Leveraging Cross-Domain Experience for Agentic Problem Solving

Anonymous ACL submission

Abstract

Existing AI agent frameworks accumulate rich execution experiences during problem solving, yet these experiences remain largely siloed within individual systems. Differences in action spaces, tool interfaces, and logging conventions prevent agents from reusing procedural knowledge discovered by other frameworks, limiting both generalization and systematic evaluation of agentic capabilities. We present AGENT KB, a framework-agnostic experience memory that enables cross-framework knowledge reuse without retraining or modifying base agents. AGENT KB abstracts heterogeneous agent trajectories into structured experience units and exposes them through lightweight interfaces. At inference time, a two-stage reuse process separates planning-level workflow guidance from execution-level feedback refinement, while a compatibility-aware integration mechanism mitigates harmful interference from incompatible experiences. We evaluate AGENT KB across multiple agent frameworks on GAIA, Humanity's Last Exam, GPQA, and SWE-bench, spanning both reasoning-intensive and execution-centric tasks. Across diverse agent-model combinations, AGENT KB consistently improves performance without additional fine-tuning. Ablation studies further indicate that staged reuse and refinement are critical, and that automatically distilled experiences perform comparably to manual curation. These results suggest that cross-framework experience abstraction constitutes a practical and scalable foundation for advancing agentic problem solving beyond individual frameworks.

1 Introduction

Modern AI agents have demonstrated impressive capabilities in complex reasoning and tool-mediated problem solving (Chan et al., 2024; Hong et al., 2024; Zhou et al., 2025). However, as agent systems proliferate, a striking limitation has emerged: agents repeatedly rediscover solutions and reproduce similar failures across frameworks, even when effective solutions already exist elsewhere. These failures are rarely random; rather, they reflect recurrent and procedural breakdowns that persist across tasks and environments. Despite accumulating large volumes of interaction data, agents remain unable to systematically reuse procedural knowledge discovered by other systems, revealing a fundamental gap in how agent experience is represented and reused.

A natural response is to equip agents with memory (Wang et al., 2025c; Ouyang et al., 2025). Existing memory mechanisms primarily strengthen individual agents through episodic storage or long-term context (Xu et al., 2025; Packer et al., 2023; Hu and Ying, 2025), or synthesize demonstrations tailored to specific frameworks and task schemas (Zheng et al., 2024; Tan et al., 2025). More recent efforts explore cross-task learning from interaction data, including Learn-by-Interact (Su et al., 2025) and A-Mem (Xu et al., 2025), but these approaches remain confined to single agent frameworks. As a result, even when a successful solution exists in another agent stack, it is rarely reusable in practice. Differences in logging formats, action ontologies, and tool interfaces prevent raw trajectories or retrieved examples from transferring cleanly across systems, limiting the effectiveness of standard RAG-style stores or per-agent scratchpads.

Enabling cross-framework experience reuse poses several interrelated challenges that no existing system fully addresses. First, *representation heterogeneity*: agent frameworks encode experience using incompatible logging conventions, action vocabularies, and tool interfaces, rendering raw trajectories non-transferable. Second, *context mismatch*: workflows that succeed in one framework may fail in another due to differences in tool availability, API signatures, or environmental constraints. Third, *knowledge interference*: unre-

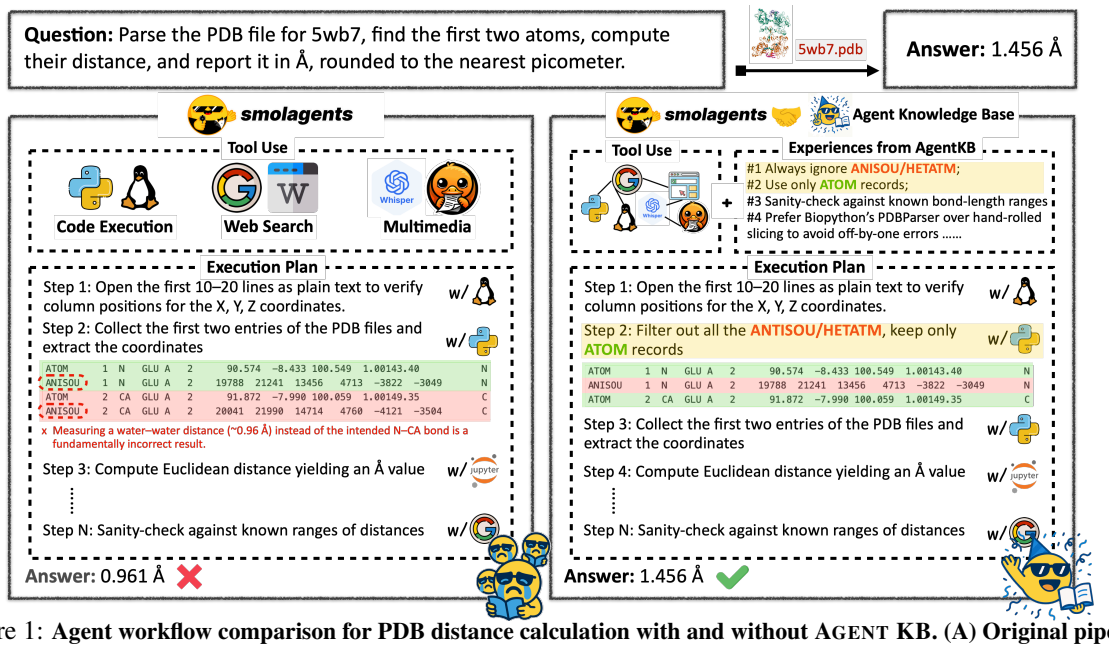

Figure 1: **Agent workflow comparison for PDB distance calculation with and without AGENT KB. (A) Original pipeline:** the agent reads the first two ATOM/HETATM/ANISOU records, often selecting solvent entries and reporting an incorrect O-H distance (0.961 Å). **(B) AGENT KB-enhanced workflow:** the agent applies reusable experience rules by filtering non-ATOM records, enforcing correct ATOM ordering, and validating N-CA bond-length ranges, thereby identifying the backbone N-CA pair and reporting the correct distance (1.456 Å).

stricted reuse of external experiences can degrade reasoning quality by introducing inconsistent plans or conflicting refinements, particularly in multi-step or tool-intensive tasks.

These challenges point to a deeper issue: experience reuse across agent frameworks fails not simply because information is unavailable, but because experience is reused without sufficient abstraction. Addressing this issue requires an experience representation that is sufficiently abstract to transfer across frameworks, yet sufficiently constrained to guide both high-level planning and low-level execution without introducing interference. This tension motivates a failure-aware design in which experience reuse must be selective, structured, and conservative, particularly in the presence of execution errors, tool mismatches, or partial plan failures. In the following, we instantiate this perspective by formulating cross-framework experience reuse as an abstraction problem and introducing a framework-agnostic experience memory to address it.

To address these failure modes, we formulate cross-framework experience reuse as an *abstraction problem* rather than a retrieval problem (Figure 1). Instead of replaying or retrieving raw trajectories, experience must be represented as a reusable yet constrained object that remains compatible across heterogeneous agent frameworks. We instantiate this formulation in AGENT KB, a cross-framework, plug-and-play experience memory that enables controlled and transferable knowledge reuse without retraining or modifying base agents. AGENT KB distills heterogeneous agent trajectories into structured experience units through *framework-agnostic abstraction* and exposes them via lightweight interfaces that integrate seamlessly with diverse agent systems, including smolagents (Zhu et al., 2025), OWL (Hu et al., 2025b), SWE-Agent (Yang et al., 2024), and OpenHands (Wang et al., 2025a). To accommodate novel action spaces, AGENT KB requires only a one-time mapping from native actions to an intent-level vocabulary (e.g., *search, execute, verify*), enabling a continuously growing repository of collective agent experience. To mitigate knowledge interference, AGENT KB employs a disagreement-aware integration mechanism that constrains how external experiences are incorporated, promoting conservative reuse across frameworks.

This work makes four contributions. (1) We formulate cross-framework experience reuse as an abstraction problem and instantiate it in AGENT KB, a *framework-agnostic, plug-and-play experience memory* that enables experience sharing across heterogeneous agent frameworks *without retraining or architectural modification*. (2) We introduce a lightweight *disagreement-gated refinement mechanism* that supports conservative integration of external experiences and mitigates reasoning interference. (3) We propose a two-stage reuse paradigm

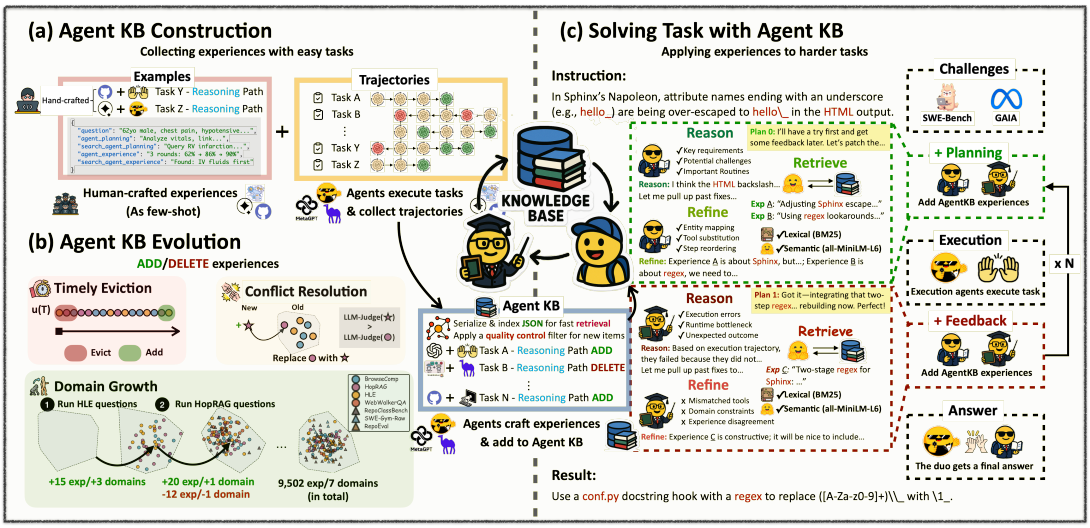

Figure 2: **End-to-end workflow of AGENT KB.** (a) **Construction**: heterogeneous agent trajectories and few-shot human seeds are abstracted into structured experiences and indexed in the AGENT KB. (b) **Evolution**: AGENT KB expands across domains through addition, conflict resolution, and timely eviction, maintaining quality while scaling. (c) **Solving tasks**: agents apply a two-stage `Reason`-`Retrieve`-`Refine` loop, planning with retrieved workflows and refining via feedback.

that separates *planning-level workflow abstraction* from *execution-level feedback refinement*, enabling transferable experience reuse under framework-specific constraints. (4) We present a systematic empirical study across GAIA (Mialon et al., 2024), Humanity's Last Exam (Bio/Chem)[1] (Skarlinski et al., 2025), GPQA (Rein et al., 2024), and SWE-bench (Jimenez et al., 2024), including controlled comparisons against a Naive-RAG baseline that isolates structured abstraction from retrieval augmentation alone.

Across benchmarks, AGENT KB consistently improves performance for diverse agent–model combinations. At `pass@1` (single attempt, equal compute), smolagents (`GPT-4.1`) improves from 55.2% to 61.2% on GAIA, versus 57.1% for Naive-RAG under the same retrieval budget, confirming that gains stem from structured abstraction rather than retrieval alone. With iterative passes, smolagents reaches 73.9% (`pass@3`); OpenHands gains 4.0 pp at `pass@1` on SWE-bench and 17.0 pp at `pass@3`; GPQA accuracy increases from 62.6% to 72.7%. These gains hold across both open-source (`Qwen`, `DeepSeek`) and proprietary (`GPT`, `Claude`) backbones. Ablation studies further show that automatically distilled experiences perform comparably to manual curation, supporting the scalability of AGENT KB as a shared experience memory.

[1] `https://huggingface.co/datasets/futurehouse/hle-gold-bio-chem`

2 Related Work

2.1 Agentic Memory Systems

Sumers et al. (2024) propose a cognitive architecture taxonomy distinguishing working, episodic, semantic, and procedural memory in language agents, providing the conceptual vocabulary that situates most memory systems. Agentic memory systems have evolved from simple storage mechanisms to structured architectures that support long-horizon reasoning and decision making (Zhang et al., 2025a). Early work primarily encoded experience implicitly in latent representations (Wang et al., 2024), while later approaches introduced explicit graph-based, hierarchical, or modular memories to improve organization and retrieval (Xu et al., 2025; Anokhin et al., 2024; Packer et al., 2023; Hu and Ying, 2025); Park et al. (2023) further showed that pairing episodic storage with reflective summarization produces emergent long-term planning in open-world simulations. Beyond basic RAG (Lewis et al., 2020), retrieval has been enhanced through indexing, temporal cues, semantic tagging, and adaptive chunking (Huang et al., 2025b; Gutiérrez et al., 2024; Liu et al., 2025a; Salama et al., 2025; Hu et al., 2025a). Related work also draws inspiration from neuroscience (Squire et al., 2015) and extends memory to multi-agent or collaborative settings (Wang et al., 2025c; Zhu et al., 2024; Qiao et al., 2024; Xu et al., 2024; Shuster et al., 2021). Procedural abstractions such as reusable agent skills offer an alternative by emphasizing modular capability reuse rather than ex-

perience accumulation (Anthropic, 2025). However, most memory systems are designed for a single agent or fixed framework, implicitly assuming homogeneous action spaces, tool interfaces, and execution environments. As a result, memory remains siloed across systems and does not support experience reuse across heterogeneous agent frameworks (Wang et al., 2025b). AGENT KB differs by providing a *shared, framework-neutral* procedural store; its key contribution is the *abstraction layer* that renders heterogeneous trajectories mutually interpretable, validated through three independently ablated design choices: experience unit abstraction, two-stage reuse, and disagreement gating (Table 3).

2.2 Agentic Knowledge Transfer

A growing body of work studies how agent execution traces can be summarized into reusable priors for future problem solving. Foundational agentic primitives including interleaved reasoning and acting (Yao et al., 2023), verbal self-reflection (Shinn et al., 2023), and autonomous tool use (Schick et al., 2023) establish the execution substrate on which knowledge transfer methods build. Early retrieval-based approaches condition agents on example trajectories or sub-workflows to stabilize tool use within a single agent or task family (Zheng et al., 2024; Zhou et al., 2024; Wang et al., 2025c); Patil et al. (2024) extends this to API selection, using retrieval-aware training so an agent can ground tool calls against a dynamically updated API index. Domain-specific pipelines (Bran et al., 2024; Tang et al., 2025) similarly compile knowledge from chemistry tools into LLM agents, achieving expert-level performance on narrow chemical reasoning tasks. Related templating and refinement methods further structure planning but typically operate within narrow domains or predefined schemas (Tan et al., 2025; Liu et al., 2025d; Khattab et al., 2024). More recent work introduces explicit knowledge repositories or collaborative planning mechanisms to facilitate multi-agent coordination (Zhu et al., 2024; Qiao et al., 2024; Liu et al., 2025b). Even in these settings, experience is often stored in isolated, framework-specific memories, with coordination relying on in-context learning or retrieval augmentation rather than shared experience representations (Lu et al., 2023; Zhong et al., 2024). Empirical studies further show that effective transfer depends strongly on experience quality (Alakuijala et al., 2025). Perspectives on lifelong and case-based cognition argue for AI-native infrastructures encoding solutions, workflows, and relational structure (Wang et al., 2025b; Pink et al., 2025). The **`Retrieve-Reuse-Revise-Retain`** cycle, a design principle rooted in case-based reasoning (Kolodner, 1992), is widely employed, yet existing systems rarely support experience reuse across heterogeneous agent frameworks. Related data-centric approaches, such as Learn-by-Interact (Su et al., 2025) and Early Experience (Zhang et al., 2025b), focus on improving a single agent within fixed environments and action spaces; on SWE-bench, Learn-by-Interact targets a single framework with a fixed action ontology, whereas AGENT KB integrates four frameworks with heterogeneous tool interfaces. These methods do not address cross-framework representation heterogeneity. In contrast, AGENT KB targets experience abstraction across heterogeneous agent frameworks, explicitly addressing tool incompatibility and knowledge interference through a standardized action vocabulary, staged reuse, and hybrid retrieval—distinct from standard RAG in that raw trajectories are never injected verbatim.

3 Methodology

We instantiate these principles in AGENT KB, a framework-agnostic experience memory for inference-time knowledge reuse. AGENT KB captures execution traces, abstracts them into reusable experience units, and exposes them through lightweight interfaces. At inference time, agents retrieve relevant experiences to support both planning and execution refinement. Concretely, experience reuse is decomposed into two stages: planning-level workflow guidance conditioned on task descriptions, and feedback-driven refinement conditioned on execution outcomes. Both stages follow the same **`Reason-Retrieve-Refine`** structure, while operating at different levels of granularity.

3.1 Accumulating AGENT KB

Experience Representation. We transform agent execution logs into structured experiences through human-guided abstraction:

$$E = \langle \pi, \gamma, S, \mathcal{C} \rangle, \quad \pi \in \mathbb{R}^d, \quad S = \{(a_i, r_i)\}_{i=1}^{\ell} \tag{1}$$

where $\pi = \phi_{\text{enc}}(q) \in \mathbb{R}^d$ (d=384) is the task embedding produced by a fixed encoder ϕ_{enc} (`all-MiniLM-L6-v2`); γ encodes goal constraints as structured predicates over the task state space;

Table 1: **Results on GAIA benchmark (val set).** All standard baselines are evaluated at `pass@1` (single attempt, no retrieval augmentation). For memory-augmented methods (A-MEM (Xu et al., 2025) and AGENT KB), we show the *same* base agent at `pass@1`/`pass@2`/`pass@3`, where each pass adds one retrieval-and-refine round on top of the previous. `pass@1` rows are thus directly comparable to standard baselines; `pass@2`/`pass@3` show additional gain from iterative refinement. AGENT KB at `pass@1` corresponds to a single structured retrieval-augmented call, distinct from Naive-RAG which injects raw trajectories without abstraction.

Method	Models	Config	Average	Level 1	Level 2	Level 3
Agentic Model						
Search-o1-32B (Li et al., 2025a)	`Qwen-3`	`pass@1`	39.8	53.8	34.6	16.7
WebThinker-32B-RL (Li et al., 2025b)	`Qwen-3`	`pass@1`	48.5	56.4	50.0	16.7
Closed-source Frameworks						
TraseAgent (Trase, 2024)	`Claude-3.5`	`pass@1`	70.3	83.0	69.8	46.2
Deep Research (OpenAI, 2024)	Unknown	`pass@1`	67.4	74.3	69.1	47.6
Desearch (AI, 2024)	`GPT-4o`	`pass@1`	57.0	71.7	58.1	23.1
Alita (Qiu et al., 2025)	`Claude-3.7`	`pass@1`	72.7	81.1	75.6	46.2
Open-source Frameworks						
OWL (Hu et al., 2025b)	`o3-mini`	`pass@1`	60.6	81.1	58.1	26.9
TapeAgents (Bahdanau et al., 2024)	`Claude-3.7`	`pass@1`	55.8	71.7	53.5	30.8
Magnetic-1 (Fourney et al., 2024)	`o1`	`pass@1`	46.1	56.6	46.5	23.1
FRIDAY (Wu et al., 2024b)	`GPT-4 turbo`	`pass@1`	34.6	45.3	34.9	11.5
smolagents (Roucher et al., 2025)	`GPT-4o`	`pass@1`	43.6 → 57.0 ↑13.4	52.8 → 71.7 ↑18.9	41.9 → 57.0 ↑15.1	30.8 → 26.9 ↓3.9
⇝ +A-MEM (Xu et al., 2025)		`pass@2`	53.9 → 64.2 ↑10.3	64.2 → 83.0 ↑18.9	53.5 → 64.0 ↑10.5	34.6 → 26.9 ↓7.7
		`pass@3`	57.0 → 69.1 ↑12.1	69.8 → 86.8 ↑17.0	55.8 → 69.8 ↑14.0	34.6 → 30.8 ↓3.8
smolagents (Roucher et al., 2025)	`GPT-4.1`	`pass@1`	55.2 → 57.1 ↑+1.9	67.9 → 70.5 ↑+2.6	53.5 → 56.3 ↑+2.8	34.6 → 33.3 ↓1.3
⇝ +NAIVE-RAG		`pass@2`	57.1 → 57.3 ↑+0.2	70.5 → 70.5 ↑+0.0	56.3 → 57.1 ↑+0.8	33.3 → 32.0 ↓1.3
		`pass@3`	57.3 → 57.5 ↑+0.2	70.5 → 71.2 ↑+0.7	57.1 → 57.6 ↑+0.5	32.0 → 30.7 ↓1.3
smolagents (Roucher et al., 2025)	`GPT-4.1`	`pass@1`	55.2 → 61.2 ↑6.1	67.9 → 79.3 ↑11.3	53.5 → 58.1 ↑4.7	34.6 → 34.6 ↑0.0
⇝ +AGENT KB		`pass@2`	61.8 → 67.3 ↑5.5	73.6 → 83.0 ↑9.4	62.8 → 67.4 ↑4.7	34.6 → 34.6 ↑0.0
		`pass@3`	68.5 → 73.9 ↑5.5	77.4 → 84.9 ↑7.6	68.6 → 73.3 ↑4.7	50.0 → 53.9 ↑3.9
smolagents (Roucher et al., 2025)	`Claude-3.7`	`pass@1`	58.8 → 65.5 ↑6.7	64.2 → 75.5 ↑11.3	61.6 → 66.3 ↑4.7	38.5 → 38.5 ↑0.0
⇝ +AGENT KB		`pass@2`	63.6 → 69.7 ↑6.1	77.4 → 79.3 ↑1.9	61.6 → 69.8 ↑8.1	42.3 → 50.0 ↑7.7
		`pass@3`	72.7 → 75.2 ↑2.4	81.1 → 84.9 ↑3.8	74.4 → 74.4 ↑0.0	50.0 → 57.7 ↑7.7
OWL (Hu et al., 2025b)	`GPT-4o`	`pass@1`	43.6 → 52.7 ↑9.1	52.8 → 64.2 ↑11.3	41.9 → 54.7 ↑12.8	30.8 → 23.1 ↓7.7
⇝ +AGENT KB		`pass@2`	53.9 → 60.6 ↑6.7	64.2 → 75.5 ↑11.3	53.5 → 61.6 ↑8.1	34.6 → 26.9 ↓7.7
		`pass@3`	57.0 → 63.6 ↑6.7	69.8 → 79.3 ↑9.4	55.8 → 61.6 ↑5.8	34.6 → 38.5 ↑3.8

each pair $(a_i, r_i) \in \mathcal{A} \times \mathcal{R}$ consists of an abstract action $a_i \in \mathcal{A}$ and a natural-language rationale r_i; and $\mathcal{C}$ carries compatibility metadata for cross-framework transfer. Abstraction is performed using few-shot prompting (10-15 human-curated exemplars per domain) together with a standardized action vocabulary shared across frameworks (e.g., smolagents (Zhu et al., 2025), OWL (Hu et al., 2025b), SWE-Agent (Yang et al., 2024), OpenHands (Wang et al., 2025a)). This vocabulary operates at the level of *abstract intent* (e.g., search, filter, execute, verify), rather than framework-specific tool APIs. Frameworks with novel action spaces require a one-time, human-written mapping from native actions to this shared vocabulary (typically 1–2 hours per framework); once written, the mapping is reused across all tasks and domains without touching agent or task logic, and the vocabulary itself is not assumed to be pre-shared across frameworks. Full construction details appear in Appendix A.2, with prompt templates in Appendix D.

Adaptive Memory Management. The knowledge base grows through addition, deduplication, and utility-based eviction (Fig. 2b). As experiences are retrieved and applied, each entry accumulates outcome feedback. This mechanism governs *memory curation*—which procedural records to retain—and does not modify the base agent's parameters or reasoning ability.

To manage growth, each experience E_j is assigned a scalar utility $u_j^{(0)} = 0.5 \in [0, 1]$. Let $\mathcal{R}_k(q) = \text{argtop-}k_j\, \sigma_j^{\text{hyb}}$ denote the retrieved set for query q. After task completion, define the binary reward

$$r_j = \mathbf{1}[E_j \in \mathcal{R}_k(q) \text{ and task succeeds}].$$

When top-k experiences are retrieved jointly, the task outcome is attributed to all k retrieved entries; this shared-credit heuristic avoids the combinatorial cost of counterfactual attribution. The utility updates via an exponential moving average (EMA),

$$u_j \leftarrow (1 - \eta)\, u_j + \eta\, r_j, \quad \eta \in (0, 1),$$

where η is the EMA learning rate. When capacity is reached, entries are ranked by u_j and the lowest-scoring experiences are evicted.

3.2 Execution via `Reason`-`Retrieve`-`Refine`

As shown in Fig. 2c, AGENT KB intervenes only at the *planning* and *feedback* stages, leaving the base

Table 2: **Results on multiple benchmarks.** Baseline rows show `pass@1` performance with no retrieval augmentation. AGENT KB rows show the *same* base agent with k retrieval-and-refine passes; `pass@1` is directly comparable to the baseline (single attempt, structured retrieval), while `pass@2`/`pass@3` provide iterative improvement. All comparisons within a block use the same model backbone and task budget.

(a) SWE-bench Lite (Jimenez et al., 2024) (300 instances) with max iterations of 50 and 100.

Method	Models	Success Rate (%) Max Iter 50	Max Iter 100	Budget Cap
SWE-agent (Yang et al., 2024)	GPT-4.1	24.3	27.0	$3.0
pass@1 +AGENT KB		31.7 ↑7.4	35.3 ↑8.3	$3.0
pass@2 +AGENT KB		36.7 ↑12.4	38.0 ↑11.0	$3.0
pass@3 +AGENT KB		**38.0** ↑13.7	**42.3** ↑15.3	$3.0
OpenHands (Wang et al., 2025a)	GPT-4.1	24.3	28.7	$4.5
pass@1 +AGENT KB		28.3 ↑4.0	31.7 ↑3.0	$4.5
pass@2 +AGENT KB		37.3 ↑13.0	42.3 ↑13.7	$4.5
pass@3 +AGENT KB		**38.7** ↑14.3	**45.7** ↑17.0	$4.5
OpenHands	Claude-3.7	30.0	41.3	$4.5
pass@1 +AGENT KB		46.7 ↑16.7	48.3 ↑7.0	$4.5
pass@2 +AGENT KB		49.7 ↑19.7	51.7 ↑10.3	$4.5
pass@3 +AGENT KB		**51.0** ↑21.0	**53.3** ↑12.0	$4.5

(b) GAIA (Mialon et al., 2024) (165 instances) with `pass@1` baseline and AGENT KB variants.

Method	Models	Accuracy (%) Avg	L1	L2	L3
OWL (Hu et al., 2025b)	GPT-4o	43.6	52.8	41.9	30.8
pass@1 +AGENT KB		52.7 ↑9.1	64.2 ↑11.3	54.7 ↑12.8	23.1 ↓7.7
pass@2 +AGENT KB		60.6 ↑6.7	75.5 ↑11.3	61.6 ↑8.1	26.9 ↓7.7
pass@3 +AGENT KB		**63.6** ↑6.7	**79.3** ↑9.4	**61.6** ↑5.8	**38.5** ↑3.8
smolagents (Roucher et al., 2025)	GPT-4.1	55.2	67.9	53.5	34.6
pass@1 +AGENT KB		61.2 ↑6.1	79.3 ↑11.3	58.1 ↑4.7	34.6 ↑0.0
pass@2 +AGENT KB		67.3 ↑5.5	83.0 ↑9.4	67.4 ↑4.7	34.6 ↑0.0
pass@3 +AGENT KB		**73.9** ↑5.5	**84.9** ↑7.6	**73.3** ↑4.7	**53.9** ↑3.9
smolagents	Claude-3.7	58.8	64.2	61.6	38.5
pass@1 +AGENT KB		65.5 ↑6.7	75.5 ↑11.3	66.3 ↑4.7	38.5 ↑0.0
pass@2 +AGENT KB		69.7 ↑6.1	79.3 ↑1.9	69.8 ↑8.1	50.0 ↑7.7
pass@3 +AGENT KB		**75.2** ↑2.4	**84.9** ↑3.8	**74.4** ↑0.0	**57.7** ↑7.7

(c) HLE (Bio/Chem) (Skarlinski et al., 2025) (149 instances) with `pass@1` baseline and AGENT KB variants.

Method	Models	Accuracy (%)
AutoGen (Wu et al., 2024a)	GPT-4.1	7.4
SciMaster (Chai et al., 2025)	GPT-4.1	9.5
Biomni (Huang et al., 2025a)	GPT-4.1	10.7
OpenHands (Wang et al., 2025a)	GPT-4.1	9.5
pass@1 +AGENT KB		10.1 ↑0.7
pass@2 +AGENT KB		12.1 ↑2.7
pass@3 +AGENT KB		14.1 ↑4.7

(d) GPQA benchmark (Rein et al., 2024) (198 instances) with `pass@1` baseline and AGENT KB variants.

Method	Models	Accuracy (%)
Direct Reasoning	o3-mini	75.0
	Claude-3.7	67.4
	GPT-4.1	64.6
OpenHands (Wang et al., 2025a)	GPT-4.1	62.6
pass@1 +AGENT KB		67.2 ↑4.6
pass@2 +AGENT KB		70.7 ↑8.1
pass@3 +AGENT KB		**72.7** ↑10.1

agent and its execution logic unchanged. Retrieved experiences are incorporated through a lightweight **Reason**-**Retrieve**-**Refine** cycle that adapts prior trajectories rather than replaying them verbatim.

Retrieval Pipeline. Given query q and knowledge base $\mathcal{B} = \{E_j\}_{j=1}^{N}$, retrieval uses two complementary scores. The semantic score is

$$\sigma_i^{\text{sem}} = \cos(\pi_q, \pi_i) := \frac{\pi_q^\top \pi_i}{\|\pi_q\| \, \|\pi_i\|}, \quad \pi_q = \phi_{\text{enc}}(q),$$

and the lexical score σ_i^{text} is given by BM25. Both are min-max normalized, $\tilde{\sigma}_i = (\sigma_i - \min_j \sigma_j)/(\max_j \sigma_j - \min_j \sigma_j)$, and fused as

$$\sigma_i^{\text{hyb}} = \alpha \, \tilde{\sigma}_i^{\text{text}} + (1 - \alpha) \, \tilde{\sigma}_i^{\text{sem}}, \quad \alpha \in [0, 1].$$

The top-k candidates $\mathcal{R}_k(q)$ are deduplicated before refinement (default α=0.5, k=3).

Planning Stage. The first **Reason**-**Retrieve**-**Refine** cycle operates on the incoming task description to produce a preliminary execution plan. In the **Reason** step, the system extracts key requirements and anticipated challenges, forming structured queries that emphasize reusable subroutines or completion patterns rather than surface-level instructions. The **Retrieve** step selects relevant past experiences via the hybrid scorer. In the **Refine** step, retrieved candidates are adapted to the current agent by aligning abstract action patterns with available tools and APIs, using metadata $\mathcal{C}$ for cross-framework compatibility. When multiple experiences are retrieved, the system synthesizes them into a single plan by resolving entity correspondences, substituting tools, and reordering steps to satisfy task constraints. The resulting executable plan ρ is returned to the agent framework for execution in its native environment.

Feedback Stage. After initial execution, AGENT KB proceeds with a second **Reason**-**Retrieve**-**Refine** cycle based on execution traces. In the **Reason** phase, the system examines these traces to detect errors, bottlenecks, or unexpected results. During the **Retrieve** phase, it generates queries from traces, focusing on successful refinement experiences under similar failure modes rather than similar tasks. In the **Refine** phase, candidate fixes are prepared for live execution, taking into account observed errors and domain constraints. To prevent harmful interference, a disagreement gate is applied:

$$\mathcal{G}(\rho, \rho') = \mathbb{1}\left[\frac{\phi_{\text{enc}}(\rho)^\top \phi_{\text{enc}}(\rho')}{\|\phi_{\text{enc}}(\rho)\| \, \|\phi_{\text{enc}}(\rho')\|} \geq \beta \right],$$

where ϕ_{enc} is the shared encoder from Eq. (1), $\rho \in \mathcal{R}^*$ is the original plan, $\rho' \in \mathcal{R}^*$ the proposed refinement, and $\beta = 0.8$ by default. Only

refinements with $\mathcal{G}(\rho, \rho') = 1$ are applied to the agent.

4 Experiment

4.1 Setup

Dataset. We benchmark AGENT KB on four suites covering reasoning and software engineering. GAIA (Mialon et al., 2024) contributes 165 tasks: 53 Level 1 (factual lookup), 86 Level 2 (multi-step reasoning), and 26 Level 3 (analysis and synthesis). The cleaned biology & chemistry subset provides 149 tasks needing multimodal reasoning with scientific images and tools. GPQA (Rein et al., 2024) offers 198 graduate-level MCQs in physics, chemistry, and biology, requiring experts. Both GAIA and HLE support web browsing, file I/O, and tools within standard limits, whereas GPQA focuses on reasoning without the use of external tools. Benchmarks are reported with `pass@1`, `pass@2`, and `pass@3` accuracy. SWE-bench Lite (Jimenez et al., 2024) comprises 300 GitHub issues across Python repositories; success is tested with 50- and 100-iteration limits for reproducibility. It restricts network access and limits operations to those within the repository. These benchmarks show how knowledge refinement affects reasoning (GAIA, HLE, GPQA) and software engineering (SWE-bench).

AGENT KB Construction. We bootstrap AGENT KB with 80 seed trajectories written by five computer-science graduate students (60 BrowseComp/HopRAG-style browsing traces and 20 SWE-Gym-style coding logs). These curated demonstrations are never retrieved directly; instead, they guide automatic rollouts executed by smolagents (Zhu et al., 2025), OWL (Hu et al., 2025b), SWE-Agent (Yang et al., 2024), and OpenHands (Wang et al., 2025a) across BrowseComp (Wei et al., 2025), HopRAG (Liu et al., 2025c), HLE[2] (Phan et al., 2026), WebWalkerQA (Wu et al., 2025), RepoClassBench (Deshpande et al., 2024), SWE-Gym-Raw (Pan et al., 2025), and RepoEval (Zhang et al., 2023). We normalize successful and unsuccessful runs into experience units, yielding approximately 9k workflow summaries and 7k execution snippets prior to evaluation. All source datasets are held-out from evaluation: no test-set trajectories or labels appear in AGENT KB, analogous to pre-indexing a RAG corpus. To isolate structured abstraction from raw retrieval, we include a **Naive-RAG** baseline that injects the same top-k raw trajectories verbatim, without abstraction, staged planning, or disagreement gating (Appendix B.1).

Model Configurations. We attach the same AGENT KB to all planners via lightweight RPC so experiences in one framework are *instantly* available to others, showing true cross-framework knowledge transfer. GAIA experiments use smolagents (Roucher et al., 2025) (backed by `GPT-4o`, `GPT-4.1`, `Claude-3.7`, `Qwen-3 32B`, and `DeepSeek-R1`) and OWL (`GPT-4o`); SWE-bench Lite uses SWE-Agent (`GPT-4.1`, `o3-mini`) and OpenHands (`GPT-4o`, `GPT-4.1`, `Claude-3.7`, `Qwen-3 32B`, `DeepSeek-R1`, `o3-mini`). Each benchmark has three passes: `pass@1` retrieves experiences without labels, `pass@2` adds failure diagnoses, and `pass@3` reattempts unresolved cases with an expanded retrieval pool. Unless stated, we fix the model to `GPT-4.1`, temperature to 1.0, and top-k to 3, as in Figure 3. Cost estimates assume OpenAI pricing ($1.36/M prompt, $5.44/M tokens); see Appendix B.2 for details.

Baselines. On GAIA, we compare against agentic reasoning models Search-o1 (Li et al., 2025a) and WebThinker (Li et al., 2025b), closed-source systems Alita (Qiu et al., 2025), Deep Research (OpenAI, 2024), Desearch (AI, 2024), and TraseAgent (Trase, 2024), and open-source frameworks TapeAgents (Bahdanau et al., 2024), Magnetic-1 (Fourney et al., 2024), and FRIDAY (Wu et al., 2024b). On HLE, we additionally compare against AutoGen (Wu et al., 2024a), SciMaster (Chai et al., 2025), and Biomni (Huang et al., 2025a).

4.2 Main Results

Equal-cost comparison (`pass@1`). At `pass@1`, AGENT KB performs one structured retrieval-and-refine round—the same compute as unaugmented baselines. On GAIA, smolagents (`GPT-4.1`) reaches 61.2% (+6.1 pp) versus 57.1% for Naive-RAG at identical retrieval budget, showing gains come from structured abstraction rather than retrieval alone. Naive-RAG saturates quickly (57.1% → 57.5% across passes), while AGENT KB compounds to 73.9% at `pass@3`.

Table 1 shows consistent gains across *heterogeneous* agent stacks on GAIA. With smolagents, `GPT-4.1` improves from 55.2% to 73.9% `pass@3` (+18.7), with the largest gains at Level 2

[2]We deliberately removed the biology & chemistry subset from Humanity's Last Exam as a test set.

Table 3: Ablation study for AGENT KB's components.

Ablation Setting	Avg	Level 1	Level 2	Level 3
smolagent	55.15	67.92	53.49	34.62
smolagents +AGENT KB	61.21	79.25	58.14	34.62
w/o Planning Step	59.39	75.47	56.98	34.62
w/o Feedback Step	59.39	73.58	58.14	34.62
w/o **Reason** Module	60.00	77.36	56.98	34.62
w/o **Retrieve** Module	57.58	73.58	54.65	34.62
w/o **Refine** Module	55.15	69.81	53.49	30.77
w/ Raw Workflow	58.18	73.58	55.81	34.62

(53.5% → 73.3%, +19.8). `Claude-3.7` reaches 75.2% `pass@3` with a substantial lift on Level 3 (38.5% → 57.7%), matching or exceeding closed-source systems such as h2oGPTe (63.6%). To compare against other memory systems under identical conditions, we run A-Mem (Xu et al., 2025) on the same smolagents+`GPT-4o` stack: A-Mem reaches 69.1% (`pass@3`), while AGENT KB reaches 73.9%, showing further gains from hybrid retrieval and staged reuse over a strong episodic-memory baseline. OWL (`GPT-4o`) improves by 20.0 pp overall (43.6% → 63.6%) with consistent gains on harder instances (30.8% → 38.5%).

On SWE-bench Lite, a substantially different execution-centric benchmark, Table 2b shows comparable improvements. When paired with SWE-Agent, `GPT-4.1` improves from 24.3% to 38.0% under 50 iterations and to 42.3% under 100 iterations. OpenHands achieves a 14.4 gain at 50 iterations (24.3% → 38.7%) and a 17.0 gain at 100 iterations (28.7% → 45.7%). The strongest backbone, `Claude-3.7`, shows the largest absolute improvements, reaching 51.0% at 50 iterations and 53.3% at 100 iterations.

Beyond GAIA and SWE-bench, AGENT KB also improves performance on challenging scientific QA benchmarks. On HLE (Table 2c), OpenHands starts below Biomni (Huang et al., 2025a) (9.5% vs. 10.7%) but surpasses it once retrieval is applied, reaching 12.1% at `pass@2` and 14.1% at `pass@3`. On GPQA (Table 2d), OpenHands with `GPT-4.1` improves from 62.6% to 72.7%, approaching recent proprietary systems. All gains are obtained without additional fine-tuning or tool customization, underscoring the transferability of the shared experience store across agent architectures ("zero-shot" here means no task labels from the evaluation sets enter the KB; construction uses auxiliary splits of HLE and related datasets, not the Bio/Chem test subset).

Figure 3 further shows that performance consistently improves with successive retrieval passes across both reasoning benchmarks (GAIA, HLE, GPQA) and software engineering tasks (SWE-bench). Across models, `pass@1` provides an initial boost by incorporating compatible workflows, while later passes (`pass@2`/`pass@3`) contribute targeted refinements. These effects are most pronounced on scientific and general reasoning tasks, where iterative reuse of experience yields compounding gains. Representative GAIA and SWE-bench walkthroughs are provided in Appendix C, with full execution traces in Appendix C.1.

4.3 Ablation Studies

Several components of AGENT KB involve explicit design choices (e.g., multi-stage retrieval, scoring, confidence gating, and memory scale), which may appear heuristic in isolation but are motivated by distinct failure modes in cross-framework transfer.

Table 4: Performance of smolagents (`GPT-4.1`) on GAIA and SWE-bench with different knowledge types. Baseline uses no augmentation, HAND CRAFTED uses student-annotated experiences, and AGENT KB uses automatically extracted and refined experiences.

Knowledge type	GAIA Average	GAIA Level 1	GAIA Level 2	GAIA Level 3	SWE-bench Lite
Baseline	55.15	67.92	53.49	34.62	24.33
+ HAND CRAFTED	**76.97**	**84.91**	**79.07**	53.85	**55.67**
+ AGENT KB	75.15	**84.91**	74.42	**57.69**	51.00

Impact of multi-stage retrieval and refinement. To assess the contribution of individual stages in the reasoning and retrieval pipeline, we conduct systematic ablation studies in Table 3. Details of the ablation elements can be found in Appendix A.4. Removing the **Refine** stage incurs the largest drop (−6.06), confirming that retrieved workflows must be adapted rather than replayed. Removing either **Retrieve** pass caps average `pass@1` at 59.39%, with sharper Level 1 erosion without the feedback stage (79.25%→73.58%) and without the planning stage (79.25%→75.47%), underscoring their complementary planning/feedback roles. The **Refine** stage imposes the largest penalty when ablated (61.21%→55.15% overall; Level 3 34.62%→30.77%), while dropping **Retrieve** loses 3.63 and **Reason** only 1.21, indicating that knowledge grounding and structured hypothesis drafting together prevent regression even when raw workflow logs (58.18%) are available. Figure 4a further shows that incorporating a disagreement gate stabilizes performance, with optimal results at $\beta \approx 0.8$.

Impact of retrieval strategy. We compare three retrieval methods: lexical (BM25), semantic (embedding), and hybrid. The hybrid consistently achieves

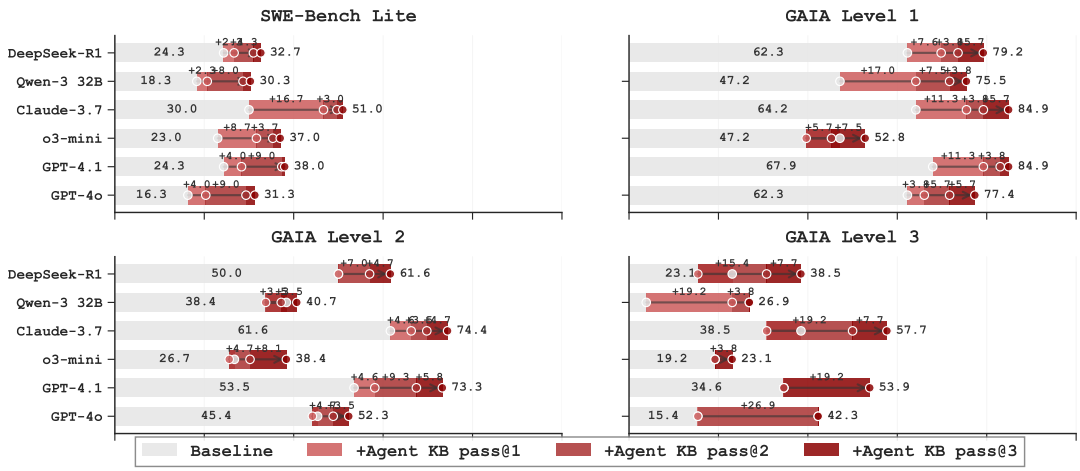

Figure 3: Score improvements (%) across benchmarks for multiple LLMs enhanced with AGENT KB.

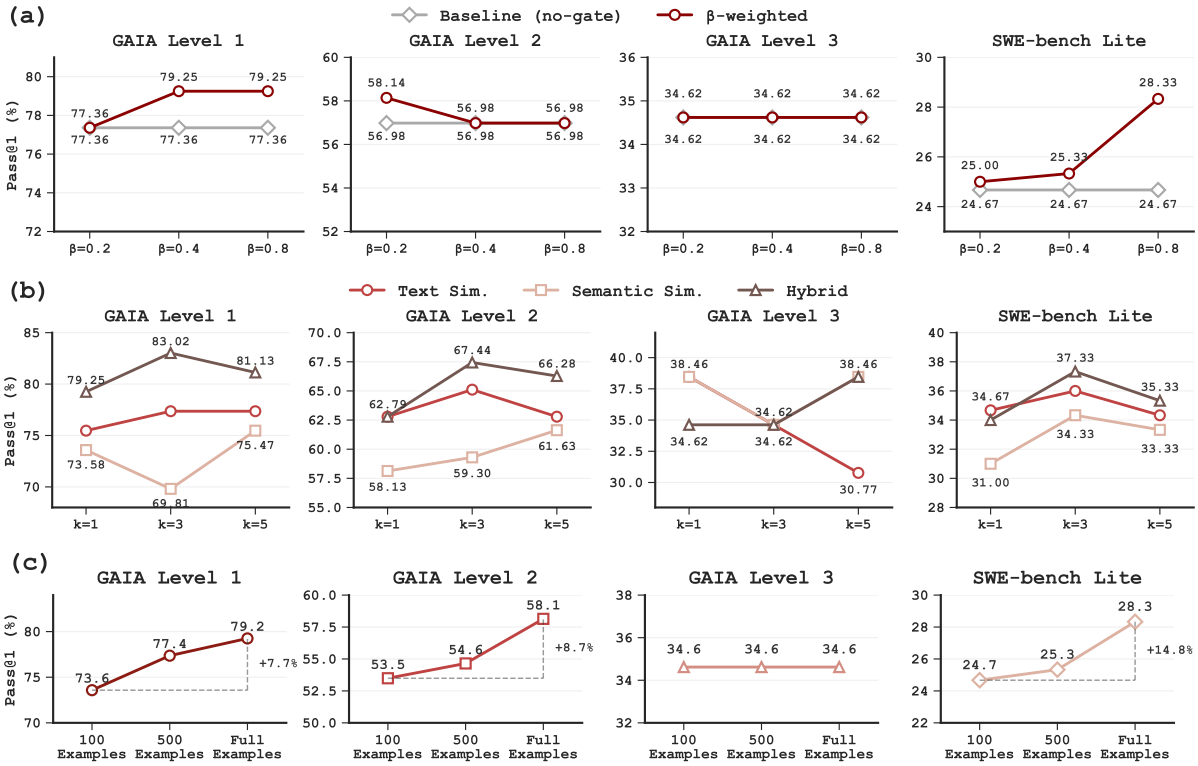

Figure 4: Ablation analysis of retrieval configuration, knowledge-base size, and feedback weighting. (a) Impact of confidence weighting hyper-parameter β on feedback integration. (b) Comparison of retrieval strategies across top-k settings. Text similarity, semantic similarity, and hybrid methods are evaluated on GAIA Levels 1-3 and SWE-bench Lite. (c) Effect of knowledge-base size on validation performance.

the highest accuracy in reasoning and software engineering benchmarks, combining precise matches with semantic coverage. This fusion is vital for cross-framework knowledge transfer, as agent architectures require either exact tool matches or conceptual similarity, depending on the task. Figure 4b shows that hybrid retrieval maintains strong performance across top-k settings, with peak accuracy at $k = 3$ (83.0% on GAIA Level 1), effective across benchmarks.

Impact of knowledge base size. Figure 4c shows performance degrades gracefully as AGENT KB shrinks. With 100 examples, general reasoning and software benchmarks retain capability, indicating that small stores offer useful prior trajectories. Expanding to 500 examples yields consistent gains in reasoning, while software tasks benefit greatly, highlighting the importance of this scale for code repair. Advanced reasoning tasks remain flat, suggesting that the quality of abstraction is

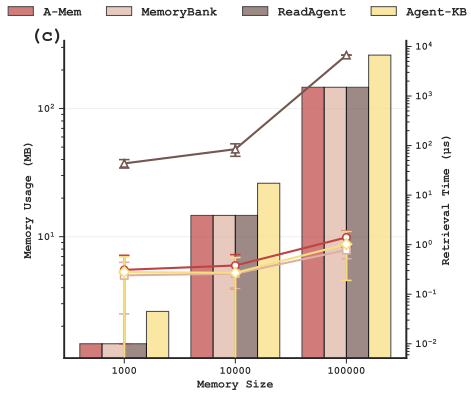

Figure 5: Retrieval latency & memory footprint when scaling different stores. Baselines are taken from Xu et al. (2025).

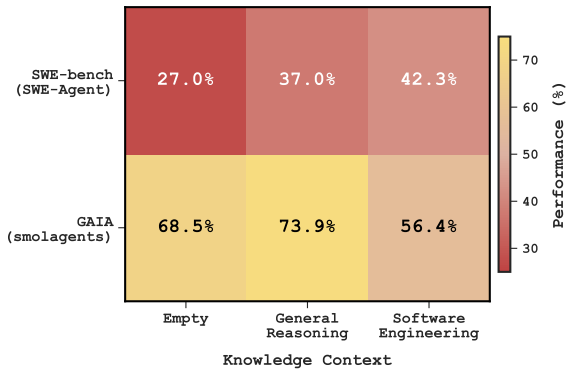

Figure 6: Cross-domain knowledge transfer analysis. Performance comparison when applying domain-specific knowledge bases to different task types.

bottlenecked, not the quantity. Larger knowledge bases reliably improve performance, but supporting complex tasks needs better structuring and retrieval.

Latency and memory overhead. Figure 5 compares raw lookup latency against store size for AGENT KB and alternatives such as A-Mem (Xu et al., 2025), MemoryBank (Zhong et al., 2024), and ReadAgent (Lee et al., 2024). AGENT KB maintains competitive latency while occupying a similar memory footprint across different store sizes. Alongside the cost accounting in Appendix B.2, this indicates that AGENT KB's construction and inference overhead remains minor relative to the performance gains.

4.4 Further Analysis

Beyond validating components, we analyze experiences learned by AGENT KB. Specifically, we study (i) if agent-crafted experiences match the quality of curated ones, and (ii) how knowledge transfers across domains. These analyses reveal what experiences are reusable and where transfer fails.

Agent-crafted versus hand-crafted experiences. Table 4 shows that AGENT KB's automatically refined knowledge matches hand-crafted experiences (annotated by five computer science students) on GAIA (75.15% vs. 76.97%), surpasses them on Level 3 (57.69% vs. 53.85%). These results suggest that agent-crafted experiences can capture transferable procedural knowledge of comparable quality to manual curation, while scaling more effectively to complex tasks.

Transferability of knowledge bases. Figure 6 compares *general reasoning experiences* (BrowseComp, HopRAG, HLE, WebWalkerQA) with *software engineering experiences* (RepoClassBench, SWE-Gym-Raw, RepoEval). Reasoning experience reaches 73.9% on GAIA and 37.0% on SWE-bench; SWE experience reaches 42.3% on SWE-bench but drops to 56.4% on GAIA. Transfer is therefore asymmetric: SWE knowledge does not generalize to open-ended reasoning, while reasoning experience retains partial utility in SWE domains. Applying SWE-only experiences to GAIA underperforms the combined store (56.4% vs. 73.9%), and some harder-level configurations regress relative to baseline—a real limitation that the disagreement gate partially mitigates but does not eliminate. The combined AGENT KB outperforms either domain-specific store because experience diversity helps reasoning tasks while the gate bounds negative transfer from SWE experiences.

4.5 Error Analysis

On the GAIA benchmark, we compare error distributions in baseline and AGENT KB-augmented settings. For `GPT-4.1` (Figure 7a), 49 errors are shared across both runs, while AGENT KB corrects 25 baseline-only errors and introduces 15 new ones, resulting in a net reduction of 10. For `Claude-3.7` (Figure 7b), 22 baseline errors are fixed with 11 new errors, leading to a net improvement of 11. We classify errors into six categories: retrieval, planning, reasoning, format, perception, and execution, with pie charts illustrating their proportions. With `GPT-4.1`, retrieval errors drop from 24 to 20, and planning errors decrease from 13 to 10, indicating improvements in query formulation and workflow reuse. `Claude-3.7` shows larger reductions in reasoning errors (13 to 8) and fewer retrieval failures (19 to 16). These gains reflect the effect of our knowledge base, which encodes reusable

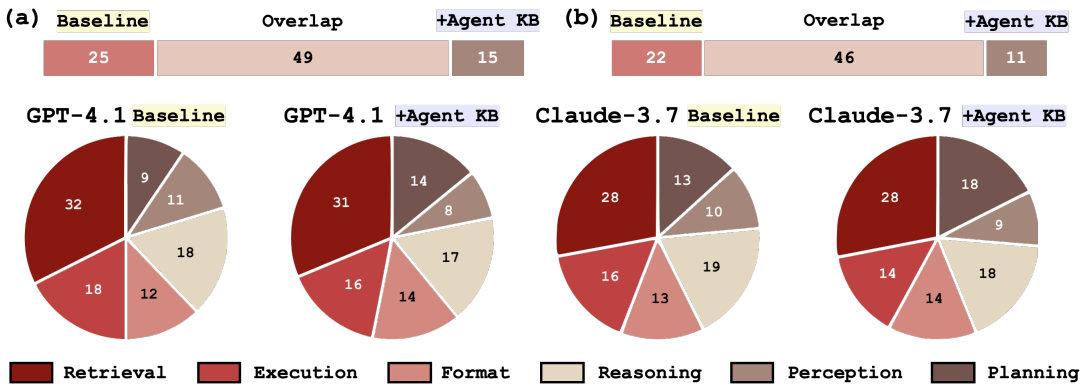

Figure 7: Error analysis on GAIA comparing smolagents with and without AGENT KB. Venn diagrams summarize shared and setting-specific failures, while bar charts break down errors by category. Results are shown for `GPT-4.1` and `Claude-3.7`.

search protocols, planning templates, and formatting standards. Tool capabilities primarily limit perception errors, but their impact is reduced by avoiding unnecessary steps. Overall, both models achieve similar error reductions: `Claude-3.7` enhances reasoning robustness, while `GPT-4.1` shows better perception alignment on GAIA.

5 Conclusion

We presented AGENT KB, a cross-framework memory layer that abstracts heterogeneous agent traces into reusable experience units. By coupling *hybrid retrieval* with a *disagreement-gated, two-stage refinement pipeline*, it addresses the core challenges of representation heterogeneity, context mismatch, and knowledge interference. Controlled comparisons against Naive-RAG confirm gains arise from structured abstraction rather than retrieval budget, and ablations validate each component individually. Automatically generated experiences match hand-curated quality, supporting scalable deployment. Cross-domain transfer remains asymmetric and onboarding new frameworks still requires a one-time action mapping; domain-aware routing and automatic onboarding are natural directions for future work.

Limitations

AGENT KB has four notable limitations. First, **onboarding** a new agent framework requires a one-time, human-written mapping from native actions to the shared intent vocabulary (typically 1–2 hours), which is not yet automatic and may be more involved for fine-grained or highly novel action spaces; knowledge-base construction also draws from publicly available interaction datasets (BrowseComp, SWE-Gym, etc.) whose licensing and data provenance should be verified before deployment. Second, **heuristic components**—the retrieval fusion weight α and disagreement gate threshold β—are chosen via held-out validation; high β may filter semantically diverse but beneficial refinements, while low β risks interference. Third, **cross-domain transfer is asymmetric**: SWE experiences do not generalize to open-ended reasoning, and some configurations regress on harder instances; the disagreement gate bounds but does not eliminate interference. Finally, AGENT KB improves **inference-time procedural memory** without modifying the base agent's parameters, so gains are bounded by what structured retrieval-augmented prompting can achieve.

Ethics Statement

Our work builds on publicly available datasets (GAIA, HLE, GPQA, SWE-bench) and follows their respective licenses. We do not foresee direct ethical concerns; however, when deploying agent memory systems in practice, one should carefully consider data privacy, potential bias in retrieved knowledge, and the risk of misuse in high-stakes domains.

Acknowledgments

AI assistants were used to support writing and editing, including improving clarity, organization, and LaTeX formatting. All technical content, experimental design, results, and interpretations were developed and verified by the authors.

References

Desearch AI. 2024. Desearch.

Minttu Alakuijala, Ya Gao, Georgy Ananov, Samuel Kaski, Pekka Marttinen, Alexander Ilin, and Harri Valpola. 2025. Memento no more: Coaching ai agents to master multiple tasks via hints internalization. *arXiv preprint arXiv:2502.01562*.

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

Anthropic. 2025. Claude code skills. `https://docs.anthropic.com/en/docs/claude-code/skills-tutorial`.

Dzmitry Bahdanau, Nicolas Gontier, Gabriel Huang, Ehsan Kamalloo, Rafael Pardinas, Alex Piché, Torsten Scholak, Oleh Shliazhko, Jordan Prince Tremblay, Karam Ghanem, et al. 2024. Tapeagents: a holistic framework for agent development and optimization. *arXiv preprint arXiv:2412.08445*.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2024. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):469–478.

Jingyi Chai, Shuo Tang, Rui Ye, Yuwen Du, Xinyu Zhu, Mengcheng Zhou, Yanfeng Wang, Yuzhi Zhang, Linfeng Zhang, Siheng Chen, et al. 2025. Scimaster: Towards general-purpose scientific ai agents, part i. x-master as foundation: Can we lead on humanity's last exam? *arXiv preprint arXiv:2507.05241*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. ChatEval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Ajinkya Deshpande, Anmol Agarwal, Shashank Shet, Arun Iyer, Aditya Kanade, Ramakrishna Bairi, and Suresh Parthasarathy. 2024. Class-level code generation from natural language using iterative, tool-enhanced reasoning over repository. *arXiv preprint arXiv:2405.01573*.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. 2024. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025a. HiAgent: Hierarchical working memory management for solving

long-horizon agent tasks with large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32779–32798.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Ping Luo, and Guohao Li. 2025b. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation.

Pengbo Hu and Xiang Ying. 2025. Unified mind model: Reimagining autonomous agents in the llm era. *arXiv preprint arXiv:2503.03459*.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. 2025a. Biomni: A general-purpose biomedical ai agent. *biorxiv*.

Tenghao Huang, Kinjal Basu, Ibrahim Abdelaziz, Pavan Kapanipathi, Jonathan May, and Muhao Chen. 2025b. R2D2: Remembering, reflecting and dynamic decision making for web agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*.

Omar Khattab, Arnav Singhvi, Keshav Santhanam, Zhiyuan Zhang, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations*.

Janet L. Kolodner. 1992. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. 2024. A human-inspired reading agent with gist memory of very long contexts. In *Proceedings of the 41st International Conference on Machine Learning*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proc. NeurIPS*.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025a. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025b. Webthinker: Empowering large reasoning models with deep research capability. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Baichuan Liu, Chen Li, Ming Tan, Wenqiang Liu, and Yiming Yang. 2025a. Echo: A large language model with temporal episodic memory. *arXiv preprint arXiv:2502.16090*.

Hanchao Liu, Rongjun Li, Weimin Xiong, Ziyu Zhou, and Wei Peng. 2025b. WorkTeam: Constructing workflows from natural language with multi-agents. In *Proceedings of the 2025 Annual Conference of the Nations of the Americas Chapter of the ACL: Industry Track*.

Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025c. Hoprag: Multi-hop reasoning for logic-aware retrieval-augmented generation. *arXiv preprint arXiv:2502.12442*.

Siwei Liu, Jinyuan Fang, Han Zhou, Yingxu Wang, and Zaiqiao Meng. 2025d. Sew: Self-evolving agentic workflows for automated code generation. *arXiv preprint arXiv:2505.18646*.

Junru Lu, Siyu An, Mingbao Lin, Gabriele Pergola, Yulan He, Di Yin, Xing Sun, and Yunsheng Wu. 2023. Memochat: Tuning LLMs to use memos for consistent long-range open-domain conversation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2024. deepresearch.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. 2025. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. Memgpt: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training software engineering agents and verifiers with SWE-gym. In *Proceedings of the 42nd International Conference on Machine Learning*.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th*

Annual ACM Symposium on User Interface Software and Technology.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. 2026. Humanity's last exam. *Nature*, 649:1139–1146.

Mathis Pink, Qinyuan Wu, Vy Ai Vo, Javier Turek, Jianing Mu, Alexander Huth, and Mariya Toneva. 2025. Position: Episodic memory is the missing piece for long-term llm agents. *arXiv preprint arXiv:2502.06975*.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024. Agent planning with world knowledge model. *Advances in Neural Information Processing Systems*, 37:114843–114871.

Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, et al. 2025. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv preprint arXiv:2505.20286*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 2025. 'smolagents': a smol library to build great agentic systems. `https://github.com/huggingface/smolagents`.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

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

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EACL 2021*.

Michael Skarlinski, Jon Laurent, Albert Bou, and Andrew White. 2025. About 30% of humanity's last exam chemistry/biology answers are likely wrong. `https://www.futurehouse.org/research-announcements/hle-exam`. Accessed: 2025-09-23.

Larry R Squire, Lisa Genzel, John T Wixted, and Richard G Morris. 2015. Memory consolidation. *Cold Spring Harbor perspectives in biology*, 7(8):a021766.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö Arık. 2025. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *The Thirteenth International Conference on Learning Representations*.

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

Xiaoyu Tan, Bin Li, Xihe Qiu, Chao Qu, Wei Chu, Yinghui Xu, and Yuan Qi. 2025. Meta-agent-workflow: Streamlining tool usage in llms through workflow construction, retrieval, and refinement. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 458–467.

Xiangru Tang, Tianyu Hu, Muyang Ye, Yanjun Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, Arman Cohan, and Mark Gerstein. 2025. Chemagent: Self-updating memories in large language models improves chemical reasoning. In *The Thirteenth International Conference on Learning Representations*.

Trase. 2024. Meet trase systems. [Online]. `https://www.trasesystems.com/`.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2025a. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. 2024. MemoryLLM: Towards self-updatable large language models. In *Proceedings of the 41st International Conference on Machine Learning*.

Yu Wang, Chi Han, Tongtong Wu, Xiaoxin He, Wangchunshu Zhou, Nafis Sadeq, Xiusi Chen, Zexue He, Wei Wang, Gholamreza Haffari, Heng Ji, and Julian McAuley. 2025b. Towards lifespan cognitive systems. *Transactions on Machine Learning Research*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025c. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, et al. 2025. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024a. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024b. Os-copilot: Towards generalist computer agents with self-improvement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*.

Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Kang Liu, and Jun Zhao. 2024. Generate-on-graph: Treat LLM as both agent and KG in incomplete knowledge graph question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

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

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484.

Jieyu Zhang, Ranjay Krishna, Ahmed Hassan Awadallah, and Chi Wang. 2025a. Ecoassistant: Using llm assistants more affordably and accurately. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, et al. 2025b. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

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

Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. 2024. Trad: Enhancing llm agents with step-wise thought retrieval and aligned decision. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–13.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. 2025. Symbolic learning enables self-evolving agents. In *The Thirteenth International Conference on Learning Representations*.

He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Yi Yao, Hanhao Li, Ningning Wang, Pai Liu, et al. 2025. Oagents: An empirical study of building effective agents. *arXiv preprint arXiv:2506.15741*.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. 2024. Knowagent: Knowledge-augmented planning for LLM-based agents. In *Proc. NAACL Findings*.

A Additional Method Details

A.1 Human-Crafted Seed Experiences

This section describes how we create the seed experiences used to initialize AGENT KB. These examples are crafted manually and used only during the offline construction phase—not at inference time. Annotators review failed execution logs to identify recurring failure patterns (*e.g.*incorrect tool selection, misaligned reasoning chains, or missing preconditions) and abstract them into correction templates. All hand-crafted experiences follow a unified schema:

AGENT KB Schema

```
{
  "question": "<task description>",
  "agent_plan": "<abstracted plan>",
  "agent_experience":       "<general
insight>"
}
```

The procedure for constructing seed experiences is as follows.

- **Step 1: Team Setup.** Five computer science graduate students familiar with the GAIA benchmark and agent-reasoning workflows collaborated to design high-quality prompts. The goal was to transform successful agent trajectories into structured, reusable instructions that capture key decision rules, tool usage, and constraints.
- **Step 2: Log Review.** Each student reviewed execution logs from GAIA tasks across all difficulty levels, focusing on successful reasoning traces and efficient tool-use patterns. Failed cases were revisited and corrected by applying insights drawn from successful trajectories.
- **Step 3: Prompt Authoring and Standardization.** Identified reasoning patterns were abstracted into reusable workflows and rewritten as natural-language instructional prompts using a consistent structure, imperative voice, and tool-neutral references. A representative hand-crafted experience is shown below:

  `Search for the 2015 paper "Pie Menus or Linear Menus, Which Is Better?" on a scholarly database (e.g., Google Scholar or IEEE Xplore) and note the authors in "First M. Last" format. For each author, look up their publication history on DBLP or Google Scholar and list all their papers with publication years. Determine which author has works published before 2015 and collect that author's prior publications. Sort the earlier papers by year, identify the very first one, and verify its title against the database entry to ensure accuracy.`

- **Step 4: Effectiveness Testing and Selection.** Each hand-crafted experience was evaluated via few-shot prompting on similar GAIA tasks. The top 80 prompts by empirical performance were retained as the seed experience set.
- **Step 5: Generalization to Other Benchmarks.** The 80 seed experiences then served as few-shot exemplars for generating abstracted experiences across other reasoning benchmarks.

A.2 Construction of AGENT KB

Data Collection. AGENT KB is built from execution traces collected across multiple agent frameworks and task domains. Each trace includes a task description, step-by-step reasoning, tool invocations, intermediate outcomes, and any errors encountered. Traces are then normalized by removing framework-specific artifacts, mapping tool calls to a shared action vocabulary, and condensing fine-grained reasoning steps into high-level decisions.

Experience Generation. The 80 seed experiences (Section A.1) serve as few-shot exemplars to guide an LLM-based generation model. For each collected trace, the model produces a structured experience entry

$$E_{\text{gen}} = \langle \pi, \gamma, S, \mathcal{C} \rangle, \quad \pi = \phi_{\text{enc}}(q) \in \mathbb{R}^d,$$

following the same schema as the seed entries (Eq. (1)).

Indexing and Retrieval. All validated experiences are embedded and indexed. Given a new query q_{new}, relevant entries are ranked by cosine similarity:

$$\text{sim}(q_{\text{new}}, e) = \frac{\phi_{\text{enc}}(q_{\text{new}})^\top \phi_{\text{enc}}(e)}{\|\phi_{\text{enc}}(q_{\text{new}})\| \, \|\phi_{\text{enc}}(e)\|},$$

and the top-k results are returned to downstream agents. The prompt templates used for experience generation are provided in Appendix D.

A.3 Implementation of `smolagents`

Framework Overview. Our implementation builds on the `smolagents` multi-agent system (Zhu et al., 2025), which employs a master–slave collaboration architecture. The *CodeAgent* serves as the primary agent responsible for overall planning and coordination, while the *SearchAgent* specializes in information retrieval. Both agents communicate through standardized interfaces, promoting modularity and extensibility.

Agent Tool Capabilities. Each agent is equipped with modality-specific tools. CodeAgent incorporates three inspection tools to support planning, multi-tool orchestration, and analysis:

- `TextInspectorTool`: reads non-image files (HTML, PDF, DOCX, XML/CSV), converts raw content to clean Markdown, and optionally answers natural-language queries with automatic length trimming.
- `AudioInspectorTool`: validates supported audio formats, transcribes audio via Whisper, and generates structured responses conditioned on user queries.
- `VisualInspectorTool`: processes image files or URLs and invokes multimodal models (Qwen-VL with GPT-4o fallback) to describe visual content or answer image-based queries.

SearchAgent focuses on external information retrieval and includes four tools:

- `SearchTool`: a unified interface over multiple web search engines supporting query rewriting, year filtering, and visit tracking.
- `WikiSearchTool`: queries the MediaWiki API and returns canonical titles, URLs, and summary extracts.
- `CrawlerReadTool`: asynchronously downloads full web pages and strips boilerplate for clean text extraction.
- `CrawlerArchiveSearchTool`: retrieves archived versions of web pages to support provenance checks and time-sensitive queries.

Task Distribution and Communication. Execution begins with CodeAgent, which analyzes the user task, decomposes it into subtasks, and delegates those requiring external retrieval to SearchAgent. SearchAgent performs targeted searches and returns structured results for integration into the execution flow. To improve efficiency, the framework supports task-priority management, concurrent execution of independent subtasks, optimized resource scheduling, and result caching to avoid redundant computation.

A.4 Ablation of `Reason`-`Retrieve`-`Refine`

To evaluate each component of AGENT KB, we conduct ablation studies by systematically removing or disabling specific modules. The full deployment loop operates in two retrieval phases:

- **Planning stage** forms the initial plan, following the `Reason`-`Retrieve`-`Refine` cycle to summarize the query, select relevant experiences, and synthesize them into an executable workflow.
- **Feedback stage** reuses the same cycle on execution traces, reasoning over critic highlights, retrieving relevant precedents, and refining the plan under the disagreement gate.

The ablation variants are defined as follows, with results summarized in Table 3:

- **w/o Planning Stage**: Removes the first-stage `Reason`-`Retrieve`-`Refine` cycle entirely.
- **w/o Feedback Stage**: Removes the second-stage `Reason`-`Retrieve`-`Refine` cycle entirely.
- **w/o `Reason` Module**: Skips the reasoning step in both stages; retrieval is applied directly to the raw query without prior summarization.
- **w/o `Retrieve` Module**: Skips retrieval in both stages; agents rely solely on prompt-based instructions without consulting prior experiences.
- **w/o `Refine` Module**: Skips refinement in both stages; retrieved content is injected verbatim without structured adaptation.
- **w/ Raw Workflow**: Applies the full retrieval pipeline but without modular control—the model follows a standard prompting strategy

throughout, bypassing the structured **Reason** and **Refine** phases.

These ablation experiments isolate each module's contribution to overall accuracy, robustness, and coherence in complex reasoning tasks.

B Dataset and Statistics

B.1 Experience Breakdown

AGENT KB draws from seven benchmark datasets spanning code reasoning, web navigation, multi-hop retrieval, and human-level evaluation. Table 5 summarizes each source, its original task count, and the number of experience entries generated after processing.

Table 5: Overview of datasets used to construct the experience knowledge base.

Dataset	Type	Tasks	Experiences
General QA experiences			
BrowseComp (Wei et al., 2025)	Web navigation	1,266	1,266
MultiHopRAG (Liu et al., 2025c)	Multi-hop reasoning	2,556	2,556
HLE (Phan et al., 2026)	Expert-level QA	3,000	~2,000
WebWalkerQA (Wu et al., 2025)	Open-domain QA	680	680
Software engineering experiences			
RepoClassBench (Deshpande et al., 2024)	Code understanding	100	1,000
SWE-Gym-Raw (Pan et al., 2025)	Code generation	100	1,000
RepoEval (Zhang et al., 2023)	Code completion	100	1,000
Total (approx.)		7,802	~9,502

BrowseComp. All 1,266 tasks from BrowseComp (`https://huggingface.co/datasets/smolagents/browse_comp`) are processed, yielding one experience entry per task. These experiences capture web browsing, information retrieval, and multimodal reasoning patterns.

MultiHopRAG. All 2,556 tasks from MultiHopRAG (`https://github.com/yixuantt/MultiHop-RAG/tree/main/dataset`) are incorporated, with one experience per task. These experiences focus on multi-hop reasoning and retrieval-augmented generation scenarios.

HLE. From the 3,000-task HLE benchmark (`https://huggingface.co/datasets/cais/hle`), we use the text-only subset, generating one experience per task. Non-textual tasks are excluded to maintain consistency in representation. These experiences cover human-level evaluation scenarios across diverse domains.

WebWalkerQA. All 680 tasks from WebWalkerQA (`https://huggingface.co/datasets/callanwu/WebWalkerQA`) contribute one experience entry each. These experiences capture web navigation and open-domain question-answering patterns.

RepoClassBench. We select 100 representative Python cases from RepoClassBench (`https://github.com/microsoft/repoclassbench`) that align with SWE-bench repositories. For each case, we generate 10 experiences capturing distinct solution approaches, yielding 1,000 structured entries. These experiences focus on repository classification and code understanding.

SWE-Gym-Raw. From SWE-Gym-Raw (`https://huggingface.co/datasets/SWE-Gym/SWE-Gym-Raw`), we select 100 diverse problem instances and generate 10 experiences per instance, producing 1,000 entries. These experiences primarily cover code generation and bug-fixing within Python repositories.

RepoEval. From RepoEval (`https://github.com/microsoft/CodeT/tree/main/RepoCoder/datasets`), we select 100 cases and generate 10 experiences per case, adding 1,000 entries. These experiences focus on code completion and repository-level programming tasks.

B.2 Inference Cost Breakdown

Tables 6 and 7 report the token usage and monetary cost of AGENT KB for GAIA and SWE-bench Lite, respectively.

Table 6: Per-task token and cost budget for GAIA validation (165 tasks). Shares cover the per-evaluation budget and exclude the one-off knowledge-base ingestion step. Pricing: $1.36 per million prompt tokens and $5.44 per million completion tokens. Values are averaged across 165 tasks.

Module	Prompt	Completion	Cost (USD)	Share (%)
Evaluation (per GAIA task)				
Action loop	205,873	41,912	0.51	98.2
Log summary	6,182	67	0.01	1.6
Planning	237	119	0.001	0.2
Feedback	294	123	0.001	0.2
Total (evaluation)	**212,586**	**42,221**	**0.52**	**100.0**
One-off setup				
AGENT KB construction	5,140,655	768,270	10.88	–

Table 7: Per-instance cost for SWE-bench lite when AGENT KB supplies hints (`GPT-4.1`). Max steps per issue fixed at 100. Token counts are per issue. Token pricing follows the same schedule as Table 6.

Hint Source	Prompts	Completions	Cost (USD)	Hint Length
RepoClassBench	6,543	912	0.0078	90
RepoClassBench (refine)	4,217	508	0.0028	130
Top-n SWE-Gym	2,847	296	0.0019	60
Top-n RepoClassBench	3,129	402	0.0021	70
Average	**4,184**	**530**	**0.0037**	**88**

On GAIA, the retrieval loop adds only $0.27 to the total evaluation cost of $86.0, less than 0.4% of

the per-run budget. The offline ingestion step is a one-time expense amortized over future runs. On SWE-bench Lite, hinting with AGENT KB costs less than $0.004 per issue (under 7,000 prompt tokens), keeping the marginal overhead well below one cent when used with `GPT-4.1`. These results confirm that AGENT KB enables cross-framework experience sharing at minimal additional cost.

C Examples

This section presents concrete examples that demonstrate how AGENT KB processes different queries and workflow types across domains and task types.

C.1 GAIA Task with Feedback Refinement

This example illustrates a complete AGENT KB workflow on a GAIA task, demonstrating both the initial planning stage and the feedback refinement stage. The task involves identifying EC numbers for enzymes used in a virus-testing method described in a 2016 paper on sweet potato viruses (SPFMV and SPCSV) in Uganda.

Workflow Overview. Initially, AGENT KB processed the query using its hybrid retrieval mechanism to generate a preliminary plan. The agent searched for the two most commonly associated chemicals but only successfully retrieved the EC number for one, as the other chemical was not an enzyme. Based on this incomplete retrieval, the system incorrectly concluded that the answer was only "3.1.3.1," leading to an erroneous response. Subsequently, the feedback stage provided refined planning that explicitly clarified the question pertained only to *enzymes* (since only enzymes have EC numbers). This guidance prevented the agent from searching for non-enzyme chemicals and enabled it to arrive at the correct answer in fewer steps. Table 8 presents the complete workflow progression.

Key Insights. This example demonstrates several critical capabilities of AGENT KB: (1) The feedback stage retrieves *different* experiences than the planning stage, focusing on error patterns rather than task similarity. (2) The refinement explicitly adds the constraint to search for **enzymes** rather than general chemicals, addressing the root cause of the initial failure. (3) The two-stage `Reason-Retrieve-Refine` approach allows the system to self-correct without requiring external supervision or oracle labels.

C.2 SWE-bench Retrieved Experiences

The following examples illustrate how AGENT KB stores and retrieves domain-specific knowledge for software engineering tasks from the SWE-bench dataset. Table 9 organizes two representative cases, each showing a problem description paired with relevant guidance retrieved from AGENT KB. These examples demonstrate the system's ability to provide contextual assistance for code debugging and modification tasks across different bug categories.

Analysis. These examples demonstrate AGENT KB's ability to provide targeted guidance for common software engineering challenges. The first example addresses API documentation generation issues related to context-dependent string escaping, emphasizing the need to check specific contexts (attribute names, parameter names) before applying escape logic. The second example focuses on debugging a symbolic mathematics library where global state management affects constructor behavior, highlighting the importance of maintaining mathematical correctness while preventing regressions. Notice how the retrieved knowledge provides specific, actionable advice rather than generic troubleshooting steps.

C.2.1 Raw Log to Experience Generation

This subsection demonstrates how AGENT KB transforms raw agent execution logs into reusable experiences stored in the knowledge base. Rather than relying on manually curated summaries, the system operates directly on execution traces, identifying failure patterns and consolidating corrective strategies into structured experience units. Table 10 shows the complete transformation pipeline for a bioinformatics task involving protein structure analysis. We include the full raw log to illustrate how execution errors arise in practice and how the agent adapts its strategy in response. In this example, the agent initially encounters an execution failure while parsing the PDB file, followed by a correction that resolves both line-indexing and unit-conversion issues. From this execution trace, AGENT KB extracts the following experience, which captures the actionable lesson in a form that can be reused across future tasks and agent frameworks.

Agent Experience

When calculating distance, extract only the first two ATOM coordinates directly without skipping any lines. If the task asks for the distance in Ångströms, rounded

Table 8: **Complete execution workflow for enzyme EC number identification task.** The table shows the progression through both `Reason`-`Retrieve`-`Refine` stages: initial planning (rows 1–5) and feedback refinement (rows 6–10). The feedback stage retrieves different experiences focused on error patterns and provides targeted corrections that lead to the correct answer.

Stage & Component	**Content**
Initial Planning Stage (`Reason`-`Retrieve`-`Refine`)	
`Reason` Input	What are the EC numbers of the two most commonly used chemicals for the virus testing method in the paper about SPFMV and SPCSV in the Pearl Of Africa from 2016?
`Reason` Keywords	EC numbers; virus testing method; SPFMV; SPCSV; Pearl Of Africa; 2016 paper; commonly used chemicals; chemical alphabetization
`Retrieve` Result	Similar task: phage genome database studies... Suggestions: Explicitly map experimental design to logical variables—use data frames or grid tables for clarity...
`Refine` Plan	1. Identify the specific 2016 paper on SPFMV and SPCSV in Uganda 2. Retrieve full text and check methodology section 3. Extract chemicals used, focusing on enzymes 4. Determine the two most commonly used ones 5. Alphabetize names and look up EC numbers 6. Return EC numbers separated by semicolon
Initial Prediction	`3.1.3.1;3.1.3.1` (Incorrect — same EC number repeated)
Feedback Stage (`Reason`-`Retrieve`-`Refine`)	
`Reason` Error Analysis	Detected that only one EC number was found, but question asks for two *enzymes*. The agent searched for general chemicals instead of specifically enzymes.
`Retrieve` Result	Similar task: peptide binding to integrin receptors... Suggestions: Always decompose the question to clarify biological context and functional motifs in sequence-based tasks...
`Refine` Guidance	1. Search using compact keywords: "SPFMV", "SPCSV", "Pearl of Africa", "2016" 2. Identify virus detection method (typically ELISA) and the **enzymes** used in labeling 3. Look up EC numbers, alphabetize, and return
Model Reasoning	Identified two common ELISA enzymes: alkaline phosphatase (3.1.3.1) and horseradish peroxidase (1.11.1.7). Alphabetically ordered: alkaline phosphatase first.
Final Output	`3.1.3.1;1.11.1.7` (Correct)

Table 9: **SWE-bench problem-solution pairs with retrieved experiences.** The table organizes two representative debugging scenarios, showing how AGENT KB retrieves targeted repair strategies for different bug types. Each example demonstrates cross-repository knowledge transfer with specific, actionable guidance.

Component	**Content**
Example 1: API Documentation String Escaping Bug	
Problem Description	In the project that automatically generates API documentation for Python projects, an extra backslash is inserted before underscores—for example, `hello_` is rendered as `hello\_`
Retrieved Experience	Check the string processing part in the relevant functions to ensure that escape is only carried out when necessary. For example, whether all parameters ending with _ need to be escaped, or whether different handling methods are required in certain specific contexts (such as attribute names, parameter names, etc.). When modifying conditions, not only the original conditions should be taken into account, but also factors such as configuration and context should be combined to ensure the accuracy of the logic.
Example 2: Symbolic Math Library Evaluation Bug	
Problem Description	Disabling evaluation globally with `with evaluate(False)` interferes with `sympify`'s string-parsing logic, preventing some integer expressions from being instantiated as integer objects.
Retrieved Experience	When adding or modifying a conditional check (such as for `evaluate` or imaginary coordinates), ensure the logic does not inadvertently skip important validation for invalid inputs (such as actual imaginary numbers), and only disables overly strict checks for valid real inputs. This is critical to maintain mathematical correctness while fixing the bug. (Most important) When changing the logic in constructors (like `Point`/`Point2D`), verify that the minimal change solves the immediate bug, does not introduce new regressions, and does not allow forbidden cases (e.g., actual imaginary coordinates).

to the nearest picometer, keep the original Å value with three decimal places precision, without converting back and forth between Å and pm. Output the distance directly in Ångströms, keeping the computed value with three decimals (e.g., x.xxx), and do not include units or symbols in the final answer.

This example illustrates how AGENT KB preserves domain-specific reasoning, error-recovery behavior, and formatting constraints, enabling successful adaptations to be reused rather than rediscovered.

C.2.2 Complex Query Processing Example

We next illustrate how AGENT KB organizes experiences for complex, multi-constraint queries that require coordination between multiple agent roles. Such queries are common in open-domain reasoning tasks, where identifying the correct entity and verifying multiple biographical constraints demands both high-level reasoning and targeted information retrieval.

Table 11 presents the structured representation produced by AGENT KB for a representative example. The system decomposes the query into explicit constraints, generates separate planning strategies for a general reasoning agent and a specialized search agent, and records role-specific experiences that reflect successful coordination patterns. Unlike single-agent traces, these experiences encode complementary behaviors: The general agent focuses on constraint decomposition and hypothesis formation, while the search agent emphasizes query refinement, source selection, and cross-validation. By storing these experiences separately but linking them through a shared representation, AGENT KB enables future agents to reuse effective coordination strategies without access to the original execution trace.

Together, the examples in Tables 10 and 11 show that AGENT KB captures not only task-specific solutions, but also reusable procedural knowledge for error recovery, precision management, and multi-agent collaboration. This abstraction allows experiences collected from heterogeneous agent frameworks to be shared and adapted across domains, supporting robust cross-framework knowledge transfer.

D Prompt Details

We provide the full prompt templates used in AGENT KB for reproducibility. Table 12 details the prompts used during AGENT KB construction: each prompt is designed to elicit structured agent plans, simulate realistic failure cases, and extract generalizable experience guidelines from both reasoning-oriented tasks (GAIA) and code repair tasks (SWE-bench).

Table 13 presents the stage-specific prompts used in the `Reason-Retrieve-Refine` pipeline. While task descriptions differ between GAIA and SWE-bench, the prompts share a standard cognitive structure: each stage operates on retrieved experiences represented as *(question, plan, experience)* triples and produces strictly formatted, actionable outputs for downstream reasoning.

All prompt templates are released with the codebase to facilitate reproducibility and adaptation to new agent frameworks and task domains.

Table 10: **Raw log to experience transformation.** Complete pipeline from raw execution trace to extracted experience for a protein structure analysis task.

Component	**Content**
Raw Log	`{ "agent_name": "gpt-4.1", "question": "Using the Biopython library in Python, parse the PDB file of the protein identified by the PDB ID 5wb7 from the RCSB Protein Data Bank. Calculate the distance between the first and second atoms as they are listed in the PDB file. Report the answer in Angstroms, rounded to the nearest picometer.", "prediction": "1.46", "true_answer": "1.456", "intermediate_steps": [{ "task": "You have one question to answer...", "step_type": "task" }, { "facts": "Here are the facts that I know so far...", "plan": "Here is the plan of action that I will follow...", "step_type": "planning" }, { "tool_calls": [{"id": "call_1", "type": "function", "function": {"name": "python_interpreter", "arguments": "..."}}], "error": {"type": "AgentExecutionError", "message": "Code execution failed..."}, "step_type": "action" }] }`
Error Pattern	Incorrect line extraction from PDB file; precision loss through unit conversions (Å ↔ pm)
Extracted Experience	When calculating distance, extract only the first two ATOM coordinates directly without skipping any lines. If the task asks for the distance in Ångströms, rounded to the nearest picometer, keep the original Å value with three decimal places precision, without converting back and forth between Å and pm. Output the distance directly in Ångströms, keeping the computed value with three decimals (e.g., x.xxx), do not round it to fewer decimal places. The output is just the number without any units or symbols, such as x.xxx, not x.xxx Ångströms.

Table 11: **Complex query processing with multi-agent coordination.** Complete structure showing query, generated JSON, and role-specific experiences.

Component	Content
Query	There's this popular figure with multiple Grammy awards in the entertainment industry, who put out their first album before 1969, had substance dependence, and was dismissed from school before they turned 20. Their first life partner died in 1997. They became uniformed personnel at some point in their lives. What's the name of the hospital they died? **True Answer:** St. John's Health Center
Generated JSON	

```
{
  "question": "There's this popular figure with multiple Grammy awards...",
  "true_answer": "St. John's Health Center",
  "agent_planning": "1. Parse the question to extract all key constraints: multiple
      Grammy awards, first album before 1969, substance dependence, dismissed from
      school before age 20, first life partner died in 1997, served as uniformed
      personnel, determine place/hospital of death.
  2. Conceptual plan:
  - Identify the possible entertainers matching all constraints.
  - For each candidate:
    a) Verify the timeline for first album release (before 1969)
    b) Check Grammy history
    c) Search biographical records for substance abuse and educational background
    d) Confirm information about life partner's death year and uniformed service
    e) Pinpoint the date and location/hospital of death of the matched figure.",
  "search_agent_planning": "1. Receive precise person identifier from Code Agent
      or use biographical clues to triangulate the subject.
  2. Formulate search queries for identification and specific hospital information.
  3. Prioritize official biographical sources, reputable news outlets, Grammy records.
  4. Cross-check critical data points to validate subject match.
  5. Extract facts about location and hospital of death from obituaries.",
  "agent_experience": [
    "Break down multifaceted questions into smaller constraint checks",
    "Explicitly log and verify biographical constraints with multi-source confirmation",
    "Select high-reliability sources for biographical and award data",
    "Delegate to Search Agent early with specific sub-queries",
    "Validate final answers by chaining all found facts back to original constraints"
  ],
  "search_agent_experience": [
    "Decompose complex queries into sequential search refinements",
    "Craft highly specific queries for ambiguous identifiers",
    "Favor authoritative sources over entertainment/tabloid content",
    "Cross-validate information from multiple independent sources",
    "Format results with direct attribution and clear source references"
  ]
}
```

Table 12: **Prompt templates used for AGENT KB construction.** The table contrasts different prompt formulations used to elicit structured agent planning, error simulation, and experience extraction.

Prompt Templates for GAIA

```
You will act as an advanced AI evaluation system tasked with analyzing a complex problem handled by a Code Agent with an embedded Search Agent. Your analysis will extract valuable insights from this process. Follow these instructions carefully:
1. I will provide a question and its correct answer (true_answer).
2. First, simulate the Code Agent's planning process in detail. Describe how it would:
- Break down the problem into logical components
- Determine which tools to use (code execution, data processing, API calls)
- Decide when to delegate to the Search Agent
- Plan data transformations and analysis steps
- Structure the final solution
Include specific reasoning steps, potential code snippets considered, and decision points. Only include content in the agent plan, without any other description.
3. Next, simulate the Search Agent's planning process in detail. Describe how it would:
- Parse the search query requirements from the Code Agent
- Formulate effective search queries
- Determine which sources to prioritize
- Extract and validate relevant information
- Process and structure the search results for the Code Agent
Include specific query formulation strategies and information filtering approaches. Only include content to search the agent plan, without any other description.
4. Based on the question and your simulated planning processes, create a realistic error scenario. Describe:
- Where and how the agents might fail
- Incorrect assumptions they might make
- Data misinterpretations or code errors
- Logical flaws in their approach
5. Finally, provide two sets of actionable experience guidelines:
- For the Code Agent: Specific principles to improve problem-solving, tool selection, verification, and integration of search results
- For the Search Agent: Specific principles to enhance query formulation, source evaluation, information extraction, and result formatting
The behavioral guidelines should be generalizable principles that would help the agents perform better on similar tasks, without directly revealing the specific answer to the question I provided.
Important: If the question does not require the search agent to solve, leave "search_agent_plan" and "search_agent_experience" empty in your response.
Output your complete analysis in the following JSON format with no additional text:
{
   "question": "<question I provide>",
   "true_answer": "<correct answer I provide>",
   "agent_plan": "<your detailed Code Agent plan simulation>",
   "search_agent_plan": "<your detailed Search Agent plan simulation>",
   "agent_experience": "<your actionable Code Agent guidelines>",
   "search_agent_experience": "<your actionable Search Agent guidelines>"
}
Here is an example:
{
   "question": "<question from hand-crafted experience pool>",
   "true_answer": "<correct answer>",
   "agent_plan": "<real Code Agent plan>",
   "search_agent_plan": "<real Search Agent plan>",
   "agent_experience": "<hand-crafted agent experience>",
   "search_agent_experience": "<hand-crafted search agent experience>"
}
```

Prompt Templates for SWE-bench

```
You are an advanced code repair analysis system tasked with constructing structured experiences for Agent KB from SWE-bench tasks. Given a natural language problem description, a model-generated fix, and supporting repair hints, follow the steps below to extract reusable knowledge entries.
1. I will provide a natural language problem description, a model-generated fix, and supporting repair hints.
2. First, perform code reconstruction:
- Given a detailed natural language description of a Python class or function, generate its correct implementation.
- Ensure the implementation is complete and syntactically valid.
- Output key: "code"
3. Next, perform error analysis and extract repair principles:
- You are given two versions of code: one with errors and one corrected.
- Analyze the differences and identify the key problems in the faulty version.
- Based on this comparison, produce a list of 10 code repair precautions.
- These should be generalizable principles addressing common issues (e.g., indentation, type conversion, exception handling, logic errors).
- Avoid using titles; output only the explanations.
- Output key: "repair_precautions" (as a list of 10 strings)
4. Next, classify each provided hint:
- Each natural language hint is used to prompt the LLM to repair the code.
- Classify each hint into one repair category (e.g., "syntax", "logic", "exception handling").
- Extract important keywords and write a one-sentence summary of the hint.
- Output key: "hint_analysis" (as a list of objects with keys "category", "keywords", "summary")
5. Next, identify the repair type(s) for this case:
- Given the original problem description, identify the {K} most relevant categories this code repair case falls under.
- Select from a pre-defined set of bug types.
- Output key: "categories" (as a list of {K} strings)
6. Finally, rank the most relevant hints:
- You are given a set of all the hints provided to the model.
- Analyze the model's generated fix and its reasoning trace.
- Identify the {N} most relevant hints, which may be either positively helpful or misleading.
- Sort them in order of influence on the final patch.
- Output key: "ranked_hints" (as a list of {N} strings)
Output your complete analysis in the following JSON format with no additional text:
{
   "code": "<correct implementation>",
   "repair_precautions": ["<precaution 1>", "...", "<precaution 10>"],
   "hint_analysis": [
     {"category": "<category>", "keywords": ["<kw1>", "<kw2>"], "summary": "<one-sentence summary>"}, ...
   ],
   "categories": ["<bug type 1>", "...", "<bug type K>"],
   "ranked_hints": ["<hint 1>", "...", "<hint N>"]
}
Here is an example:
{
   "code": "<reconstructed correct code>",
   "repair_precautions": ["...", "...", "...", "...", "...", "..."],
   "hint_analysis": [{"category": "logic", "keywords": ["off-by-one"], "summary": "Adjust loop bounds to include the final element."}],
   "categories": ["logic", "boundary conditions"],
   "ranked_hints": ["<most influential hint>", "<second most influential hint>", "..."]
}
```

Table 13: **Prompt templates for the AGENT KB pipeline.** We use stage-specific prompts for **Reason**-**Retrieve**-**Refine** across benchmarks.

	GAIA	SWE-bench

Reason	Analyze similar tasks and past experiences to generate concise, actionable suggestions for improving the current plan. Based on the patterns identified in relevant tasks and insights from the Agent KB, provide specific recommendations.
	
	Key Requirements:
	1. Focus exclusively on technical/behavioral improvements derived from similar task patterns and experience.
	2. Provide root-cause solutions and implementation strategies based on past successes.
	3. Format output strictly as:
	{1. Specific suggestion 1}
	{2. Specific suggestion 2}
	...
	
	No headings, explanations, or markdown.
	You can refer to similar tasks, plans, and corresponding experience to provide your suggestions:
	{
	"question": "<Question retrieved from Agent KB>",
	"agent_plan": "<Retrieved agent plan>",
	"agent_experience": "<Retrieved agent experience>",
	}
	...
		Analyze similar bug-fixing tasks and past patching experiences to generate concise, actionable suggestions for improving the current fix plan. Based on recurring failure patterns and successful resolutions stored in the Agent KB, provide specific recommendations tailored to the current issue.
		
		Key Requirements:
		1. Focus exclusively on technical improvements derived from similar bug patterns and historical fixes.
		2. Provide root-cause solutions and concrete implementation strategies (e.g., guard conditions, API usage, control flow).
		3. Format output strictly as:
		{1. Specific suggestion 1}
		{2. Specific suggestion 2}
		...
		
		No headings, explanations, or markdown.
		You may refer to similar issues, patch plans, and corresponding experience to provide your suggestions:
		{
		"question": "<SWE-bench issue title or description>",
		"agent_plan": "<Retrieved historical patch plan>",
		"agent_experience": "<Retrieved debugging or fix experience>"
		}
		...

Retrieve	Given the current task and the agent's preliminary plan, retrieve the most relevant historical experiences from Agent KB that can help complete the task.
	
	Retrieval Priorities:
	1. Prefer tasks with similar reasoning structure (e.g., multi-hop factual queries, temporal reasoning, entity comparison).
	2. Favor experiences that demonstrate successful decomposition, verification strategies, or evidence aggregation.
	3. Include agent plans that show effective use of external tools, intermediate checks, or uncertainty handling.
	
	Format each retrieved experience as:
	{
	"question": "<GAIA task or query description>",
	"agent_plan": "<Historical high-level reasoning plan or decomposition>",
	"agent_experience": "<Reasoning strategies, verification steps, or failure modes avoided>"
	}
	...
	
	Retrieve 3 to 5 relevant entries and return them in the above format for use in downstream reasoning and refinement.
		Given the current bug description, initial patch plan, and model thought process, retrieve the most relevant historical experiences from Agent KB.
		
		Retrieval Priorities:
		1. Prefer experiences with similar bug types (e.g., off-by-one errors, null pointer exceptions, wrong return value).
		2. Favor patches with successful unit test outcomes and generalizable fix patterns.
		3. Include agent plans that show tool usage, exception guards, or correct interface assumptions.
		
		Format each retrieved experience as:
		{
		"question": "<SWE-bench issue title or commit description>",
		"agent_plan": "<Historical high-level patch or thought process>",
		"agent_experience": "<Failure modes avoided or debug strategies that worked>"
		}
		...
		
		Retrieve 3 to 5 relevant entries and return them in the above format for use in downstream reasoning and refinement.

Refine	Analyze the execution logs to determine the causes of the agent's incorrect responses. Based on the findings of the log and insights from the provided similar tasks and experience, generate some concise, actionable suggestions that the agent must follow to improve accuracy.
	
	Key Requirements:
	1. Focus exclusively on technical/behavioral fixes derived from log patterns and the Agent KB.
	2. Provide root-cause resolution (e.g., code logic, data validation, API handling) as well as generic advice.
	3. Format output strictly as:
	{1. Specific suggestion 1}
	{2. Specific suggestion 2}
	...
	No headings, explanations, or markdown.
	You can refer to similar tasks and corresponding experience to provide your suggestions:
	{
	"question": "<Question retrieved from Agent KB>",
	"agent_plan": "<Retrieved agent plan>",
	"agent_experience": "<Retrieved agent experience>",
	}
	...
	
	Execution logs summary:
	<Log summary>
		Analyze the execution trace and patch attempt to determine the root causes of the agent's failure. Based on the trace, the generated patch, and insights from similar historical experiences in the Agent KB, produce concise, actionable constraints that the agent must follow in future attempts.
		
		Key Requirements:
		1. Focus exclusively on technical and behavioral root causes (e.g., incorrect API assumptions, missing validation, faulty control flow).
		2. Leverage similar bug-fixing experiences to identify patterns that led to failure or success.
		3. Translate the analysis into concrete behavioral constraints or correction rules.
		4. Format output strictly as:
		{1. Specific correction or constraint 1}
		{2. Specific correction or constraint 2}
		...
		
		No headings, explanations, or markdown.
		You may refer to similar tasks and corresponding experience:
		{
		"question": "<SWE-bench issue>",
		"agent_plan": "<Retrieved historical plan>",
		"agent_experience": "<Retrieved experience>"
		}
		...
		
		Execution trace summary:
		<Trace summary>