but underexplored edges while reducing the weight of thoroughly tested, poorly performing edges. The system updates these weights after each task and clamps them to the range $[0.1, 1.0]$. To prevent overfitting from a single observation, updates only occur when $n_e \geq 2$.

B.3.4 Storage and Consolidation

Upon successful task completion, an LLM extracts 2–4 concise insights ($\leq$30 words each, imperative form) from the trajectory. Each insight becomes a new node; same-task and similar-concept edges are created as described above.

To bound graph growth, periodic consolidation runs every 50 tasks: near-duplicate nodes (cosine similarity ≥ 0.85) are merged, and poorly performing similar-concept edges ($n_e \geq 5$, success rate $< 20\%$) are pruned. This keeps the graph compact while retaining empirically validated associations.

B.4 Short-Term Memory Provider

Prior work on memory-augmented agents focuses primarily on long-term, cross-task memory. We contribute a complementary *short-term* provider that aggregates within-task working state. The task agent already has access to its full conversation history, but cannot efficiently derive cross-step summaries (*e.g.*, *which URLs have been visited?*, *what facts have been verified so far?*) by scanning raw tool-call logs. The short-term provider is designed to fill this gap under three constraints: extraction is rule-based, so the retrieval path has deterministic behavior and negligible latency and makes no LLM calls; the output does not duplicate raw observations already present in the conversation history; and the provider adopts a conservative display policy that is intended to reduce low-information injections when insufficient state has accumulated.

B.4.1 Extraction and State Management

At each step, a rule-based extractor parses the structured tool-call record and updates three bounded data structures:

1. **Verified facts** (capacity: 15 active). We employ five regular expression patterns to extract common fact formats. These include key-value pairs with numeric values (*e.g.*, "Population: 21.54 million"), explicit answer statements, founding dates, geographic coordinates, and "X is Y" assertions. This extraction process yields a maximum of three facts per step. To prevent redundancy, a word-level Jaccard similarity threshold (> 0.6) filters out near-duplicates. Finally, if the memory capacity is exceeded, the oldest facts are evicted.
2. **URL history** (capacity: 30). Each visited or discovered URL is recorded alongside its associated tool, step index, and outcome (success or error). The eviction policy prioritizes unvisited URLs (FIFO), followed by the oldest entries overall.
3. **Search history** (capacity: 20). Each search query is logged with its step index and result count, employing a FIFO eviction policy.

Error detection runs in parallel. Fourteen compiled regex patterns classify tool failures into specific categories (*e.g.*, access denied, not found, timeout, rate limit, and no results). To avoid false positives from normal page content, this process scans only the first 500 characters of each observation.

B.4.2 Output Formatting

The formatter renders working memory into up to three prioritized sections (Table 10): Collected Data (≥ 2 facts to display), Pages Visited (≥ 3 URLs), and Recent Searches (≥ 4 queries). Under the conservative display policy, when all sections fall below their minimum thresholds the provider returns an empty response rather than rendering sparse content. When the character budget is exceeded, truncation occurs at the last complete section boundary so that no section is partially rendered.

Table 10: Short-term memory output sections, display thresholds, and capacity limits.

Section	Min to Show	Max Items	Priority
Collected Data	2 facts	8	Highest
Pages Visited	3 URLs	15	Medium
Recent Searches	4 queries	8	Lowest

B.4.3 Worked Example

We illustrate the formatter's behavior on a small input trace.

Input (3 steps).

`Step 1` — `web_search("country X population 2024")` → observation containing `"Population: 21.54 million (source: wikipedia.org)"`
`Step 2` — `crawl_page("https://en.wikipedia.org/wiki/X")` → observation containing `"GDP growth: 3.2%"`
`Step 3` — `web_search("country X GDP 2024")` → 3 results (no new extractable fact)

Output (rendered).

```
» COLLECTED DATA:
- Population: 21.54 million (source: wikipedia.org)
- GDP growth: 3.2%
:: PAGES VISITED: (suppressed; 1 URL < 3 threshold)
?? RECENT SEARCHES: (suppressed; 2 queries < 4 threshold)
```

Here Collected Data meets its 2-fact threshold and is displayed. The Pages Visited and Recent Searches sections each fall below threshold and are suppressed, so the final output contains only the verified facts. Had only one fact been extracted with no URLs and one search, all three sections would fall below threshold and the provider would return an empty response, deferring entirely to the agent's conversation history rather than injecting sparse content.

B.5 Decision Prompt Format and Output Schema

All three decision prompts share a common structure comprising a role instruction, a provider description guide, phase-specific context fields, decision rules, and a JSON-only output constraint. These prompts remain identical during both training and inference. The three phases differ in their input fields, output schemas, and allowed action sets.

During the BEGIN phase, the prompt includes the task description, the retrieval query, a randomly shuffled provider guide, and shuffled probe results from all 13 providers with their scores removed. Here, the action is strictly constrained to `query`. The agent must output `{"action": "query", "provider": <one of 13>}` and is explicitly instructed not to answer the task itself. For the IN phase, the prompt contains the task question, the current step index, a short-term memory summary, and recent tool-call observations. The agent makes a binary decision formatted as `{"action": "query"|"no_op", "provider": "short_term_memory"|null}`. It prefers `query` when the short-term memory holds useful, otherwise inaccessible content, and defaults to `no_op` otherwise.

In the END phase, the prompt provides the task outcome, a trajectory summary, and the provider guide annotated with outcome gates. The agent outputs `{"selected_providers": [<subset of 13>]}`, which typically includes three to six providers, though an empty list is permitted. Finally, Table 11 details the provider guide. This guide serves as the definitive reference for each provider's storage and retrieval behavior during the BEGIN and END decisions.

B.5.1 Full Decision Prompts

We provide the verbatim prompt templates used at inference. Placeholders in braces (*e.g.*, `{task_question}`) are substituted at runtime.

Table 11: Provider descriptions injected into the memory agent prompts. The BEGIN prompts incorporate both storage and retrieval descriptions, whereas the END prompts include only the storage description accompanied by an outcome gate annotation.

Provider	Storage Description	Retrieval Description	Gate
LightWeight	Extracts strategic (planning/decisions) and operational (tools/techniques) memories after task success	Provides long-term strategic/operational experience	Success
Agent-KB	Summarizes trajectories into structured fields: Agent Planning, Search Agent Planning, Agent Experience	Student Guidance (reference plans) and Teacher Guidance (lessons learned)	Success
SkillWeaver	Analyzes trajectories to identify core algorithms, generates generalized Python functions	Matches relevant skills, dynamically wraps them as tools	Success
ExpeL	Extracts Insights (Tips or Warnings) from all outcomes; stores refined trajectories on success	Hybrid retrieval of Insights and successful trajectories	All
Voyager	Generates trajectory summaries from all outcomes, stored as JSON with embeddings	Semantic search for most similar historical tasks	All
DILU	Stores query embeddings and trajectory summaries from both successes and failures	Semantic search on query similarity	All
Generative	Stores task queries and trajectory summaries	Vector retrieval + LLM importance scoring	All
Memp	Induces process scripts on success; triggers memory adjustment on failure	Retrieves the most similar process scripts	All
Cheatsheet	Maintains globally updated cheatsheet of best practices and rules	Uses LLM as Curator to synthesize/update the cheatsheet	All
Mobile-E	Stores Tips (text) and Shortcuts (structured sequences) for device automation	Keyword-based retrieval of Tips and Shortcuts	Tips: All; Shortcuts: Success
AWM	Induces abstract, general workflow templates from specific trajectories	Vector search for the most similar workflows	Success
EvolveR	Extracts principles with semantic merge/prune; periodically prunes low-value ones	Retrieves relevant principles with positive/negative examples	All
InsightGraph	Extracts concise insights as graph nodes with semantic edges; periodic consolidation	Hybrid TF-IDF + semantic search $\rightarrow$ graph expansion	Success

Listing 1: BEGIN phase decision prompt.

```
You are a Memory Decision Agent. Your ONLY task is to
output a JSON decision object.

[Available Memory Providers]
### lightweight_memory
- Storage: Extracts strategic and operational memories...
- Retrieval: Provides long-term strategic/operational experience.
### expel
- Storage: Extracts Insights (Tips/Warnings) from all outcomes...
- Retrieval: Hybrid retrieval of Insights and trajectories...
... (all 13 providers, randomly shuffled)

[Task Context]
- Task ID: {task_id}
- Phase: BEGIN
- Step: 0

[Task Description]
{task_question}

[Memory Request]
- Type: retrieval
- Query: {query}

[Memory Probe Results -- Actual Content Preview]
(Each provider was probed with your task query.)
### memp (42 items)
  Use web_search to find the target page, then crawl...
### expel (18 items)
  Insight: For spreadsheet tasks, use inspect_file...
... (all 13 providers, randomly shuffled, no scores)

[Decision Rules for RETRIEVAL]
- Phase is BEGIN: you MUST return "query" to retrieve
  relevant memories.

[CRITICAL INSTRUCTION]
You MUST respond with ONLY a valid JSON object.
Do NOT answer the task question.
You MUST choose a provider from the available list.

Output format:
{"action": "query", "provider": "<provider_name>"}

Your JSON response:
```

Listing 2: IN phase decision prompt.

```
[Task Context]
Task ID: {task_id}
Original Question: {task_question}
Current Phase: IN (Step {step_index})

[Short-Term Memory (Current Task)]
>> COLLECTED DATA:
  - Population: 21.54 million (source: wikipedia.org)
  - GDP growth: 3.2% (source: worldbank.org)
:: PAGES VISITED:
  - [OK] https://en.wikipedia.org/wiki/... (crawl_page)
  - [OK] https://data.worldbank.org/... (crawl_page)
?? RECENT SEARCHES:
  - "country X population 2024" (3 results)

[Recent Actions/Observations]
  1. [tool_call] web_search("country X GDP growth")
  2. [observation] Found 3 results: ...

[Available Memory]
  - short_term_memory: Structured working memory tracking
    cross-step verified facts, visited URLs, and search
    history.

[Decision Rules]
- Choose QUERY when:
  * Short-term memory has useful content.
  * The task agent would benefit from recalling earlier steps.
- Choose NO_OP when:
  * Short-term memory is empty.
```

```
  * The agent already has all necessary context visible.

[Decision Required]
1. QUERY: Retrieve structured state from short-term memory.
2. NO_OP: Skip memory injection for this step.

{"action": "query" | "no_op", "provider": "short_term_memory" | null}
```

Listing 3: END phase multi-store prompt.

```
You are a Memory Storage Decision Agent. Your task is to
decide which memory providers should store this trajectory.

[Task Result]: SUCCESS
[Task Query]: {task_question}

[Trajectory Summary]:
{trajectory_summary}

[Available Memory Providers]:
  - expel: extracts Insights from both successes/failures.
    [All outcomes]
  - lightweight_memory: stores strategic and operational
    experience. [Success-only]
  - skillweaver: converts trajectories into reusable Python
    functions. [Success-only]
  ... (all 13 providers, randomly shuffled)

[Decision Guidelines]:
1. You can select MULTIPLE providers (recommended 3-6)
2. If content is not valuable, return an empty list

[Output Format]:
{"selected_providers": ["provider1", "provider2", ...]}

Your JSON response:
```

B.5.2 LLM-AS-JUDGE PROMPTS

The LLM-as-Judge prompts assess the quality of each routing decision for training data synthesis. Here, we present the BEGIN prompt. The IN (QUERY) and IN (NOOP) prompts follow an analogous structure, incorporating their respective evaluation dimensions detailed in Table 8.

Listing 4: LLM-as-Judge prompt for BEGIN phase scoring.

```
You are a memory quality evaluator for an AI agent system.

A Memory Agent retrieved historical memory to help a Task
Agent solve a task. Evaluate whether this retrieval was useful.

## Task
{task_context}

## Retrieved Memory Content
{memory_content}

## Task Agent's Subsequent Actions
{subsequent_actions}

## Evaluation Criteria
Score each dimension from 0-5:

1. **Utilization** (0-5): Did the Task Agent reference or
   use information from the retrieved memory?
   - 0: Memory completely ignored
   - 2: Some overlap, likely coincidental
   - 4: Agent clearly referenced memory content
   - 5: Agent directly quoted or built upon memory

2. **Relevance** (0-5): How relevant was the retrieved
   memory to this specific task?
   - 0: Completely irrelevant (different domain)
   - 2: Somewhat related but not specifically useful
   - 4: Highly relevant with directly applicable strategies
   - 5: Perfectly targeted, critical information
```

```
## Output Format
{"utilization": <0-5>, "relevance": <0-5>,
 "reasoning": "<brief explanation>"}
```

B.5.3 DECISION OUTPUT FORMAT

The structured output of the memory agent differs across phases. During the BEGIN and IN phases, the agent generates a single-provider JSON object:

```
{"action": "query", "provider": "expel"}
{"action": "no_op", "provider": null}
```

For the END phase, the output is a JSON object containing a list of multiple selected providers:

```
{"selected_providers": ["expel", "voyager", "dilu",
                        "lightweight_memory", "agent_kb"]}
```

C FURTHER ANALYSES

This section provides complementary analyses: we detail the training-free PromptRoute baseline and its Decision Knowledge Base, dissect MemAgent's routing behavior across the three decision phases, and quantify the prompt-level injection overhead relative to individual memory providers. We additionally examine how router capacity affects performance.

C.1 ROUTER CAPACITY

We scale the memory-agent backbone from Qwen3.5-4B to Qwen3.5-9B while retaining the same training and evaluation protocol. Table 12 reports both untrained and fine-tuned variants on GAIA.

Table 12: Router-capacity analysis on GAIA. Accuracy (%).

Method	Overall	L1	L2	L3
Qwen3.5-4B w/o Training	63.6	71.7	67.4	34.6
MemAgent (Qwen3.5-4B)	69.7	**83.0**	67.4	50.0
Qwen3.5-9B w/o Training	63.6	75.5	61.6	46.2
MemAgent (Qwen3.5-9B)	**70.9**	81.1	**69.8**	**53.8**

Increasing router capacity yields a limited but consistent overall improvement after training: the 9B variant raises overall accuracy from 69.7% to 70.9% and Level-3 accuracy from 50.0% to 53.8%. The untrained 4B and 9B variants obtain the same overall accuracy, indicating that the gain arises from the interaction between capacity and routing supervision rather than model size alone.

C.2 PROMPTROUTE: DECISION KNOWLEDGE BASE

PromptRoute serves as a training-free baseline that replaces the SFT-trained memory agent with a frozen API LLM (GPT-5-mini). It augments the routing prompt with a Decision Knowledge Base (DKB), which functions as a structured JSON knowledge store. For each of the 13 providers, the DKB records empirical success rates, primary task-type affinities, and representative examples of successful and failed routing. Furthermore, it tailors information to specific phases. During the BEGIN phase, the DKB encodes routing rules that map task types to providers, subject to a minimum evidence threshold (≥ 3 occurrences and a win rate of $\geq 40\%$). For the IN phase, it supplies simple heuristics (*e.g.*, querying when the short-term memory contains content and skipping when it is empty). Finally, for the END phase, it outlines provider storage compatibility (*e.g.*, success-only versus all outcomes). At inference time, the relevant subset of the DKB is serialized and integrated into the routing prompt alongside the provider guide and probe results.

The DKB is initialized through a one-time bootstrapping process using scored trajectories generated during forced exploration. A keyword-based classifier assigns each task to one of eight categories

(*e.g.*, image analysis, web search, and calculation), after which provider-level statistics and routing rules are aggregated. This entire process requires no GPU training, as it relies solely on LLM API calls for the initial exploration. Ultimately, PromptRoute achieves a 65.5% accuracy on GAIA (+6.7% over the no-memory baseline) with zero training cost, reaching 94% of the performance of the trained model (69.7%). The remaining performance gap suggests that applying SFT on per-decision rewards captures finer-grained routing distinctions than prompt-based knowledge retrieval. Nevertheless, PromptRoute remains a promising alternative for future research.

C.3 Routing Behavior

We examine MemAgent's routing decisions across the three decision phases (BEGIN, IN, END) and the three benchmarks.

C.3.1 Phase-Specific Routing Distribution

BEGIN phase. The model routes across all 13 providers rather than concentrating on a single dominant option. The top four providers are Memp 22.6%, DILU 19.0%, Mobile-E 16.7%, and ExpeL 16.7%, with the remaining nine sharing the balance. All 13 providers are selected at least once, and no JSON parse failures occur on the evaluation set.

IN phase. Across the 165 GAIA evaluation tasks (comprising 1,102 IN-phase decisions in total), the memory agent selects QUERY on approximately 30.4% of the steps and NOOP on 69.6%. The high NOOP rate is largely driven by early steps. For instance, Step 0 defaults to NOOP because the short-term memory is empty at the start of a task. As short-term memory accumulates, the QUERY rate rises to roughly 55% to 71% by Steps 5 and 6 before eventually tapering off. The memory agent consistently selects NOOP whenever the short-term memory is empty or its contents are already reflected in the recent conversation.

END phase. MemAgent stores trajectories in three to eight providers rather than all 13. This selectivity is crucial, as indiscriminate storage pollutes retrieval pools with low-relevance memories. Upon a successful task outcome, storage includes both all-outcomes providers (*e.g.*, ExpeL, Voyager, and DILU) and success-only providers (*e.g.*, LightWeight, Agent-KB, and InsightGraph). Conversely, upon failure, the memory agent selects only all-outcomes providers.

C.3.2 Probe Scores

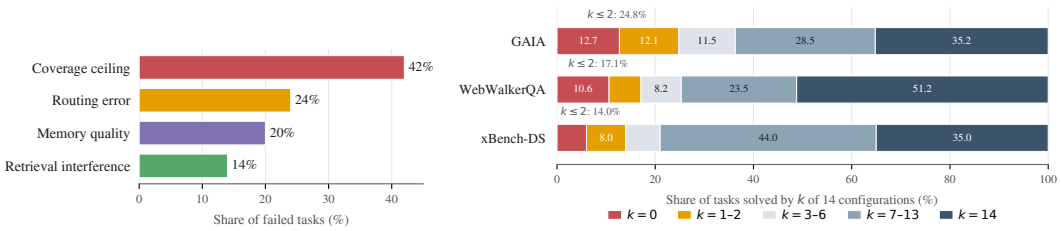
We observe that including numerical probe scores in the BEGIN prompt induces a highly skewed selection distribution. When these scores are present, 51.8% of selections are concentrated on just two providers, even though their score magnitudes are not directly comparable. Conversely, removing the scores and randomly shuffling the probe list yields a more uniform distribution with no loss in downstream accuracy. This behavior suggests that the agent inherently treats numerical scores as a direct ranking signal rather than a content preview; omitting them forces the agent to rely entirely on the retrieved content itself.

C.3.3 Provider Performance Specialization

Provider coverage and specialization across benchmarks reveal complementary strengths that motivate the routing problem.

Per-benchmark top performers.

- **GAIA**: AWM (+6.1pp), Voyager (+5.5pp), SkillWeaver (+4.8pp), Mobile-E (+4.8pp), ExpeL (+3.0pp)
- **WebWalkerQA**: ExpeL (+7.6pp), Mobile-E (+4.7pp), LightWeight (+4.1pp), InsightGraph (+2.9pp), Generative (+2.9pp)
- **xBench DeepSearch**: SkillWeaver (+8.0pp), DILU (+6.0pp), InsightGraph (+6.0pp), ExpeL (+5.0pp), LightWeight (+4.0pp)



(a) GAIA failure root causes.

(b) Task solvability and number of successful methods k.

Figure 5: **Failure analysis and task solvability across benchmarks.** Left: among failed tasks, coverage ceiling is the dominant failure source. Right: GAIA is more routing-critical than WebWalkerQA or xBench-DS, with more tasks solved by at most two of the 14 configurations.

No single provider is optimal across all three benchmarks. AWM dominates on multi-step GAIA reasoning; ExpeL leads on web navigation (WebWalkerQA); SkillWeaver specializes in code-heavy search tasks (xBench-DS). This heterogeneity motivates the routing problem: a fixed provider choice will underperform a learned provider-per-task policy on at least one benchmark.

Task coverage analysis. Table 13 illustrates how many of the 14 methods (13 memory providers and the no-memory baseline) individually solve each task. The three benchmarks exhibit distinct coverage profiles. GAIA is the most heterogeneous, featuring 12.1% routing-critical tasks (solved by only one or two providers) and 35.2% trivial tasks (solved by all 14). WebWalkerQA is skewed toward trivial tasks (51.2%), with only 6.5% classified as routing-critical. Conversely, xBench-DS contains the fewest routing-critical tasks (8.0%) and is dominated by a broad band of tasks solved by 7 to 13 providers (44.0%), indicating a moderate but widely covered difficulty level. Routing-critical tasks, where only a handful of providers succeed, represent the primary scenarios in which a memory agent can outperform any fixed choice.

Table 13: Task coverage distribution: the number of methods (13 memory providers + no-memory baseline) that solve each task.

Correct	**GAIA**	**WebWalkerQA**	**xBench-DS**
0 (unsolvable)	12.7%	10.6%	6.0%
1–2 (routing-critical)	12.1%	6.5%	8.0%
3–6	11.5%	8.2%	7.0%
7–13	28.5%	23.5%	44.0%
14 (all correct)	35.2%	51.2%	35.0%

C.3.4 Failure Analysis

Fig. 5a categorizes MemAgent's failure modes on GAIA tasks. The dominant cause is a *coverage ceiling* (42%), representing tasks that neither the 13 providers nor the no-memory baseline can solve due to missing tool support or an inability to answer. The remaining failures fall into three categories. First, *routing errors* (24%) occur when MemAgent selects an incorrect option despite one or two providers being capable of success. Second, *memory quality* issues (20%) arise when the chosen provider is appropriate but yields suboptimal retrieved content. Third, *retrieval interference* (14%) happens when injected memory degrades performance on tasks the baseline normally solves. This mirrors the force-all-store ablation in Table 3, demonstrating that indiscriminate storage harms performance.

Fig. 5b contextualizes these failures across benchmarks by binning tasks according to the number of successful providers (k). The ceiling-and-critical region ($k \leq 2$) is largest on GAIA (24.8%), followed by WebWalkerQA (17.1%) and xBench-DS (14.0%). On xBench-DS, 79% of tasks are solved by at least seven providers, meaning routing primarily redistributes performance across a broadly solvable pool. In contrast, routing on GAIA more frequently operates near the edge of feasibility.

C.4 INJECTION OVERHEAD

To quantify the context-length cost of memory augmentation, Table 14 reports the number of tokens injected into the task-agent context during the BEGIN phase, together with the resulting average trajectory length on the 165 GAIA tasks. This measurement isolates the prompt-level overhead incurred at task initialization, and excludes both the router's decoding cost and any method-specific insertions that occur in later phases.

Table 14: Added context size in the BEGIN phase and average trajectory length on GAIA. SkillWeaver is reported as "tools only" because it exposes retrieved skills as callable tools rather than injecting plain text, so a token count is not directly comparable.

Method	**Tokens injected**	**Avg. steps**	**Δ Steps**
No Memory	N/A	9.1	N/A
InsightGraph	57	8.6	-0.5
LightWeight	154	8.3	-0.8
Voyager	178	8.3	-0.8
DILU	184	8.3	-0.8
Generative	195	8.5	-0.6
Mobile-E	204	8.4	-0.7
Memp	286	8.6	-0.5
Cheatsheet	310	8.8	-0.3
AWM	341	8.7	-0.4
EvolveR	369	8.7	-0.4
Agent-KB	419	8.9	-0.2
ExpeL	971	8.4	-0.7
SkillWeaver	tools only	8.3	-0.8
MemAgent	**369**	**8.0**	$\mathbf{-1.1}$

All memory methods yield shorter average trajectories than the no-memory baseline, confirming that retrieved context helps the task agent reach a solution in fewer steps. However, this relationship is non-monotonic: incorporating more tokens does not reliably translate into fewer steps, suggesting that performance gains are driven by the *relevance* of the retrieved content rather than its sheer volume. MemAgent optimizes this trade-off, achieving the largest reduction in trajectory length (-1.1 steps, a 12% relative decrease from the baseline) while adding only about 369 tokens per task. This count is well below Agent-KB (419) and less than half of ExpeL (971). These results demonstrate that learned routing enhances context efficiency by selecting *useful* memories, rather than simply inflating the prompt size.

D WHY DO MEMORY PROVIDERS EXHIBIT COMPLEMENTARY BEHAVIOR?

No single memory provider dominates across the three evaluated benchmarks (Table 1). This section offers a qualitative lens on that complementarity: each provider exposes a distinct representation and retrieval contract, which makes different kinds of past experience available for transfer. We (i) organize providers by their indexable units and characteristic failure modes and (ii) illustrate these differences through concrete cases drawn from the populated memory stores.

D.1 THREE REPRESENTATIONAL FAMILIES AND THEIR FAILURE MODES

We group providers by their stored indexable units. A "memory" takes various forms: a trajectory summary in Voyager, a 30-word insight in InsightGraph, a callable function in SkillWeaver, or a running cheatsheet in Dynamic Cheatsheet. The inherent incomparability of these units necessitates a content-aware router. As summarized in Table 15, the following provider families each optimize for a different notion of cross-task reuse.

Table 15: **Three representational families.** Each family optimizes a different notion of cross-task reuse and, consequently, fails in different regimes.

Family	Members	Indexable unit
Similarity-retrieval	Voyager, DILU, Generative, Cheatsheet	Trajectory or query embedding
Structural-abstraction	AWM, SkillWeaver, Memp	Workflow template, executable function, or procedural script
Insight-accumulation	ExpeL, Agent-KB, InsightGraph, Mobile-E, EvolveR, LightWeight	Atomic, composable insight fragments

Similarity-retrieval. These providers index trajectories by semantic similarity of either the task query or the trajectory summary. They succeed when the new task's surface form closely matches a stored task (*e.g.*, repetitive "watch YouTube link → transcribe → extract fact" patterns). They fail in two regimes. *(i) Surface paraphrase masks logical change*: two tasks can read almost identically yet impose different constraints (*e.g.*, "albums *before* 2000" vs. "albums *between* 2000–2009"); nearest-neighbour retrieval returns the closest surface match even when the underlying constraint is wrong. *(ii) Domain shift with preserved structure*: when the new task's domain is unseen in storage, similarity retrieval cannot surface a relevant trajectory even if the reasoning skeleton matches a stored one. Dynamic Cheatsheet suffers an additional *monotonicity bias*: it rewrites a single global cheatsheet each retrieval, so its content is dominated by whichever task type appeared most recently and early-domain information is implicitly forgotten.

Structural-abstraction. These providers deliberately discard task-specific content and retain only the shape of a solution: AWM parameterizes workflows with variable names (`target_subject_page`, `revision_year`); SkillWeaver compiles generalized Python functions; Memp induces a "mental-model" script stripped of specific entities. They excel when two tasks share abstract computational logic despite unrelated surfaces. Their failure modes are complementary. *Over-generalization* – Memp's entity-stripped scripts lose the minutiae some tasks depend on. *Under-coverage* – SkillWeaver retains very few skills, so if the incoming task does not match an existing skill signature, retrieval returns nothing. *Missing warnings* – AWM ingests only successful trajectories; on tasks where the agent's natural tendency is a known pitfall, AWM has no mechanism to surface a caution.

Insight-accumulation. These providers decompose each trajectory into many small, reusable units; each unit is retrievable independently and composes with others. This is strongest on tasks whose success depends on *assembling* several atomic best-practices (*e.g.*, site-restricted search + cross-check against reference list + cite exact section). The failure modes are driven by retrieval geometry. *Lexical flooding*: with thousands of stored insights, many share common nouns ("Wikipedia", "search"), so top-k degenerates to generic advice and task-specific lessons drop below the cutoff. *Keyword brittleness*: Mobile-E uses pure keyword matching, so queries that do not share surface vocabulary with stored tips retrieve nothing. *Pruning removes negative evidence*: EvolveR discards low-scoring principles, but rare cautionary principles are exactly the ones that would save the agent on the rare task where they apply. *Graph pollution*: InsightGraph creates SIMILAR_CONCEPT edges aggressively, and a handful of generic insights (*e.g.*, "use site-restricted search") become hubs that dominate graph expansion regardless of the query.

D.2 A Qualitative Lens: Task Surface and Transfer Requirements

The failure modes above arise because each provider exposes a different unit of transfer. Similarity-retrieval providers favor memories whose task descriptions are close to the current query; structural-abstraction providers expose reusable procedures with task-specific details removed; and insight-accumulation providers retrieve smaller fragments that can be combined. None of these contracts is uniformly preferable: their utility depends on what the current task needs and on the content that the populated store can actually return.

This perspective motivates content-aware probing without assuming that the router explicitly recovers a latent logical form. Probe snippets reveal the concrete memory available from each provider for

the current task, allowing the router to compare provider outputs rather than relying only on static provider descriptions or task-level surface similarity.

D.3 Case Studies: Complementary Transfer Patterns in Storage

We illustrate these complementary transfer patterns with five cases drawn from the populated memory stores. The cases show how a provider-specific representation can expose useful content that another provider misses; Case 5 presents the reverse situation, in which surface proximity misleads a similarity retriever.

Case 1: Authoritative-PDF retrieval (InsightGraph bridges unrelated surfaces).

- **Task A**: *"In Emily Midkiff's June 2014 article in a journal named for one of Hreidmar's sons. . . , what word was quoted from two different authors in distaste for the nature of dragon depictions?"*
- **Task B**: *"What was the volume in m^3 of the fish bag calculated in the University of Leicester paper 'Can Hiccup Supply Enough Fish to Maintain. . . '?"*

Because the tasks share essentially no surface vocabulary, dense task-to-task retrieval yields no useful cross-task signals. Yet, InsightGraph connects them via a SIMILAR_CONCEPT edge between *"Use targeted site- and filetype-restricted searches combining author, date, and journal-name variants to locate authoritative PDFs"* (from Task A) and *"Use exact-title and site-restricted search queries (quotes and `site:`) to rapidly locate authoritative PDFs"* (from Task B). Although the two tasks have different surface vocabulary, both can reuse the same operational pattern: *identify a unique title → restrict search by site/filetype → extract one field from the PDF*. Task-level similarity retrieval does not expose this connection because its unit of comparison is the full task description. InsightGraph can expose it because its indexable unit is an atomic insight, and graph expansion propagates that insight from one task into the retrieval context of the other.

Case 2: Workflow reuse across heterogeneous domains (AWM succeeds; Voyager fails). A stored AWM workflow from a "Mercedes Sosa studio albums" training task abstracts the solution into a generalized skeleton: *define variables* {*artist_name, wiki_url, start_year, end_year*} → *retrieve page → parse structured list → filter by year range*. During evaluation, two GAIA tasks from entirely different domains (a Wikipedia-page-count question and a five-member World-of-Warcraft group composition question) both benefit from this workflow because they can reuse the same operation pattern: *open target page → parse structured list → filter by predicate → count*. In contrast, Voyager stores the literal narrative of the Mercedes Sosa trajectory ("I searched and crawled. . . planned a MediaWiki API query. . . "), which remains tied to the music domain; consequently, its top-1 retrieval for the WoW task yields an unrelated trajectory. This performance gap emerges precisely when the new task requires the *shape* of a workflow rather than the narration of a specific execution.

Case 3: Anti-pattern warning (ExpeL succeeds; AWM is silent). A GAIA task asks: *"How many edits were made to the Wikipedia page on Antidisestablishmentarianism. . . ?"*. The natural agent behavior is to copy the revision counter from the page's History UI, which overcounts because of pagination and soft-reverts. This failure mode appears in the training trajectories. ExpeL, which ingests both successes and failures, stored a warning:

> *"When a Wikipedia task asks for the number of 'edits' or 'revisions', verify whether the counter in the page history UI is paginated; do not rely directly on the history-page counter."*

AWM discards this kind of cautionary note by design (success-only ingestion), so its retrieval on this task cannot prevent the pitfall. The task is solved by ExpeL, InsightGraph, and Agent-KB, but not by AWM, Voyager, DILU, or the no-memory baseline.

Case 4: Executable-skill generalization (SkillWeaver's all-or-nothing payoff). SkillWeaver stored a Python function `find_adjacent_players(team, as_of_date)` that returns players with jersey numbers immediately before and after a target player on a given date. An evaluation task asking for "pitchers with the number before and after Taishō Tamai's number as of July 2023" matches this function signature *exactly*; the skill runs deterministically and returns the answer. A sibling Excel-map navigation task that asks for adjacent cells has no matching skill, and SkillWeaver

provides no useful memory. The case reveals the trade-off of executable-skill memory: when the abstract algorithm matches, the skill is strictly better than any workflow description because it *executes*; when it does not, the skill provides no benefit.

Case 5: Similarity trap (DILU misleads on near-paraphrase). The converse failure arises when two tasks share surface form but differ in the constraint they impose:

- **Stored trajectory**: *"How many studio albums did Mercedes Sosa release **between 2000 and 2009**?"*
- **Eval task**: *"How many studio albums did Mercedes Sosa release **before 2000**?"*

The two queries are near-paraphrases in cosine space. DILU returns the stored trajectory with high confidence, and the agent copies its year-range filter `[2000, 2009]` into the new query, yielding an off-by-decade answer. AWM, which exposes `{start_year, end_year}` as placeholders, avoids the trap because placeholders must be re-bound from the current query before execution. This case is the mirror of Cases 1–2: when surface is *too* close to structure, similarity retrieval overfits; structural abstraction always re-binds variables.

D.4 Implications

These cases illustrate three empirical properties of the provider pool. *(i)* A provider's representation and retrieval contract determine which parts of past experience are exposed for reuse. *(ii)* Surface similarity can be useful, but it can also miss transferable procedures or retrieve a near-paraphrase with incompatible constraints; structural abstractions and smaller insight units offer complementary alternatives. *(iii)* Content-aware probing lets MemAgent inspect what each populated provider can return for the current task and select among these complementary outputs. The analysis explains provider heterogeneity without assuming explicit latent-structure recovery.