EXPHARNESS: MODEL-AGNOSTIC EXPERIENCE LEARNING THROUGH A TRAINABLE HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly operate within a harness, the scaffolding that determines what enters the executor's context, yet the experience they accumulate across tasks rarely flows back into this harness. Existing approaches include executor fine-tuning and external memory retrieval, but combining task-adaptive retrieval with experience reuse across frozen executors remains challenging. To this end, we introduce `ExpHarness`, a learnable experience harness that improves frozen and replaceable LLM executors without modifying their parameters. Specifically, `ExpHarness` distills trajectories into reusable skills and failure lessons within a self-evolving experience graph, and trains a lightweight retrieval copilot that decides, per task, how broadly to explore the graph and how strongly to favor historically useful experiences over merely similar ones. The copilot is optimized with reinforcement learning from a utility-grounded reward combining the with/without-experience score difference and an absolute-performance term; the same reward updates the graph during training. Extensive experiments on ExpSuite, spanning 10 static benchmarks and 2 agentic environments, show that `ExpHarness` improves over the strongest baseline by 12.1% and 4.5% on static tasks and by 21.4% and 12.7% on agentic tasks with the smaller and larger executors, respectively, while reducing interaction steps by up to 21.6%. Transfer experiments further examine reuse of the learned harness across executors of different scales and reasoning capabilities, with joint graph and copilot transfer performing closest to target-specific training among the evaluated transfer variants.

1 INTRODUCTION

LLM agents have demonstrated strong capabilities in complex tasks requiring reasoning, tool use, and multi-step environment interaction (Yao et al., 2022; Shinn et al., 2023). In practice, such an agent is a frozen executor LLM wrapped in a *harness*, the surrounding scaffolding that determines what information enters the executor's context at each step. A key bottleneck, however, is that this harness rarely learns: each task is solved largely from scratch, and successful strategies, failure lessons, and transferable insights accumulated from prior interactions are discarded rather than systematically reused. A natural remedy is to fine-tune the executor on its own experience, but this solution becomes increasingly inflexible as LLM capabilities evolve rapidly. Whenever a stronger executor is released, executor-specific training may need to be repeated, and many capable LLMs are too large or inaccessible for parameter-level modification in the first place. These observations motivate a different locus for experience learning: *rather than training the executor, can an LLM agent learn from accumulated experience by training its harness, keeping the executor itself frozen and replaceable?*

Existing experience learning methods, as summarized in Table 1, provide partial solutions to this problem, including textual experience distillation (Zhao et al., 2024), memory organization (Chhikara et al., 2025), utility-aware experience selection (Zhang et al., 2026), and adaptive retrieval or search policies (Jiang et al., 2025b). These methods emphasize different parts of the experience pipeline. We study how relational storage, adaptive retrieval, and outcome-based feedback can be combined in an external harness and evaluated when the executor changes. Building such an experience harness is non-trivial for three reasons. *First, surface relevance is not the same as experience utility.* An experience that appears similar to the current task may provide little downstream benefit (Shi et al., 2023; Cuconasu et al., 2024), while a useful experience may encode a transferable strategy, shared

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

Table 1: **Comparison with representative experience learning methods.** The columns summarize four design features for the representative methods listed. `ExpHarness` combines an experience graph, structure-aware expansion, utility-aware ranking, and adaptive retrieval around a frozen executor.

Method	Relational Experience Store	Structure-Aware Expansion	Utility-Aware Ranking	Adaptive Retrieval
ExpeL (Zhao et al., 2024)	✗	✗	✗	✗
Mem0 (Chhikara et al., 2025)	✓	✗	✗	✗
MemRL (Zhang et al., 2026)	✗	✗	✓	✗
S3 (Jiang et al., 2025b)	✗	✗	✗	✓
`ExpHarness`	✓	✓	✓	✓

sub-goal, or failure pattern that is not among the nearest neighbors of the task embedding, since past experiences are often connected through common strategies, environmental constraints, or recurring mistakes rather than surface similarity (Gutiérrez et al., 2024). *Second, the harness must adapt to each task.* Some tasks benefit from broad exploration over related experiences, while others require focused selection of a few high-utility ones, so a fixed retrieval rule cannot serve all tasks (Mallen et al., 2023; Jeong et al., 2024). *Third, the harness must remain useful when the executor changes.* Executors vary in reasoning, planning, and instruction-following ability, and a harness learned with one executor should carry over to a stronger, weaker, or reasoning-oriented executor without being rebuilt from scratch (Wei et al., 2023; Sclar et al., 2024).

To address these challenges, we propose **`ExpHarness`**, a learnable experience harness that improves frozen LLM executors without modifying their parameters. Rather than updating the executor, `ExpHarness` treats it as a replaceable task solver and places all learning in the harness, so that the learned harness can be carried over rather than rebuilt when stronger LLMs become available. Specifically, the harness comprises two core components—an experience graph and a retrieval copilot—jointly updated through utility-grounded feedback. First, the *Experience Graph*, a graph-structured experience store, stores compact experience units summarized from historical trajectories, including skills distilled from successful trajectories and lessons distilled from failures, and links related units so that retrieval can reach experiences sharing strategies, sub-goals, or failure patterns beyond the nearest neighbors of the task. Second, a lightweight *Retrieval Copilot* predicts task-adaptive controls that determine how broadly retrieval explores the graph and how strongly the final ranking favors historical utility over semantic relevance. To update both components, *utility-grounded feedback* combines the executor's score with retrieved experiences and its score difference relative to a no-experience run. This feedback optimizes the copilot with reinforcement learning and updates shared reward estimates for the retrieved entries, supplementing semantic relevance with historical task outcomes.

We evaluate `ExpHarness` on ExpSuite, covering both single-turn static tasks (question answering, mathematical reasoning, and code generation) and multi-step agentic environments (ALFWorld and AppWorld). Across static tasks, `ExpHarness` improves over the strongest baseline by 12.1% with the smaller executor and 4.5% with the larger executor. The advantage becomes more pronounced in agentic environments, where `ExpHarness` improves the weighted average score over the strongest baseline by 21.4% and 12.7% with the smaller and larger executors, respectively, while reducing average interaction steps by 12.7% and 21.6% compared with the competing baseline with the lowest average step count. Furthermore, transfer experiments compare reuse of the store, copilot, and full harness across executors, with joint transfer performing closest to target-specific training among the evaluated variants. Ablation studies show performance degradation under four variants that modify experience filtering, memory organization, graph diffusion, and utility-aware ranking.

2 Preliminaries

2.1 Model-Agnostic LLM Agents

We consider an LLM agent that solves a task $x \in \mathcal{X}$ by producing an output y, which can be an action, answer, or code sequence, and receives a task score $s = S(x, y) \in \mathbb{R}$ from the environment or evaluator. The agent is built around an executor LLM π_{exec}, which maps the task input to an output, i.e., $y = \pi_{\text{exec}}(x)$. We adopt a *model-agnostic* view of the executor: the experience learning mechanism should not depend on the internal architecture, parameters, gradients, logits, or training procedure of π_{exec}. Instead, π_{exec} is treated as a frozen and replaceable task solver, which can be closed-source, costly to update, specialized for a domain, or newly released. The

improvement mechanism interacts with the executor only through its input-output behavior and task-level feedback. This setting decouples experience learning from executor training, allowing the same external experience system to improve different frozen executors through input context rather than parameter updates.

2.2 Experience-Augmented Learning

To improve the executor through its input, we equip the agent with an external experience system $\mathcal{M}$ that stores reusable knowledge distilled from historical trajectories. A trajectory is denoted as $\tau = (x, \xi, y, s)$, where x is the task input, ξ denotes the intermediate execution process, y is the final response or action sequence, and $s \in \mathbb{R}$ is the task score. Specifically, ξ corresponds to the agent interaction trace in agentic environments, and to the intermediate thinking process in question-answering or reasoning tasks. Each trajectory is summarized into a compact natural-language experience unit $e = \mathrm{Summarize}(\tau)$ and stored in $\mathcal{M}$. At execution time, the agent retrieves a subset of experiences $E \subseteq \mathcal{M}$ and injects them into the executor's input context, yielding $y = \pi_{\mathrm{exec}}(x, E)$. The task score $s = S(x, y)$ then depends indirectly on the choice of E.

Since π_{exec} is fixed, the learnable component is the *retrieval policy* π_{ret}, which selects E from $\mathcal{M}$ given the current task. The learning objective is to maximize expected task performance over a task distribution $\mathcal{D}$:

$$\max_{\pi_{\mathrm{ret}}} \ \mathbb{E}_{x\sim\mathcal{D}}[\, S(x,\, \pi_{\mathrm{exec}}(x,\, E))\,], \quad E \sim \pi_{\mathrm{ret}}(\cdot \mid x, \mathcal{M}). \tag{1}$$

This formulation decouples execution from experience learning: π_{exec} solves the task, while π_{ret} decides which experiences should be provided as context.

3 ExpHarness: A Learnable Experience Harness for Frozen Executors

3.1 Overview

`ExpHarness` is a learnable experience harness for LLM agents. Given a task x, `ExpHarness` improves a frozen executor LLM π_{exec} by deciding which external experiences enter its context, without modifying the executor. The key idea is to decouple execution from experience learning: all learnable state resides in the harness $\mathcal{H} = (G, \pi_{\mathrm{ret}})$, consisting of an experience graph G and a retrieval copilot π_{ret}, which allows the same interface to be used with different frozen executors; the effectiveness of transfer is evaluated empirically. As shown in Figure 1, these two components are coupled by a feedback-driven training loop. First, historical trajectories are compressed into experience units and maintained in an experience graph (§3.2). Second, the retrieval copilot predicts task-adaptive controls to search the graph via semantic seeding, structure-aware expansion, and utility-aware ranking (§3.3). Third, downstream feedback updates both the copilot and the utility statistics of the graph, biasing future retrieval toward experiences that are empirically useful rather than merely semantically related (§3.4).

3.2 Experience Graph Construction and Maintenance

Trajectory-to-experience conversion. Let a historical trajectory be denoted as $\tau = (x, \xi, y, s)$, where x is the task input, ξ denotes the intermediate execution process, y is the final response or action sequence, and $s \in \mathbb{R}$ is the task score returned by the environment or evaluator. Specifically, ξ corresponds to the agent interaction trace in agentic environments, and to the intermediate thinking process in question-answering or reasoning tasks. `ExpHarness` converts each trajectory into a compact natural-language experience unit:

$$e = \mathrm{Summarize}(\tau). \tag{2}$$

The summarizer does not aim to preserve the full trajectory. Instead, it extracts reusable knowledge from the trajectory. High-scoring trajectories are distilled into *skills*, such as successful reasoning patterns, planning strategies, or task-specific heuristics. Low-scoring trajectories are distilled into *lessons*, such as failure modes, invalid actions, or constraints to avoid.

Each experience unit becomes an entry $v = (e_v, h_v, u_v, n_v) \in V$ in the experience graph, where e_v is the textual experience, h_v is its embedding, u_v is its shared-reward estimate, and n_v is its retrieval count. The utility u_v and count n_v are updated online from downstream feedback, allowing the graph to gradually distinguish useful experiences from merely relevant ones.

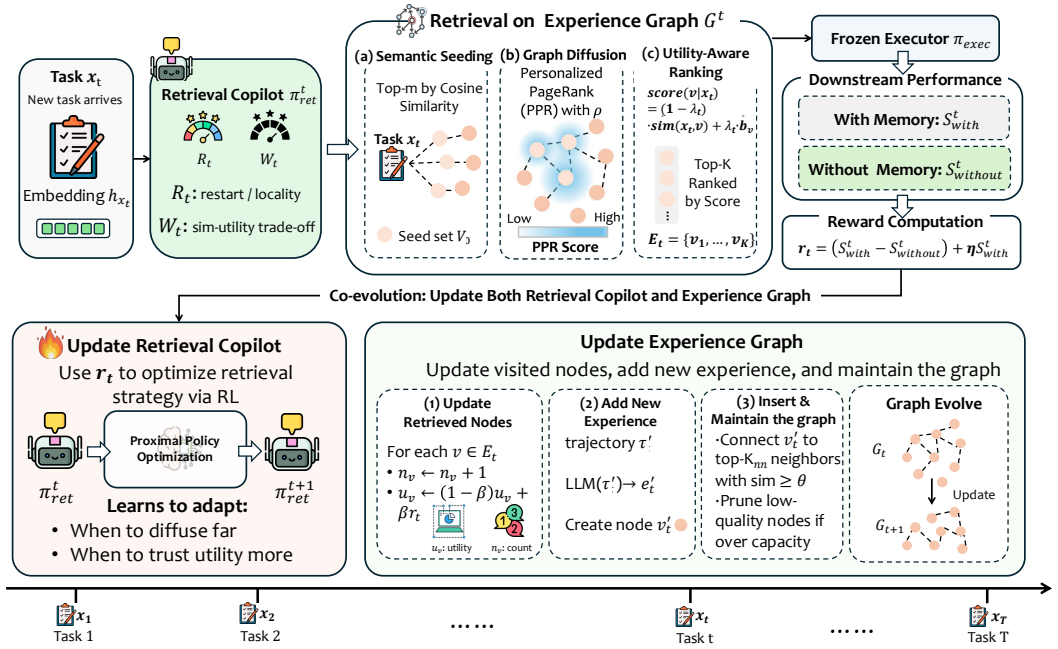


Figure 1: **Overview of `ExpHarness`.** `ExpHarness` enables a frozen and replaceable executor LLM to improve through a learnable experience harness, consisting of a self-evolving experience graph and a trainable retrieval copilot. For each incoming task x_t, the task is embedded as h_{x_t} and passed to the retrieval copilot π_{ret}^t, which predicts two adaptive controls: R_t for restart/locality control and W_t for the similarity–utility trade-off. Retrieval is performed on the current experience graph G^t through three steps: **(a) Semantic Seeding**, which selects a seed set V_0 by cosine similarity; **(b) Graph Diffusion**, which expands from the seeds using personalized PageRank controlled by ρ; and **(c) Utility-Aware Ranking**, which combines semantic relevance and utility confidence to select the top-K experiences E_t. During training, the frozen executor π_{exec} is evaluated with and without retrieved experiences, producing s_{with}^t and s_{without}^t. Their difference, together with the absolute task score, forms the reward $r_t = (s_{\text{with}}^t - s_{\text{without}}^t) + \eta s_{\text{with}}^t$. This reward drives a co-evolution process: it updates the retrieval copilot via PPO and updates the experience graph by refining visited-node utilities, adding new experience nodes, connecting them to similar neighbors, and pruning low-quality nodes when necessary. The feedback and update paths operate during training. At evaluation, the copilot and graph are fixed and only the experience-conditioned execution path is used; the executor LLM remains frozen throughout.

Relational organization. Rather than keeping entries as isolated records, `ExpHarness` links related experiences so that retrieval can later move from a matched entry to its neighbors. Formally, the experience graph is a sparse undirected graph $G = (V, E)$. When inserting a new entry v_i, `ExpHarness` connects it to semantically similar existing entries:

$$(v_i, v_j) \in E \iff v_j \in \mathcal{N}_{K_{\text{nn}}}(v_i) \text{ and } \cos(h_{v_i}, h_{v_j}) \geq \theta, \tag{3}$$

where $\mathcal{N}_{K_{\text{nn}}}(v_i)$ denotes the K_{nn} nearest neighbors of v_i under cosine similarity, and θ is a similarity threshold. This construction keeps the graph sparse while preserving local neighborhoods of related experiences.

3.3 Copilot-Controlled Utility-Guided Retrieval

Retrieval copilot. `ExpHarness` trains a lightweight retrieval copilot π_{ret} to control how the graph is searched. The copilot never produces the final answer or action; this is always done by the frozen executor π_{exec}. Instead, given a task x, the copilot outputs two discrete control variables:

$$(R, W) \sim \pi_{\text{ret}}(\cdot \mid x), \tag{4}$$

where $R \in \{0, \ldots, 100\}$ controls the restart/locality of graph diffusion and $W \in \{0, \ldots, 100\}$ controls the trade-off between semantic relevance and historical utility. We rescale them as $\rho = R/100$ and $\lambda = W/100$, where $\rho, \lambda \in [0, 1]$. Intuitively, ρ controls how closely retrieval stays to the initial semantic seeds: a larger ρ induces a higher restart probability, confining expansion to the immediate

neighborhood of the seeds, while a smaller ρ allows probability mass to propagate further across the graph; λ determines how strongly the final ranking favors historically useful experiences over purely similar ones.

Semantic seeding. Given task x with embedding h_x, `ExpHarness` first retrieves the top-m nodes by cosine similarity:

$$V_0 = \text{Top-m}_{v \in V} \cos(h_x, h_v). \tag{5}$$

The seed set V_0 provides an initial set of task-relevant experiences. However, these seeds are not directly used as the final retrieval result, because high semantic similarity does not necessarily imply high downstream utility.

Structure-aware expansion. Starting from V_0, `ExpHarness` expands retrieval along the relational links of the graph via graph diffusion, implemented as personalized PageRank. The restart distribution is defined as

$$q(v) = \begin{cases} 1/|V_0|, & v \in V_0, \\ 0, & v \notin V_0, \end{cases} \tag{6}$$

and the diffusion iterates as

$$p_{t+1} \;=\; \alpha(\rho)\, q \;+\; \big(1 - \alpha(\rho)\big)\, A_{\text{norm}}^{\top}\, p_t, \tag{7}$$

where A_{norm} is the row-normalized adjacency matrix and $\alpha(\rho) \in (0, 1)$ is a restart probability that increases monotonically with ρ. A larger ρ yields a higher restart probability, which confines probability mass closer to the semantic seeds and reduces the scope of diffusion; a smaller ρ allows the distribution to spread further across the graph. After convergence, the top-ranked nodes form the candidate set $C = \text{TopL}_{v \in V}\, p(v)$.

Utility-aware ranking. For each candidate $v \in C$, `ExpHarness` estimates its utility with an upper-confidence score:

$$b_v = u_v + c\sqrt{\frac{\log(N_{\text{ret}} + 1)}{\max(n_v, 1)}}, \tag{8}$$

where N_{ret} is the total number of retrieval events and $c > 0$ is an exploration coefficient. This score encourages the retriever to exploit experiences with high estimated utility while still exploring under-tested nodes.

The final retrieval score combines semantic relevance and utility confidence:

$$\text{score}(v \mid x) \;=\; (1 - \lambda)\, \widehat{\text{sim}}(x, v) \;+\; \lambda\, \widehat{b}_v, \tag{9}$$

where $\widehat{\text{sim}}(x, v)$ and $\widehat{b}_v$ are normalized within C. The retrieved experience set is

$$E_{R,W}(x) = \text{TopK}_{v \in C}\ \text{score}(v \mid x), \tag{10}$$

and its textual contents are concatenated with the task input and passed to the frozen executor:

$$y_{\text{with}} = \pi_{\text{exec}}\big(x,\, E_{R,W}(x)\big). \tag{11}$$

3.4 Harness Learning from Utility Feedback

Utility-grounded reward. To learn which retrieved experiences are actually useful, `ExpHarness` evaluates the frozen executor under two conditions. With retrieved experiences:

$$s_{\text{with}} = S\big(x,\, \pi_{\text{exec}}(x,\, E_{R,W}(x))\big), \tag{12}$$

and without any retrieved experience:

$$s_{\text{without}} = S\big(x,\, \pi_{\text{exec}}(x,\, \emptyset)\big), \tag{13}$$

where $S(\cdot)$ denotes the task-specific evaluator. The retrieval reward is

$$r(x, E_{R,W}) = s_{\text{with}}(x, E_{R,W}) - s_{\text{without}}(x) + \eta\, s_{\text{with}}(x, E_{R,W}). \tag{14}$$

The score difference $s_{\text{with}} - s_{\text{without}}$ records the observed change relative to the no-experience run, while ηs_{with} adds an absolute-performance term. The coefficient $\eta \geq 0$ sets its weight. The combined reward is therefore not a pure estimate of retrieval gain: when $s_{\text{with}} = s_{\text{without}} > 0$ and $\eta > 0$, it remains positive.

Table 2: **Performance comparison on ExpSuite-Static with 10 diverse tasks.** Results are grouped by task category: Question Answering, Mathematical Reasoning, and Code Generation. Weighted Avg. weights each benchmark score by its number of test instances (Table 5); all scores are percentages. **Bold** and underline denote the best and second-best results.

Model	Method	Question Answering					Reasoning			Coding		
		ARC-C	CommonsenseQA	GPQA-Diamond	MMLU	OBQA	GSM8K	GSM-Symbolic	MATH	HumanEval+	MBPP+	**Weighted Avg.**
Llama-3.2-3B-Instruct (Small LLM)	No Memory	51.56	54.44	18.33	42.89	54.22	69.56	60.44	37.56	43.59	38.75	51.88
	Retrieval-Centric Experience Learning Baselines											
	ReasoningBank	71.78	61.56	21.67	54.00	72.67	70.44	56.44	45.56	51.28	57.50	60.83
	ExpeL	44.67	57.78	16.67	46.44	43.11	77.11	80.44	23.33	20.51	22.50	51.49
	LightMem	71.56	**64.00**	26.67	58.00	72.89	44.89	42.00	49.11	48.72	42.50	56.47
	Mem0	68.00	59.56	**28.33**	56.44	65.78	52.00	39.78	28.22	41.03	60.00	52.42
	AWM	72.44	60.44	15.00	53.33	69.33	55.33	38.89	45.78	53.85	60.00	55.81
	MemRL	67.33	60.00	23.33	40.22	67.11	83.78	68.00	55.33	46.15	57.50	62.06
	LLM-Centric Experience Learning Baselines											
	IRCoT	62.67	59.78	15.00	38.00	63.33	77.78	72.67	29.56	51.28	50.00	56.65
	Search-o1	66.67	60.22	**28.33**	50.00	62.44	78.44	73.33	38.44	30.77	28.75	59.63
	S3	64.89	59.11	23.33	44.00	64.00	82.00	74.00	53.11	48.72	55.00	61.94
	ExpHarness	**74.00**	63.11	26.67	**60.00**	**74.00**	**84.89**	**82.00**	**57.11**	**56.41**	**62.50**	**69.57**
Llama-3.1-8B-Instruct (Large LLM)	No Memory	70.22	63.11	23.33	53.11	69.78	70.89	62.67	39.78	43.59	58.75	60.41
	Retrieval-Centric Experience Learning Baselines											
	ReasoningBank	82.22	70.44	28.33	63.78	78.44	80.67	74.22	47.56	58.97	62.50	69.93
	ExpeL	82.67	**75.78**	21.67	66.67	82.67	91.11	88.89	45.78	35.90	66.25	74.53
	LightMem	81.56	72.22	23.33	60.89	80.89	81.56	72.00	49.11	51.28	68.75	70.02
	Mem0	83.33	70.00	20.00	67.11	80.67	90.67	86.00	47.78	64.10	77.50	74.02
	AWM	81.56	74.00	18.33	60.22	78.67	81.78	76.44	47.11	64.10	76.25	70.47
	MemRL	84.44	73.78	23.33	67.56	84.00	93.33	80.00	56.44	35.90	65.00	75.34
	LLM-Centric Experience Learning Baselines											
	IRCoT	83.11	75.33	**30.00**	67.11	82.22	88.67	84.00	52.00	48.72	71.25	74.80
	Search-o1	81.33	71.78	28.33	64.44	78.00	89.33	84.67	48.00	46.15	63.75	72.54
	S3	82.00	73.56	26.67	65.56	81.56	90.89	86.00	52.89	58.97	72.50	74.89
	ExpHarness	**86.00**	75.11	28.33	**68.89**	**86.00**	**95.11**	**90.00**	**58.00**	**66.67**	**78.75**	**78.76**

Copilot policy optimization. The retrieval copilot is optimized with PPO (Schulman et al., 2017) to maximize the expected utility-grounded reward:

$$\max_{\pi_{\text{ret}}} \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{(R,W)\sim\pi_{\text{ret}}(\cdot|x)}[\, r(x,\, E_{R,W})\,]\,. \tag{15}$$

This objective trains retrieval controls using downstream task scores rather than semantic similarity alone. The executor LLM π_{exec} is never updated.

Experience graph evolution during training. The same reward also updates the experience graph, so that both halves of the harness co-evolve from a single signal. For each retrieved entry $v \in E_{R,W}(x)$, `ExpHarness` updates its retrieval count and utility estimate:

$$n_v \leftarrow n_v + 1, \tag{16}$$

$$u_v \leftarrow (1-\beta)\, u_v + \beta\, r(x,\, E_{R,W}), \tag{17}$$

where $\beta \in (0, 1]$ is the update rate. Each retrieved entry receives the same set-level reward. Thus, u_v is a historical shared-reward estimate for retrieval sets containing v, rather than an isolated causal contribution of that entry; it also reflects the absolute-performance term in Eq. 14. In addition, the completed trajectory $\tau' = (x, \xi', y_{\text{with}}, s_{\text{with}})$ is summarized into a new candidate experience entry, filtered for near-duplicates, and inserted into the graph using Eq. 3. When the graph exceeds its capacity budget, entries with low utility and low retrieval frequency are evicted. In this way, the graph continuously absorbs new skills and lessons while suppressing experiences that are rarely useful (see full training algorithm in Appendix C).

4 EXPERIMENTS

We evaluate `ExpHarness` on ExpSuite, comprising ExpSuite-Static for single-turn reasoning and generation tasks and ExpSuite-Agentic for multi-step interactive decision-making. Across both settings, the executor LLM is kept frozen, and all methods are evaluated without manually curated few-shot examples. We compare against prompting baselines, retrieval-centric experience learning methods, and LLM-centric experience learning methods. Dataset statistics, implementation details, and extended analysis are provided in Appendix E, B, and G.

Table 3: **Performance comparison on ExpSuite-Agentic across ALFWorld and AppWorld tasks.** We report success rate (SR) for ALFWorld, pass rate (PR) for AppWorld, and the weighted average score across all test tasks (weighted by the number of test tasks in each split; Table 6); all SR, PR, and Score values are percentages. **Bold** and underline denote the best and second-best results.

	Qwen3-32B (Small LLM)										Gemini-3.1-Flash-Lite (Large LLM)									
	ALFWorld				AppWorld				Avg.		ALFWorld				AppWorld				Avg.	
	ALF-Seen		ALF-Unseen		Test-N		Test-C				ALF-Seen		ALF-Unseen		Test-N		Test-C			
Method	**SR**	**#Steps↓**	**SR**	**#Steps↓**	**PR**	**#Steps↓**	**PR**	**#Steps↓**	**Score**	**#Steps↓**	**SR**	**#Steps↓**	**SR**	**#Steps↓**	**PR**	**#Steps↓**	**PR**	**#Steps↓**	**Score**	**#Steps↓**
No-Memory	19.3	41.2	35.8	42.5	30.1	30.8	21.6	31.5	25.1	34.7	62.7	27.9	64.9	26.3	34.2	33.0	23.2	36.6	38.3	32.9
Prompt-based Agentic Baselines																				
ReAct	31.4	36.7	47.0	38.6	30.0	32.2	21.8	31.9	28.9	33.8	65.7	24.1	63.4	25.1	32.7	24.8	32.1	16.1	42.6	20.5
Reflexion	23.6	40.6	40.3	38.7	31.6	31.3	21.9	32.7	26.9	34.6	63.1	23.3	60.4	23.7	48.2	21.9	31.1	15.6	44.2	19.4
Retrieval-Centric Experience Learning Baselines																				
ReasoningBank	37.9	35.3	26.1	42.6	29.4	25.6	22.2	24.3	26.8	29.2	67.2	29.1	81.3	24.1	45.3	23.1	41.2	25.5	52.5	25.4
ExpeL	54.3	27.5	29.1	41.4	26.7	35.1	28.2	25.6	32.3	30.2	71.6	21.4	63.4	28.7	43.6	21.1	29.9	28.3	44.6	25.8
LightMem	49.2	29.6	25.0	39.2	28.5	34.2	23.7	37.5	29.0	35.8	70.1	27.8	70.9	27.8	34.2	31.1	29.7	34.2	43.6	31.6
Mem0	30.0	37.3	18.7	42.8	20.8	33.4	15.7	35.2	19.5	36.4	58.2	30.6	63.4	22.4	32.7	31.5	22.9	34.8	36.9	31.5
AWM	37.9	35.3	32.8	40.1	28.0	28.8	22.8	33.0	27.8	33.7	73.1	24.7	80.6	18.8	50.7	20.9	31.5	25.8	49.7	23.6
MemRL	25.7	40.3	20.2	49.6	45.2	18.6	28.9	24.7	30.2	29.9	61.9	28.9	71.6	26.1	40.5	13.5	31.7	16.7	44.6	19.5
LLM-Centric Experience Learning Baselines																				
IRCoT	33.6	37.9	27.6	40.6	52.8	11.4	36.1	17.8	37.6	23.4	75.4	27.0	87.3	19.6	46.9	18.7	34.9	25.4	52.0	23.4
Search-o1	32.1	38.0	11.2	47.3	45.4	16.5	26.5	14.2	28.7	23.7	76.1	22.4	79.4	23.5	52.3	20.7	39.8	24.6	54.3	23.3
S3	58.6	23.5	42.5	22.5	55.4	9.7	35.0	15.0	44.0	16.5	77.1	21.8	80.6	21.5	53.0	18.5	40.8	22.0	55.3	21.1
`ExpHarness`	**70.0**	**20.0**	**75.4**	**18.0**	**57.1**	**8.8**	**39.3**	**13.6**	**53.4**	**14.4**	**85.0**	**17.6**	**88.1**	**17.0**	**57.1**	**12.8**	**48.4**	**14.8**	**62.3**	**15.2**

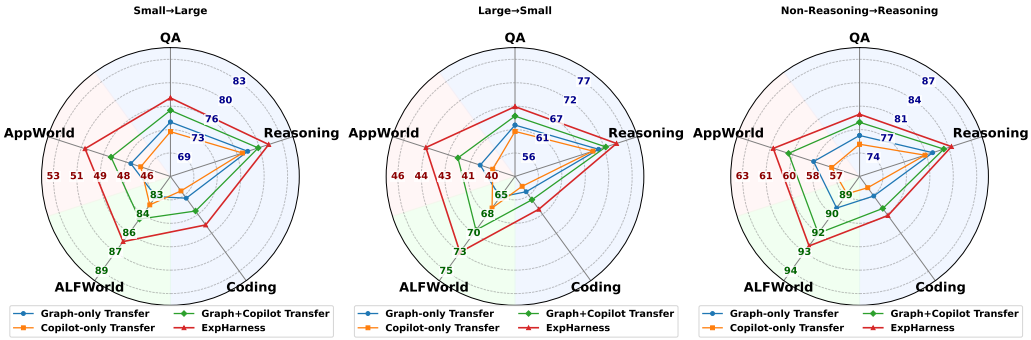
Task description. ExpSuite consists of two settings. **(i) ExpSuite-Static** includes ten benchmarks across three categories: question answering (ARC-C (Clark et al., 2018), CommonsenseQA (Talmor et al., 2019), GPQA-Diamond (Rein et al., 2024), MMLU (Hendrycks et al., 2021a), OBQA (Mihaylov et al., 2018)), mathematical reasoning (GSM8K (Cobbe et al., 2021), GSM-Symbolic (Mirzadeh et al., 2025), MATH (Hendrycks et al., 2021b)), and code generation (HumanEval+ (Liu et al., 2023), MBPP+ (Liu et al., 2023)). We report accuracy, exact match, and Pass@1 (Chen et al., 2021) respectively. **(ii) ExpSuite-Agentic** includes ALFWorld (Shridhar et al., 2020) (Seen/Unseen splits; SR and #Steps) and AppWorld (Trivedi et al., 2024) (Test-Normal/Test-Challenge splits; PR and #Steps). For executor models, we use Llama-3.2-3B-Instruct (Small LLM) and Llama-3.1-8B-Instruct (Large LLM) (Grattafiori et al., 2024) for ExpSuite-Static, and Qwen3-32B (Small LLM) (Yang et al., 2025) and Gemini-3.1-Flash-Lite (Large LLM) (DeepMind, 2026) for ExpSuite-Agentic.

Baselines. We compare `ExpHarness` with four groups of baselines. **(i) No-Memory** directly uses the frozen executor without external experiences. **(ii) Retrieval-centric** baselines maintain external experience records, including ReasoningBank (Ouyang et al., 2026), ExpeL (Zhao et al., 2024), LightMem (Fang et al., 2026), Mem0 (Chhikara et al., 2025), AWM (Wang et al., 2024), and MemRL (Zhang et al., 2026). **(iii) LLM-centric** baselines use an LLM-based policy to retrieve or exploit external information, including IRCoT (Trivedi et al., 2023), Search-o1 (Li et al., 2025), and S3 (Jiang et al., 2025b). **(iv) Prompt-based agentic** baselines (ExpSuite-Agentic only) include ReAct (Yao et al., 2022) and Reflexion (Shinn et al., 2023). For retrieval-centric baselines, the memory-construction protocol provides the same underlying historical trajectories wherever applicable (Appendix B). The resulting numbers and lengths of memory items can differ across representations.

4.1 ExpHarness Outperforms General Prompt-based Baselines and Experience Learning Methods

We evaluate `ExpHarness` on ExpSuite, covering both single-turn static tasks and multi-step agentic environments. Results are reported in Table 2 and Table 3. We have the following observations.

`ExpHarness` Achieves the Highest Weighted Average Scores in the Evaluated Settings. On ExpSuite-Static, `ExpHarness` improves the test-instance-weighted average score over MemRL, the strongest baseline under this aggregation, by 12.1% with the small executor and 4.5% with the large executor. These are relative improvements, computed as $(S_{\text{ours}} - S_{\text{baseline}})/S_{\text{baseline}}$; they do not imply the best result on every individual benchmark. On ExpSuite-Agentic, the corresponding relative improvements over S3 are 21.4% and 12.7%. Average environment interaction steps are 12.7% and 21.6% lower than those of the baselines with the lowest average step counts, respectively. Step counts describe environment interaction and do not include all retrieval, LLM inference, or training costs; they can also vary with task success.



(a) (b) (c)

Figure 2: **Harness transfer across different executor shifts.** (a) *Small-to-large transfer*: transferring the learned harness (experience graph and retrieval copilot) from a smaller executor to a larger executor. (b) *Large-to-small transfer*: transferring harness components from a larger executor to a smaller executor. (c) *Non-reasoning-to-reasoning transfer*: transferring harness components across executors with different reasoning capabilities.

Relative Gains Vary Across Executors and Task Settings. In these evaluations, the relative gains are larger for the executor labeled small within each setting, and larger in ExpSuite-Agentic than in ExpSuite-Static. These comparisons are descriptive: executor identities, benchmarks, and baseline scores differ across settings. They are consistent with useful experience reuse in interactive tasks, but do not isolate model scale or task horizon as the cause of the differences.

4.2 The Learned Harness Transfers Across Different LLM Executors

Because all learning in `ExpHarness` resides in the harness, a crucial question is whether a harness learned with one executor remains useful when the executor is replaced. To investigate this, we compare harness transfer under three settings. *Small-to-large transfer* uses the small executor as source and the large executor as target, testing whether experiences from cheaper models benefit stronger frozen LLMs. *Large-to-small transfer* reverses this direction, testing whether high-quality experiences from stronger models remain useful for weaker executors. *Non-reasoning-to-reasoning transfer* uses non-reasoning executors as source and reasoning-capable executors as target, testing whether experiences transfer across reasoning capability gaps. Specifically, for ExpSuite-Static, we transfer from Llama-3.1-8B-Instruct to DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025); for ExpSuite-Agentic, we transfer from Gemini-3.1-Flash-Lite to Claude-Sonnet-4 (Anthropic, 2025). For each setting, we compare Graph-only Transfer (experience graph only), Copilot-only Transfer (copilot only), Graph+Copilot Transfer (full harness), and target-specific `ExpHarness` trained on the target executor. The comparisons below concern performance relative to these variants and the target-trained reference.

Joint Transfer Is Closest to Target-Specific Training in the Small-to-Large Setting. In Figure 2(a), Graph+Copilot Transfer is closest to target-specific `ExpHarness` across the displayed domains. It also outperforms the two partial-transfer variants, supporting joint reuse of the experience store and retrieval policy in this setting.

Large-to-Small Transfer Retains a Gap to Target-Specific Training. In Figure 2(b), the transferred variants remain below target-specific `ExpHarness`, while Graph+Copilot Transfer performs best among the transfer variants across most displayed domains. The panels use different axis ranges, so their polygon areas cannot quantify the relative difficulty of the transfer directions. Without a target No-Memory reference in this figure, these comparisons alone do not establish improvement over the unaugmented target executor.

Joint Transfer Also Performs Well for Reasoning-Oriented Targets. In Figure 2(c), Graph+Copilot Transfer approaches target-specific `ExpHarness`, including on ALFWorld and AppWorld. This observation is consistent with the reuse of experience across the evaluated model pairs. Because model identity and reasoning capability change together, the experiment does not isolate reasoning capability as the cause of transfer performance.

4.3 Ablation Studies of Four Experience-Retrieval Design Choices

To examine four experience-retrieval design choices, we conduct ablation studies across five evaluation domains: QA, Reasoning, Coding, ALFWorld, and AppWorld, with results reported in Figure 3.

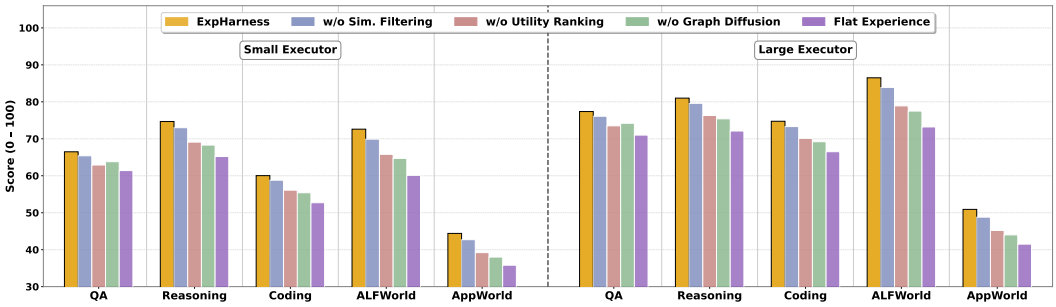


Figure 3: **Performance under four ablation settings.** We compare `ExpHarness` with variants modifying similarity filtering, memory organization, graph diffusion, and utility-aware ranking across QA, Reasoning, Coding, ALFWorld, and AppWorld.

We compare `ExpHarness` with four variants: i) **w/o Similarity Filtering** removes similarity-based filtering during experience insertion, allowing redundant experiences to be repeatedly retained; ii) **Flat Experience** replaces the experience graph with a flat experience pool and retrieves top-K experiences purely by semantic similarity; iii) **w/o Graph Diffusion** keeps the relational links but disables structure-aware expansion, retrieving only from the initial semantic seeds; and iv) **w/o Utility Ranking** ranks candidate experiences only by semantic similarity, without using historical utility statistics.

Across the reported domains, each of these four variants performs below the full method. **Flat Experience** shows the largest drop, but it changes both candidate generation and the ranking rule, so its gap cannot be attributed to relational structure alone. **w/o Graph Diffusion** and **w/o Utility Ranking** show lower scores under their respective retrieval changes, while **w/o Similarity Filtering** shows a smaller decline. These results support the combined retrieval design under the tested configurations. They do not separately isolate the effects of copilot training, the reward terms, or the node-utility update rule.

5 Additional Related Work

Reinforcement learning has become an important mechanism for improving LLMs and LLM agents, from PPO-based alignment (Ouyang et al., 2022; Liu et al., 2024) to DPO (Rafailov et al., 2023), process reward models (Lightman et al., 2024), self-play (Chen et al., 2024), self-correction (Kumar et al., 2024), and recent reasoning-oriented variants such as GRPO (Shao et al., 2024), Dr.GRPO (Liu et al., 2025), GSPO (Zheng et al., 2025), and Clip-Cov (Cui et al., 2025). Beyond directly optimizing LLM policies, experience-learning systems store and reuse prior interactions: ReAct (Yao et al., 2022) and Reflexion (Shinn et al., 2023) use reasoning traces and verbal feedback, while ExpeL (Zhao et al., 2024), ReasoningBank (Ouyang et al., 2026), LightMem (Fang et al., 2026), Mem0 (Chhikara et al., 2025), and AWM (Wang et al., 2024) construct external memories from past trajectories; IRCoT (Trivedi et al., 2023), Search-o1 (Li et al., 2025), S3 (Jiang et al., 2025b), and MemRL (Zhang et al., 2026) further introduce learned retrieval or episodic-memory policies. These methods differ in memory organization, retrieval control, and the use of downstream feedback. `ExpHarness` studies their combination in an external experience harness comprising an experience graph and a trainable retrieval copilot, jointly updated using outcome-based feedback around a frozen executor. Its transfer experiments further examine reuse of this harness across executor models.

6 Conclusion

We introduce ExpHarness, a learnable experience harness that improves frozen LLM executors through external experience reuse rather than parameter updates. To select experiences that are useful rather than merely similar, we organize them in a self-evolving experience graph and train a retrieval copilot with utility-grounded feedback, fostering a harness that adapts to each task and transfers across executors. Transfer experiments support reuse of the learned harness across the evaluated executor pairs, with joint graph and copilot transfer closest to target-specific training among the compared variants.

AI USE DISCLOSURE

We used generative AI tools solely to polish the writing of this paper. The authors reviewed and edited all AI-assisted text and take full responsibility for the content.

REFERENCES

Anthropic. System card: Claude Opus 4 & Claude Sonnet 4. Technical report, Anthropic, May 2025. URL https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, 2021.

Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 719–729, 2024.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Google DeepMind. Gemini 3.1 Flash-Lite model card. Technical report, Google DeepMind, March 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-flash-lite/.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, volume 2026, pp. 98706–98729, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024.

Pengrui Han, Rafal Kocielnik, Peiyang Song, Ramit Debnath, Dean Mobbs, Anima Anandkumar, and R. Michael Alvarez. The personality illusion: Revealing dissociation between self-reports & behavior in llms, 2025. URL https://arxiv.org/abs/2509.03730.

Pengrui Han, Xueqiang Xu, Keyang Xuan, Peiyang Song, Siru Ouyang, Runchu Tian, Yuqing Jiang, Cheng Qian, Pengcheng Jiang, Jiashuo Sun, et al. Steer2adapt: Dynamically composing steering vectors elicits efficient adaptation of llms. *arXiv preprint arXiv:2602.07276*, 2026.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021b.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. *Advances in Neural Information Processing Systems*, 37:10088–10116, 2024.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 7036–7050, 2024.

Pengcheng Jiang, Jiacheng Lin, Zhiyi Shi, Zifeng Wang, Luxi He, Yichen Wu, Ming Zhong, Peiyang Song, Qizheng Zhang, Heng Wang, et al. Adaptation of agentic ai. *arXiv preprint arXiv:2512.16301*, 2025a.

Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. s3: You don't need that much data to train a search agent via rl. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21599–21617, 2025b. doi: 10.18653/v1/2025.emnlp-main.1095.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Kevin Lu and Thinking Machines Lab. On-policy distillation. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*, 2023.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9802–9822, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2381–2391. Association for Computational Linguistics, 2018.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.

Jeongeon Park, Bryan Min, Kihoon Son, Jean Y Song, Xiaojuan Ma, and Juho Kim. Choicemates: Supporting unfamiliar online decision-making with multi-agent conversational interactions. In *Proceedings of the 31st International Conference on Intelligent User Interfaces*, pp. 1526–1550, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *International Conference on Learning Representations*, volume 2024, pp. 25055–25083, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International conference on machine learning*, pp. 31210–31227. PMLR, 2023.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4149–4158. Association for Computational Linguistics, 2019.

Xiangru Tang, Tianyu Hu, Muyang Ye, Yanjun Shao, Xunjian Yin, Siru Ouyang, Wangchunshu Zhou, Pan Lu, Zhuosheng Zhang, Yilun Zhao, et al. Chemagent: Self-updating library in large language models improves chemical reasoning. *arXiv preprint arXiv:2501.06590*, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

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

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL `https://arxiv.org/abs/2504.12516`.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

Xueqiang Xu, Jinfeng Xiao, James Barry, Mohab El-karef, Jiaru Zou, Pengcheng Jiang, Yunyi Zhang, Maxwell J Giammona, Geeth Mel, and Jiawei Han. Zero-shot open-schema entity structure discovery. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7547–7561, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models, 2026. URL https://arxiv.org/abs/2602.12275.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026.

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

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

A Limitations, Future Work, and Broader Impact

Limitations. Although `ExpHarness` provides a model-agnostic way to improve frozen LLM agents through graph-structured experience reuse, our study has several limitations. First, our experiments focus on a representative set of static reasoning benchmarks, including question answering (Clark et al., 2018; Talmor et al., 2019; Rein et al., 2024; Hendrycks et al., 2021a; Mihaylov et al., 2018), mathematical reasoning (Cobbe et al., 2021; Mirzadeh et al., 2025; Hendrycks et al., 2021b), code generation (Liu et al., 2023), and agentic environments (Shridhar et al., 2020; Trivedi et al., 2024). While these tasks cover diverse forms of experience reuse, future work could further evaluate `ExpHarness` on longer-horizon real-world applications, such as web browsing (He et al., 2024; Pan et al., 2024; Wei et al., 2025), scientific discovery (Jansen et al., 2024; Yuksekgonul et al., 2024; Tang et al., 2025; Xu et al., 2026), and collaborative multi-agent workflows (Jiang et al., 2025a; Park et al., 2026). Second, the experience graph is constructed using embedding-based semantic similarity with fixed hyperparameters, including neighbor size, similarity threshold, graph capacity, and utility update rate. Although these choices work well empirically, more adaptive graph construction and pruning strategies may further improve robustness across domains with different memory distributions.

Future Work. Several directions remain open for future investigation. First, the experience graph is currently constructed using embedding-based semantic similarity with fixed hyperparameters; replacing these with adaptive graph construction and pruning strategies could improve robustness across diverse domains. Second, the retrieval copilot is trained with scalar utility-grounded rewards from the downstream task outcomes, richer feedback signals, such as process-level credit from intermediate task milestones (Ma et al., 2023; Khalifa et al., 2025), could sharpen the learning signal further. Third, the agent currently leverages retrieved experiences through prompt-based conditioning. However, prompt-level instructions are known to translate inconsistently into the model's downstream behavior (Han et al., 2025; 2026; Turpin et al., 2023), leaving room for mechanisms that more tightly couple experience utilization with the policy itself, such as fine-tuning on experience-grounded trajectories (Lu & Lab, 2025), distilling experience-conditioned reasoning into the model's parameters (Ye et al., 2026; Zhao et al., 2026).

Broader Impact. `ExpHarness` enables LLM agents to accumulate and transfer task knowledge through an external graph-structured memory, without modifying model parameters. This design avoids executor parameter updates, although training the copilot and constructing memory incur additional costs. It also permits the use of closed-source executors through their input-output interfaces. Keeping their parameters frozen does not guarantee the safety of behavior conditioned on retrieved experience. We encourage future deployments of `ExpHarness` to be accompanied by robust evaluation protocols, human-in-the-loop checkpoints, and auditable retrieval logs.

B Implementation Details

We implement `ExpHarness` with two main components: a lightweight retrieval copilot π_{ret} and a self-evolving experience graph $G = (V, E)$. The executor LLM π_{exec} is kept frozen throughout all experiments, and only the retrieval copilot and graph memory are updated during training.

Retrieval copilot. The retrieval copilot π_{ret} is instantiated with Qwen2.5-3B-Instruct[1]. Given a task x, the copilot outputs two discrete control variables $(R, W) \sim \pi_{\text{ret}}(\cdot \mid x)$, where R controls the restart/locality of graph diffusion (larger R keeps retrieval closer to the semantic seeds) and W controls the trade-off between semantic relevance and historical utility. We rescale them as $\rho = R/100$ and $\lambda = W/100$. The copilot is optimized with Proximal Policy Optimization (PPO) (Schulman et al., 2017) using the utility-grounded reward in Eq. 14, with $\eta = 1$. We use the verl distributed RL framework with Fully Sharded Data Parallel (FSDP) for parameter and gradient offloading. The actor learning rate is 5×10^{-6} and the critic learning rate is 1×10^{-5}, both using AdamW (Loshchilov & Hutter, 2017) with cosine learning-rate decay and no warmup. The KL penalty coefficient is set to 0.01, and gradient checkpointing is enabled to reduce memory usage. The maximum prompt length is 4096 tokens and the maximum response length is 500 tokens. During training, we sample copilot outputs with temperature 1.0; during evaluation, we use greedy decoding. We train with a batch size of 16 for up to 500 gradient steps across 2 epochs. Checkpoints are saved every 20 steps and validation is performed every 30 steps.

Experience graph. The experience graph stores each experience as a node $v = (e_v, h_v, u_v, n_v)$, where e_v is the natural-language experience, h_v is its text embedding, u_v is its shared-reward estimate, and n_v is its retrieval count. We use Contriever[2] as the embedding model for both task inputs and experience nodes. When a new node is inserted into the graph, it is connected to its top-$K_{\text{nn}} = 5$ nearest neighbors under cosine similarity if the similarity is above the threshold $\theta = 0.3$, following Eq. 3. The graph capacity is capped at $|V|_{\max} = 2{,}000$ nodes. When the capacity is exceeded, we evict nodes with low utility and low retrieval frequency. For semantic seeding, we retrieve the top-$m = 10$ nodes by cosine similarity as the seed set V_0 in Eq. 5. After personalized PageRank expansion in Eq. 7, we construct the candidate set and apply utility-aware ranking using Eqs. 8–9. The final retrieved set contains the top-$K = 10$ experiences, as defined in Eq. 10. The UCB exploration coefficient is set to $c = 1.0$. For online graph updates, we update the retrieval count and utility estimate according to Eqs. 16–17, with exponential moving average rate $\beta = 0.1$.

Experience summarization. Each completed trajectory $\tau' = (x, \xi', y_{\text{with}}, s_{\text{with}})$ is summarized into a compact experience unit $e' = \text{Summarize}(\tau')$ following Eq. 2. The summarizer extracts reusable information rather than preserving the full trajectory. High-scoring trajectories are summarized into skills, such as successful reasoning patterns, planning strategies, or task-specific heuristics. Low-scoring trajectories are summarized into lessons, such as failure modes, invalid actions, or constraints to avoid. For ExpSuite-Static, we use Qwen2.5-3B-Instruct in zero-shot mode as the summarizer. For ExpSuite-Agentic, we use Gemini-3.1-Flash-Lite to generate environment-specific experience descriptions. Detailed prompts are provided in Appendix F.

Cold-start initialization. Before RL training, we initialize the experience graph with a cold-start procedure. We sample a subset of training tasks, execute them with the frozen executor π_{exec} without retrieved experiences, summarize the resulting trajectories, and insert the generated experience nodes

[1] `https://huggingface.co/Qwen/Qwen2.5-3B-Instruct`
[2] `https://huggingface.co/facebook/contriever`

into an initially empty graph. This initialization provides the copilot with a non-empty graph for early-stage retrieval and stabilizes training.

Executor LLMs. To evaluate whether `ExpHarness` is executor-agnostic, we use different frozen executor LLMs across the two ExpSuite settings. For EXPSUITE-STATIC, we use Llama-3.2-3B-Instruct as the small executor and Llama-3.1-8B-Instruct as the large executor, both accessed through the NVIDIA NIM API[3]. For EXPSUITE-AGENTIC, we use Qwen3-32B[4] as the small executor and Gemini-3.1-Flash-Lite as the large executor. Qwen3-32B is accessed through the NVIDIA NIM API, while Gemini-3.1-Flash-Lite is accessed through the Google Gemini API. In all cases, the executor is used only through its input-output interface and is never updated during training or evaluation.

Baseline memory construction. For fair comparison, all retrieval-centric experience learning baselines are provided with the same historical trajectories as `ExpHarness` whenever applicable. This includes ReasoningBank, ExpeL, LightMem, Mem0, AWM, and MemRL. Each baseline converts the trajectories into its own memory representation using the corresponding memory construction strategy and the prompts described in Appendix F. This controls the underlying trajectory source where applicable, but does not imply identical memory-item counts, context lengths, or compute budgets.

Training and evaluation. During training, each sampled task is evaluated twice: once with retrieved experiences and once without retrieved experiences. The two scores are used to compute the utility-grounded reward in Eq. 14, which simultaneously updates the retrieval copilot through PPO and updates graph utility statistics through exponential moving average. After each task, the completed trajectory is summarized into a new candidate experience, filtered for near-duplicates, and inserted into the graph. During evaluation, both the copilot and graph are fixed. The copilot outputs deterministic control variables via greedy decoding, retrieves the top-K experiences, and passes them to the frozen executor without any further graph or policy updates.

Infrastructure. All copilot training experiments are conducted on 4 NVIDIA A6000 GPUs with 48GB memory each using BF16 mixed precision. We use vLLM (Kwon et al., 2023) for efficient rollout generation with tensor parallelism size 1 and GPU memory utilization set to 0.3. The experience graph server runs on CPU and communicates with the training loop through HTTP. A single training run for EXPSUITE-AGENTIC on ALFWorld with 500 steps takes approximately 40 hours, while a training run for EXPSUITE-STATIC with 500 steps takes approximately 20 hours.

C TRAINING PROCEDURE OF EXPHARNESS

Algorithm 1 summarizes the full training procedure of `ExpHarness`. At each iteration, the copilot samples retrieval controls (R, W) for a task x and uses them to retrieve experiences from the current graph. The frozen executor is then evaluated with and without the retrieved experiences to compute the utility-grounded reward. This reward is used in two ways: it updates the retrieval copilot through PPO and updates the utility statistics of retrieved graph nodes through an exponential moving average. Finally, the completed trajectory is summarized into a new experience, filtered for near-duplicates, and inserted into the graph. At inference time, both the copilot and the graph are fixed; the copilot outputs deterministic control variables via greedy decoding, and the retrieved experiences are directly passed to the frozen executor without further updates.

D DATASET DESCRIPTIONS

We describe all evaluation datasets in ExpSuite below, categorized by their corresponding settings in Table 4. ExpSuite consists of two complementary groups: ExpSuite-Static and ExpSuite-Agentic. ExpSuite-Static evaluates experience reuse in single-turn input-output tasks, including question answering, mathematical reasoning, and code generation. ExpSuite-Agentic evaluates experience-guided decision-making in multi-step interactive environments. For all datasets, we report

[3] `https://build.nvidia.com`

[4] `https://huggingface.co/Qwen/Qwen3-32B`

Algorithm 1 Training Procedure of `ExpHarness`

Require: Task distribution $\mathcal{D}$, executor π_{exec}, initial copilot π_{ret}, initial graph $G = (V, E)$, reward weight η, utility update rate β

for each training iteration **do**
  Sample task $x \sim \mathcal{D}$
  // Retrieval
  Sample $(R, W) \sim \pi_{\text{ret}}(\cdot \mid x)$ (Eq. 4)
  Compute seed set V_0 via semantic seeding (Eq. 5)
  Expand candidate set C via personalized PageRank (Eq. 7)
  Retrieve $E_{R,W}(x)$ via utility-aware ranking (Eqs. 8–10)
  // Execution
  $s_{\text{with}} \leftarrow S\big(x,\, \pi_{\text{exec}}(x,\, E_{R,W}(x))\big)$ (Eq. 12)
  $s_{\text{without}} \leftarrow S\big(x,\, \pi_{\text{exec}}(x,\, \emptyset)\big)$ (Eq. 13)
  // Reward
  $r \leftarrow (s_{\text{with}} - s_{\text{without}}) + \eta \cdot s_{\text{with}}$ (Eq. 14)
  // Copilot update
  Update π_{ret} with PPO using reward r (Eq. 15)
  // Graph update
  for each retrieved node $v \in E_{R,W}(x)$ **do**
    $n_v \leftarrow n_v + 1$ (Eq. 16)
    $u_v \leftarrow (1 - \beta)\, u_v + \beta\, r$ (Eq. 17)
  end for
  // Graph growth
  Summarize $\tau' = (x, \xi', y_{\text{with}}, s_{\text{with}})$ into experience e' (Eq. 2)
  Filter near-duplicates; evict low-utility nodes if capacity exceeded
  Insert e' into G (Eq. 3)
end for

Ensure: Trained copilot π_{ret}, evolved experience graph G

task-specific performance metrics, and for interactive tasks we additionally report the number of environment steps to characterize environment interaction length.

Table 4: **Detailed summary of datasets used in ExpSuite.** We categorize datasets by setting, task type, and evaluation metric.

Dataset	Task	Metric
ExpSuite-Static		
ARC-C	Question Answering	Accuracy
CommonsenseQA	Question Answering	Accuracy
GPQA-Diamond	Question Answering	Accuracy
MMLU	Question Answering	Accuracy
OBQA	Question Answering	Accuracy
GSM8K	Mathematical Reasoning	Exact Match
GSM-Symbolic	Mathematical Reasoning	Exact Match
MATH	Mathematical Reasoning	Exact Match
HumanEval+	Code Generation	Pass@1
MBPP+	Code Generation	Pass@1
ExpSuite-Agentic		
ALFWorld-Seen	Seen Interactive Task Completion	SR / #Steps
ALFWorld-Unseen	Unseen Interactive Task Completion	SR / #Steps
AppWorld-Test-N	Normal App-based Workflow Execution	PR / #Steps
AppWorld-Test-C	Compositional App-based Workflow Execution	PR / #Steps

D.1 EXPSUITE-STATIC

D.1.1 QUESTION ANSWERING

ARC (AI2 Reasoning Challenge) (Clark et al., 2018) contains 7.8K science questions drawn from grade-school standardized exams. Following common practice, we adopt the Challenge split (ARC-C), which retains only questions that cannot be solved by simple retrieval or lexical co-occurrence baselines and therefore demand genuine reasoning.

CommonsenseQA (Talmor et al., 2019) consists of 12.2K multiple-choice questions built on top of ConceptNet, where answering correctly requires drawing on implicit, everyday world knowledge about concepts and their relations rather than facts stated in the question itself.

GPQA-Diamond (Rein et al., 2024) is the 198-question high-quality subset of GPQA (Graduate-Level Google-Proof QA), covering biology, physics, and chemistry. It retains questions answered correctly by both expert validators and incorrectly by a majority of non-expert validators, probing subject-matter expertise rather than surface lookup.

MMLU (Massive Multitask Language Understanding) (Hendrycks et al., 2021a) covers 57 subjects across STEM, the humanities, and the social sciences. Its multiple-choice questions span difficulties from elementary school to professional level, jointly testing factual recall and reasoning.

OpenBookQA (Mihaylov et al., 2018) contains 5.9K elementary-school science questions modeled after open-book exams. Solving each item requires combining a given scientific fact with additional commonsense knowledge, which probes multi-hop reasoning over scientific concepts.

D.1.2 MATHEMATICAL REASONING

GSM8K (Cobbe et al., 2021) is a collection of 8.5K grade-school math word problems, each solvable in 2–8 sequential arithmetic steps over the four basic operations. We evaluate by exact match on the final numeric answer.

GSM-Symbolic (Mirzadeh et al., 2025) is built from symbolic templates derived from GSM8K test questions. Each template specifies which names and numerical values can vary, along with constraints that keep the problem valid. Sampling different instantiations yields many surface variants of the same underlying problem, all still solved as ordinary numerical word problems. This tests whether a model's arithmetic reasoning holds up under changes to names and numbers, rather than depending on memorized surface forms. We evaluate by exact match on the final numeric answer.

MATH (Hendrycks et al., 2021b) consists of 12.5K competition-level problems spanning algebra, geometry, number theory, probability, counting, and precalculus. Drawn from contests such as AMC and AIME, the problems require both multi-step reasoning and substantial mathematical background.

D.1.3 CODE GENERATION

HumanEval+ is an extension of HumanEval (Liu et al., 2023) with a substantially expanded test suite that reduces false positives. It comprises 164 Python programming problems specified by a function signature and docstring, where models must produce code that passes all hidden tests.

MBPP+ (Liu et al., 2023) strengthens the Mostly Basic Python Problems benchmark with more rigorous tests, covering 974 crowd-sourced entry-level Python tasks that target standard algorithmic patterns and basic programming proficiency.

D.2 EXPSUITE-AGENTIC

D.2.1 ALFWORLD

ALFWorld-Seen (Shridhar et al., 2020) is a text-based interactive household benchmark in which an agent reads natural-language observations and issues action commands step by step to complete household goals. The **Seen** evaluation split (`valid_seen`) comprises 140 episodes spanning six task types (e.g., *pick-and-place*, *pick-heat-then-place*, *look-at-in-light*); room layouts overlap with training scenarios, so the agent encounters familiar spatial configurations but with novel task instances, isolating generalization at the instruction level.

ALFWorld-Unseen (Shridhar et al., 2020) shares the same six task types but evaluates on `valid_unseen`: 134 episodes set in entirely new room layouts with novel object placements, providing a stricter out-of-distribution test of the agent's ability to generalize beyond environments seen during training. For both splits we report task success rate (SR) and average environment steps per episode (#Steps).

D.2.2 APPWORLD

AppWorld-Test-N (Trivedi et al., 2024) is a controlled benchmark in which an agent executes day-to-day workflows across nine simulated applications (e.g., Gmail, Spotify, Amazon, Todoist, Venmo) by writing Python code that calls their APIs. The **Normal** test split (`test_normal`) consists of 56 scenarios (168 tasks) representing routine single-app or straightforward cross-app operations.

AppWorld-Test-C (Trivedi et al., 2024) evaluates agents on the **Compositional** split (`test_challenge`): 139 scenarios (417 tasks) that require coordinating complex multi-app sequences, resolving cross-service dependencies, and handling compositional reasoning across 457 APIs—forming a substantially harder probe of planning and tool-use capabilities. We report pass rate (PR) and mean environment steps (#Steps) for both splits.

E DATASET STATISTICS

In this section, we present detailed statistics for each dataset. Specifically, the statistics for ExpSuite-Static and ExpSuite-Agentic are provided in Table 5 and Table 6, respectively. For each ExpSuite-Static dataset, we randomly sample 1,500 instances and divide them into training, validation, and test sets following a 5:2:3 ratio; for datasets whose original size is smaller than 1,500 (GPQA-Diamond, HumanEval+, and MBPP+), we apply the same ratio over all available instances. For ExpSuite-Agentic, we adopt the official splits released by each benchmark: ALFWorld provides separate *Seen* and *Unseen* test environments, while AppWorld shares a single training set across two test variants, *Test-N* (normal) and *Test-C* (challenge).

Table 5: **ExpSuite-Static Data Statistics.** HEval+ denotes HumanEval+.

Split	Math			QA					Code	
	GSM8K	GSM-sym	MATH	MMLU	CSQA	OBQA	ARC-C	GPQA-Diamond	HEval+	MBPP+
Train	750	750	750	750	750	750	750	99	65	132
Valid	300	300	300	300	300	300	300	39	26	52
Test	450	450	450	450	450	450	450	60	39	80

Table 6: **ExpSuite-Agentic Data Statistics.**

Split	ALFWorld		AppWorld	
	Seen	Unseen	Test-N	Test-C
Train	3,553	3,553	90	90
Test	140	134	168	417

F PROMPT USAGE

This section separates retrieval-control prompts, experience-construction prompts, and executor prompts for `ExpHarness` and the documented baselines. In `ExpHarness`, the retrieval copilot outputs (R, W), the summarizer converts completed trajectories into experience, and the frozen executor produces answers, code, or environment actions. In ExpSuite-Static, this executor is Llama-3.2-3B-Instruct or Llama-3.1-8B-Instruct; the Qwen retrieval copilot does not answer the task. The role-separated `ExpHarness` templates below specify these interfaces rather than byte-for-byte API messages; model-specific chat serialization is distinct from the textual output tags.

F.1 COPILOT PROMPTING

The retrieval copilot π_{ret} receives a task description and outputs two control parameters (R, W) that govern how experiences are retrieved from the experience graph (see §3.3). We design task-category-specific prompt templates that share a common structure but differ in role descriptions and task framing. All templates instruct the copilot to output its parameters in the format `<search>R:W</search>`, optionally preceded by brief reasoning in `<think>...</think>` tags. Below we describe the three prompt categories and their design rationale.

ExpSuite-Agentic: ALFWorld. For ALFWorld, the copilot is framed as a *retrieval strategy controller for an embodied household agent*. The prompt presents the task description (e.g., `pick_and_place-Mug-None-Cabinet-312`) and explains how R and W control graph diffusion scope and utility weighting, respectively. Because ALFWorld tasks fall into a small number of recurring categories (e.g., pick-and-place, heat-then-place, examine-in-light), the copilot can learn to associate task types with effective retrieval strategies—for instance, using high W for familiar task types where historical utility is reliable, and low R for novel configurations that benefit from broader exploration. The controller template is shown in Table 7.

ExpSuite-Agentic: AppWorld. For AppWorld, the copilot is framed as a *retrieval strategy controller for a coding agent in an app-based environment*. The prompt structure is identical to ALFWorld, with the task description replaced by a natural-language instruction (e.g., "What is the title of the most-liked song in my Spotify playlists"). AppWorld tasks are more diverse than ALFWorld in terms of required APIs and multi-app dependencies, so the copilot must learn to vary its retrieval strategy across a wider range of task structures. The controller template is shown in Table 8.

ExpSuite-Static: QA, Math, and Code. The copilot acts as a retrieval strategy controller for all three categories. It receives the task specification and returns retrieval controls in `<search>R:W</search>`; it does not return an answer letter, numerical solution, or Python implementation. The retrieved experience text and original task are passed to a separate frozen Llama executor. Controller templates are shown in Tables 9, 10, and 11; executor templates are given in Tables 12–14. These controller templates describe the (R, W) output interface in Eq. 4, rather than an answer-generating search dialogue.

F.2 EXPERIENCE CONSTRUCTION

Each baseline that maintains an experience bank processes training rollout trajectories via LLM calls to produce structured memory items, but the construction strategy differs substantially across methods. For ExpSuite-Static, the method-specific construction templates are described below. Any `answer`, `expected_answer`, or response field in this subsection belongs to a completed training trajectory or its feedback; it is an input to experience construction, not an instruction for the retrieval copilot to answer a new task. For ExpSuite-Agentic, each method additionally uses environment-specific prompts tailored to ALFWorld household manipulation and AppWorld API-based workflows, described alongside the static prompts for each method below.

ReasoningBank uses a unified extractor that processes each trajectory into a structured memory item. For ExpSuite-Static, successful trajectories are processed with the prompt in Table 15; failed trajectories use Table 16. For ExpSuite-Agentic, ALFWorld trajectories use Table 17, and AppWorld trajectories use Table 18.

ExpeL (Zhao et al., 2024) maintains a global rule set updated via two critique passes. A *compare-critique* pass pairs each success with failures from the same dataset and asks the LLM to emit ADD/EDIT/REMOVE/AGREE operations over existing rules (Table 19). An *all-success* pass fires every $S = 8$ successes per dataset and similarly refines rules from positive evidence alone (Table 20). In addition, every rollout is stored verbatim as an exemplar memory item (with `title="exemplar"`), and finalized rules are stored as items with `title="rule"`. For ExpSuite-Agentic, ExpeL uses environment-specific critique prompts for ALFWorld (Table 21) and AppWorld (Table 22).

LightMem (Fang et al., 2026) implements a three-stage STM→LTM pipeline. A short-term memory buffer (size $K = 5$) is periodically flushed to long-term memory via a MemoryExtractor LLM call using Table 15 and Table 16. A separate offline consolidation pass then reconciles similar LTM

Table 7: **Copilot prompt template for ALFWorld** (ExpSuite-Agentic). The copilot outputs retrieval parameters (R, W) for graph-based experience retrieval. `{task_description}` is replaced with the ALFWorld task identifier.

<|im_start|>
user

You are a retrieval strategy controller for an embodied agent in the ALFRED household environment.

Task: **{task_description}**

You control how experiences are retrieved from a knowledge graph by setting two parameters:

R (0-100): Restart/locality control for graph diffusion.
Low R (0-30): Explore widely through connected experiences in the graph (broad discovery).
High R (70-100): Stay close to the most similar experiences (precise, local search).

W (0-100): Selection preference between semantic similarity and historical utility.
Low W (0-30): Prefer candidates most similar to this task.
High W (70-100): Give greater weight to historical shared-reward estimates.

When W=0, candidates are ranked only by cosine similarity within the graph-derived candidate set. Increase W to incorporate historical feedback.

Output format: <search>R:W</search>
You may reason briefly in <think>...</think> tags first.

Examples:
- New/unfamiliar task type → <search>20:10</search>
- Common task with good past data → <search>80:70</search>
- Moderate confidence → <search>50:40</search>

Output the complete retrieval-control tag. Do not output an environment action or a task answer.
<|im_end|>

Table 8: **Copilot prompt template for AppWorld** (ExpSuite-Agentic). Structure is identical to ALFWorld; only the role description and task content differ.

<|im_start|>
user

You are a retrieval strategy controller for a coding agent in the AppWorld environment.

Task: **{task_description}**

You control how experiences are retrieved from a knowledge graph by setting two parameters:

R (0-100): Restart/locality control for graph diffusion.
Low R (0-30): Explore widely through connected experiences in the graph (broad discovery).
High R (70-100): Stay close to the most similar experiences (precise, local search).

W (0-100): Selection preference between semantic similarity and historical utility.
Low W (0-30): Prefer candidates most similar to this task.
High W (70-100): Give greater weight to historical shared-reward estimates.

When W=0, candidates are ranked only by cosine similarity within the graph-derived candidate set. Increase W to incorporate historical feedback.

Output format: <search>R:W</search>
You may reason briefly in <think>...</think> tags first.

Examples:
- New/unfamiliar task type → <search>20:10</search>
- Common task with good past data → <search>80:70</search>
- Moderate confidence → <search>50:40</search>

Output the complete retrieval-control tag. Do not output an environment action or a task answer.
<|im_end|>

entries using the UPDATE/DELETE/IGNORE prompt in Table 23. For ExpSuite-Agentic, LightMem uses environment-specific extraction prompts for ALFWorld (Table 24) and AppWorld (Table 25).

Table 9: **Retrieval-copilot template for Question Answering** (ExpSuite-Static). Qwen outputs retrieval controls; the separate frozen Llama executor generates the task response.

System role: You are a retrieval strategy controller for a frozen task-solving model.

User input:
Question: **{question}**
{choices}
R (0–100): Restart/locality control. Low R allows broader graph diffusion; high R keeps retrieval closer to the semantic seeds.

W (0–100): Similarity–utility trade-off. Low W favors semantic similarity; high W gives more weight to historical utility.

When W=0, ranking uses only similarity within the graph-derived candidate set.

You may explain your retrieval choice briefly in `<think>...</think>` tags.
Output the retrieval parameters as `<search>R:W</search>`.
Do not solve the task or output its final answer or implementation.

Table 10: **Retrieval-copilot template for Mathematical Reasoning** (ExpSuite-Static). Qwen outputs retrieval controls; the separate frozen Llama executor generates the task response.

System role: You are a retrieval strategy controller for a frozen task-solving model.

User input:
Problem: **{question}**
R (0–100): Restart/locality control. Low R allows broader graph diffusion; high R keeps retrieval closer to the semantic seeds.

W (0–100): Similarity–utility trade-off. Low W favors semantic similarity; high W gives more weight to historical utility.

When W=0, ranking uses only similarity within the graph-derived candidate set.

You may explain your retrieval choice briefly in `<think>...</think>` tags.
Output the retrieval parameters as `<search>R:W</search>`.
Do not solve the task or output its final answer or implementation.

Mem0 (Chhikara et al., 2025) runs a two-step write pipeline per rollout. Step 1 extracts salient facts from the (problem, response) pair via Table 26. Step 2 retrieves the top-K most similar existing memories, then asks the LLM to emit ADD/UPDATE/DELETE/NONE operations via Table 27. For successful rollouts, a third reflect step generates a procedural summary via Table 28. For ExpSuite-Agentic, Mem0 uses environment-specific prompts for ALFWorld (Table 29) and AppWorld (Table 30).

AWM (Wang et al., 2024) induces a per-dataset workflow from a pool of up to $N = 5$ successful trajectories. The induction prompt (Table 31) asks the LLM to extract repetitive reasoning patterns as step-by-step workflows with `{placeholder}` names; the resulting workflow string is stored as a plain key–value entry indexed by dataset name and prepended verbatim at inference time without retrieval. For ExpSuite-Agentic, AWM uses environment-specific induction prompts for ALFWorld (Table 32) and AppWorld (Table 33).

`ExpHarness` uses a summarizer to extract experience text from completed executor trajectories; the graph component embeds the resulting entries and constructs their links as described in §3.2. For ExpSuite-Static, the construction prompt is given in Table 34 and Table 35. For ExpSuite-Agentic, ALFWorld trajectories use Table 36 and Table 37; AppWorld trajectories use Table 38.

F.3 Executor and Baseline Answering Prompts

At inference time, task-solving models receive the original task and any method-specific experience context. Answer tags, Python code blocks, and `complete_task(answer=...)` calls in this subsection are outputs of the task-solving model, not outputs of the `ExpHarness` retrieval copilot. Baseline search protocols and output formats are kept distinct from `ExpHarness`'s control interface.

Table 11: **Retrieval-copilot template for Code Generation** (ExpSuite-Static). Qwen outputs retrieval controls; the separate frozen Llama executor generates the task response.

System role: You are a retrieval strategy controller for a frozen task-solving model.

User input:
Task: **{task}**
Function signature: **{function_signature}**
R (0–100): Restart/locality control. Low R allows broader graph diffusion; high R keeps retrieval closer to the semantic seeds.

W (0–100): Similarity–utility trade-off. Low W favors semantic similarity; high W gives more weight to historical utility.

When W=0, ranking uses only similarity within the graph-derived candidate set.

You may explain your retrieval choice briefly in `<think>...</think>` tags.
Output the retrieval parameters as `<search>R:W</search>`.
Do not solve the task or output its final answer or implementation.

`ExpHarness`: frozen Llama executor. For ExpSuite-Static, the harness passes the original task and retrieved experience text to Llama-3.2-3B-Instruct or Llama-3.1-8B-Instruct. These frozen executors perform reasoning and generate the final answer or code. The role-separated answering templates are shown in Tables 12–14; the controller templates in §F.1 contain no answer-generation instructions. A separate summarizer processes completed training trajectories into experience; neither it nor the retrieval copilot replaces the executor.

Table 12: **Frozen Llama executor template for Question Answering** (ExpSuite-Static). The executor receives retrieved experience and generates the task response; it does not output retrieval controls.

System role: You are a helpful task-solving assistant.

User input:
Retrieved experience:
{retrieved_experiences}

Question: **{question}**
{choices}
Answer the multiple-choice question using the provided experience where relevant.
Analyze the options, then give the final answer letter inside `<answer>...</answer>`.
For example: `<answer> B </answer>`.

Table 13: **Frozen Llama executor template for Mathematical Reasoning** (ExpSuite-Static). The executor receives retrieved experience and generates the task response; it does not output retrieval controls.

System role: You are a helpful task-solving assistant.

User input:
Retrieved experience:
{retrieved_experiences}

Problem: **{question}**
Solve the problem using the provided experience where relevant.
Show your reasoning and put the final mathematical answer inside `<answer>...</answer>`.

No Memory. Chain-of-thought uses no retrieved experience; the model solves tasks directly with chain-of-thought instructions (Table 39–43).

Experience-conditioned baseline executors. The documented implementations of ReasoningBank, ExpeL, LightMem, Mem0, AWM, and IRCoT use the following base answering templates (Table 44–48), differing only in the content of **{retrieved_memories}**:

Table 14: **Frozen Llama executor template for Code Generation** (ExpSuite-Static). The executor receives retrieved experience and generates the task response; it does not output retrieval controls.

System role: You are a helpful task-solving assistant.

User input:
Retrieved experience:
{retrieved_experiences}

Task: **{task}**
Function signature: **{function_signature}**
Solve the coding task using the provided experience where relevant.
Your code should pass the supplied tests:
{test_cases}
Think through your approach, then put the complete Python function inside `<answer>...</answer>`.

- **ReasoningBank**: top-K memory items retrieved by dense embedding similarity, formatted as `## Retrieved Memory: ...`
- **ExpeL**: distilled rules (up to 10) prepended as a numbered list, plus top-10 exemplar trajectories retrieved by KNN.
- **LightMem**: LTM items after STM→LTM periodic consolidation, retrieved by dense embedding.
- **Mem0**: top-10 procedural summaries maintained via ADD/UPDATE/DELETE/NONE operations on prior (problem, response) pairs.
- **AWM**: full per-dataset-type workflow block prepended without retrieval (O(1) dict lookup).
- **IRCoT**: BM25-retrieved raw rollout pairs formatted as `Example:\nQuestion:...\nSolution:...`, accumulated across up to 3 iterative rounds.

Search-o1. Search-o1 instructs the model to issue explicit memory search queries via `<|begin_search_query|>` tags. Retrieved items are first summarized by a separate LLM call (Table 49) before being injected as search results. Inference prompts follow the same structure (Table 50–54).

Table 15: **Prompt for extracting a memory item from a successful trajectory (ReasoningBank; also used by LightMem's MemoryExtractor) in ExpSuite-Static.**

<|im_start|>
system
You will be given a user query and the corresponding trajectory that represents how an agent successfully accomplished the task.

Guidelines
You need to extract and summarize useful insights in the format of memory items based on the agent's successful trajectory. The goal of summarized memory items is to be helpful and generalizable for future similar tasks.

Important Notes
- You must first think why the trajectory is successful, and then summarize the insights.
- You can extract only 1 memory item from the trajectory.
- You must not repeat similar or overlapping items.
- Do not mention specific values or string contents, but rather focus on generalizable insights.

Output Format
Your output must strictly follow the Markdown format shown below:
Memory Item 1
Title: <the title of the memory item>
Description: <one-sentence summary of the memory item>
Content: <1–3 sentences describing the insights learned>
<|im_end|>
<|im_start|>
user
Query: **{question}**
Trajectory: **{reasoning_trajectory}**
<|im_end|>

Table 16: **Prompt for extracting a memory item from a failed trajectory (ReasoningBank; also used by LightMem's MemoryExtractor) in ExpSuite-Static.**

<|im_start|>
system
You will be given a user query and the corresponding trajectory that represents how an agent attempted the task but failed.

Guidelines
You need to extract and summarize useful insights based on the agent's failed trajectory. The goal of summarized memory items is to be helpful and generalizable for future similar tasks.

Important Notes
- You must first reflect on why the trajectory failed, then summarize lessons learned to prevent similar failures.
- You can extract only 1 memory item from the trajectory.
- You must not repeat similar or overlapping items.
- Focus on generalizable insights, not specific values.

Output Format
Your output must strictly follow the Markdown format shown below:
Memory Item 1
Title: <the title of the memory item>
Description: <one-sentence summary of the memory item>
Content: <1–3 sentences describing the lessons learned>
<|im_end|>
<|im_start|>
user
Query: **{question}**
Expected Answer: **{expected_answer}**
Trajectory: **{reasoning_trajectory}**
<|im_end|>

Table 17: **Prompt for ReasoningBank extracting a memory item in ALFWorld.**

You will be given a finished household trajectory (task goal, action–observation log) from a text-based ALFWorld environment, plus its outcome (success or failure). Distill the trajectory into a compact *reasoning rationale* that a future agent can reuse on a structurally similar task.

Output exactly one Markdown block of the form:
Memory Item
Title: **{short reusable name}**
Description: **{1–2 sentence task pattern}**
Content: **{step-by-step heuristic; cite key receptacles and verbs as schema variables, e.g. "go to {recep}" / "cool {X} with fridge 1"}**

Trajectory:
{trajectory_text}

Outcome: **{success** | **failure}**

Table 18: **Prompt for ReasoningBank extracting a memory item in AppWorld.**

You are reading a finished AppWorld task trajectory: an instruction, a sequence of `python` code calls against the AppWorld API, and a final `complete_task(answer=...)` call (or failure). Distill this trajectory into a compact reusable rationale a future agent can retrieve when it sees a similar task.

Output exactly one Markdown block of the form:
Memory Item
Title: **{short reusable name}**
Description: **{which app(s); what intent}**
Content: **{step-by-step plan, naming the key APIs and parameter shapes; substitute concrete IDs/emails with {placeholder} variables}**

Instruction: **{instruction}**
Trajectory: **{code log}**
Outcome: **{success** | **failure}**

Table 19: **Prompt for ExpeL compare-critique pass (success vs. failure pair) in ExpSuite-Static.** The LLM emits ADD/EDIT/REMOVE/AGREE operations over the current rule set. **{existing_rules}** is replaced by the numbered rule list; **{task}**, **{success_history}**, and **{fail_history}** are filled from the rollout pair.

system
You are an advanced reasoning agent that can add, edit or remove rules from your existing rule set, based on forming new critiques of past task trajectories. You will be given two previous reasoning task trials: one successful and one unsuccessful. You need to analyze what made the successful trial succeed and the unsuccessful trial fail.

user
Here are the two previous trials to compare and critique:

TRIAL TASK: **{task}**

SUCCESSFUL TRIAL: **{success_history}**

FAILED TRIAL: **{fail_history}**

Here are the EXISTING RULES: **{existing_rules}**

By examining and contrasting to the successful trial, and the list of existing rules, you can perform the following operations: add, edit, remove, or agree so that the new list of rules is GENERAL and HIGH LEVEL critiques of the failed trial or proposed way of Thought so they can be used to avoid similar failures when encountered with different questions in the future.

The available operations are: AGREE (if the existing rule is strongly relevant), REMOVE (if a rule is contradictory or duplicated), EDIT (if a rule can be enhanced), ADD (if a new, distinct rule is needed). Format:

```
AGREE <N>:  <rule>
REMOVE <N>:  <rule>
EDIT <N>:  <new rule>
ADD <N>:  <new rule>
```

Do not mention the trials in the rules — all rules should be GENERALLY APPLICABLE. Each rule should be concise. Do at most 4 operations; each existing rule can receive at most 1 operation.
<|im_end|>

Table 20: **Prompt for ExpeL all-success critique pass in ExpSuite-Static** (fired every $S = 8$ successes per dataset). **{success_history}** contains the concatenated successful trajectories; **{existing_rules}** is the current numbered rule list.

system
You are an advanced reasoning agent that can add, edit or remove rules from your existing rule set, based on forming new critiques of past task trajectories. You will be given successful reasoning task trials. Analyze them to extract general reasoning insights.

user
Here are the trials: **{success_history}**

Here are the EXISTING RULES: **{existing_rules}**

By examining the successful trials, and the list of existing rules, you can perform the following operations: add, edit, remove, or agree so that the new list of rules are general and high level insights of the successful trials or proposed way of Thought so they can be used as helpful tips to different tasks in the future.

The available operations follow the same format as the compare-critique pass (AGREE / REMOVE / EDIT / ADD). Focus on REMOVE rules first, and only ADD a rule if it is very insightful and distinct from existing rules. Do not mention the trials in the rules.
<|im_end|>

Table 21: **Prompt for ExpeL extracting a memory item in ALFWorld.**

You are an advanced reasoning agent that improves by reflecting on past ALFWorld trials. You are given two trials on similar household tasks: one successful and one unsuccessful. Compare them and decide which existing rules in your rule set should be *ADDED*, *EDITED*, or *REMOVED*, or which can stay (*AGREE*). Each rule is a one-sentence heuristic for a structural sub-problem (e.g. "before placing two of the same item, locate both before relocating").

Output one operation per line, in the form:
ADD <*rule text*>
EDIT <*rule_id*> → <*new text*>
REMOVE <*rule_id*>
AGREE <*rule_id*>

Existing rules: **{rules}**
Successful trial: **{traj_succ}**
Failed trial: **{traj_fail}**

Table 22: **Prompt for ExpeL extracting a memory item in AppWorld.**

You are an advanced reasoning agent that improves by reflecting on past AppWorld coding trials. You are given two finished trials on similar API-call tasks: one whose final `complete_task` call passed the test cases and one that did not. Compare them and emit ADD/EDIT/REMOVE/AGREE operations over the existing rule set. Each rule is a one-sentence heuristic about API usage (e.g. "always call `login` before any per-app endpoint").

Output one operation per line, same format as the ALFWorld variant (Table 21).

Existing rules: **{rules}**
Successful trial: **{code_log_succ}**
Failed trial: **{code_log_fail}**

Table 23: **Prompt for LightMem offline LTM consolidation in ExpSuite-Static.** For each target memory with at least one similar candidate (cosine similarity ≥ 0.8), the LLM decides whether to UPDATE, DELETE, or IGNORE the target. **{target_memory}** and **{candidate_memories}** are filled from the LTM.

system
You are a memory management assistant. Your task is to decide whether the target memory should be updated, deleted, or ignored based on the candidate source memories.

Decision rules:
1. **Update**: If the target and candidate memories describe the same fact but are not fully consistent (candidates provide more detail or refinements), update the target by integrating the additional information.
2. **Delete**: If the target and candidate memories contain a direct conflict, the candidates (more recent) take precedence. Delete the target.
3. **Ignore**: If target and candidates are unrelated, take no action.

Additional guidance: Use only the information provided. Do not invent details. Your operation always applies to the target memory; do not modify candidate memories.

The output must be a JSON object:
`{"action": "update" | "delete" | "ignore", "new_memory": {...}}`
(`new_memory` required only when `action = "update"`)

user
Target memory: **{target_memory}**
Candidate memories: **{candidate_memories}**

Table 24: **Prompt for LightMem extracting a memory item in ALFWorld.**

You are a Personal Information Extractor. The input is a buffer of recently observed ALFWorld turns, each tagged with a topic and a timestamp. Extract *all* salient facts about the agent's environment and behaviour: object locations, completed sub-goals, durable preferences, errors.

Format each fact as:
FACT: **{one-sentence statement} [{ts}] (source_id={seq})**

[. . . verbatim few-shot examples omitted; full prompt at `baselines/lightmem/memory.py:81...`]

Conversation buffer:
{buffer}

Table 25: **Prompt for LightMem extracting a memory item in AppWorld.**

You are a Personal Information Extractor. The input is a buffer of recently executed AppWorld turns: code chunks the agent ran, plus the JSON response from each `apis.*` call. Extract salient durable facts: app credentials/tokens, IDs of relevant entities (user, contact, note, playlist, transaction), API quirks, and successful subroutines.

Format each fact as:
FACT: **{one-sentence statement} [{ts}] (source_id={seq})**

[. . . verbatim few-shot examples omitted; full prompt at `baselines/lightmem/memory.py:81...`]

Conversation buffer:
{buffer}

Table 26: **Prompt for Mem0 Step 1: fact extraction from a (problem, response) pair in ExpSuite-Static.** The LLM returns a JSON object `{"facts": [...]}` listing salient procedural facts.

system
You are a Personal Information Organizer, specialized in accurately storing facts, user memories, and preferences. Your primary role is to extract relevant pieces of information from conversations and organize them into distinct, manageable facts.

Extract facts and preferences and return them in JSON format:
`{"facts": ["<fact 1>", "<fact 2>", ...]}`

Guidelines: store personal preferences, important details, plans, activity preferences, professional details, and other relevant information. Do not extract anything from system messages. Detect the user's language and record facts in the same language. If nothing relevant is found, return an empty list.

user
Problem: **{question}**
Solution: **{response}**

Table 27: **Prompt for Mem0 Step 2: operation decision (ADD / UPDATE / DELETE / NONE) in ExpSuite-Static. {old_memories}** contains the top-K retrieved existing memory entries as a JSON list; **{new_facts}** contains the facts extracted in Step 1.

user
You are a smart memory manager which controls the memory of a system. You can perform four operations: (1) add into the memory, (2) update the memory, (3) delete from the memory, and (4) no change.

Compare newly retrieved facts with the existing memory. For each new fact, decide whether to ADD (new information not present), UPDATE (same topic but different content; keep the more informative version), DELETE (direct contradiction; newer facts take precedence), or NONE (already present or irrelevant).

Current memory: **{old_memories}**

New retrieved facts: **{new_facts}**

Return your response strictly in the following JSON structure:
`{"memory": [{"id": "<id>", "text": "<content>", "event": "ADD|UPDATE|DELETE|NONE", "old_memory": "<old content>"}]}`
(`old_memory` required only when `event = "UPDATE"`)

Do not return anything except the JSON.

Table 28: **Prompt for Mem0 Step 3: procedural summary generation in ExpSuite-Static** (applied to successful rollouts only). The LLM produces a structured, step-by-step summary of the agent's execution history.

system
You are a memory summarization system that records and preserves the complete interaction history between a human and an AI agent. You are provided with the agent's execution history. Your task is to produce a comprehensive summary that contains every detail necessary for the agent to continue the task without ambiguity.

Structure your output as follows:
Overview: Task objective and progress status (completion percentage and milestones).

Sequential Agent Actions (numbered steps): For each step include (1) the precise agent action and all parameters, (2) the exact, unaltered action result verbatim, and (3) embedded metadata: key findings, navigation history, errors and challenges, and current context.

Guidelines: preserve every output verbatim; use chronological order; include exact URLs, element indexes, error messages, and JSON responses. Output only the structured summary with no additional commentary.

user
Problem: **{question}**
Solution: **{response}**
Outcome: SUCCESS

Table 29: **Prompt for Mem0 extracting a memory item in ALFWorld.**

You are a Personal Information Organizer specialised in storing facts about an agent inside a text-based ALFWorld environment. Read the recent action–observation log and extract all atomic facts of the form "{subject} {predicate} {object}". Examples include: object locations ("soapbottle 1 is in cabinet 4"), task progress ("user has cleaned the dishsponge"), and reusable verbs ("cool X requires opening fridge 1 first").

Today's date is **{today}**. Return a JSON object:
`{"facts": ["{fact 1}", "{fact 2}", ...]}`

[... verbatim few-shot examples omitted ...]

Conversation:
{turns}

Table 30: **Prompt for Mem0 extracting a memory item in AppWorld.**

You are a Personal Information Organizer specialised in storing facts about a user's apps and AppWorld task history. Read the recent code-and-response log and extract all atomic facts of the form "{subject} {predicate} {object}". Examples: tokens ("user has venmo access_token"), entities ("simple_note id 3084 is the songs note"), API patterns ("`apis.spotify.update_playlist` accepts `title` keyword").

Today's date is **{today}**. Return a JSON object:
`{"facts": ["{fact 1}", "{fact 2}", ...]}`

[. . . verbatim few-shot examples omitted . . .]

Conversation:
{turns}

Table 31: **Prompt for AWM workflow induction in ExpSuite-Static.** Applied once per dataset to a pool of up to $N = 5$ successful trajectories. **{examples}** is a block of `Problem: ... / Solution: ...` entries; the induced workflow is stored as a plain string and prepended verbatim at inference time.

user
Given a list of solved reasoning problems from the same category, your task is to extract the common workflows to solve these problems.

Each given problem contains a problem statement and a solution. You need to find the repetitive reasoning patterns across multiple problems, and extract each as a workflow.

Each workflow should be a commonly-reused reasoning strategy. Do not generate similar or overlapping workflows. Each workflow should have at least two steps. Represent problem-specific values with descriptive **{placeholder}** names.

Try to generate as many workflows as needed to cover all the problems in the input list.

`## Concrete Examples`
{examples}

`## Summary Workflows`

Table 32: **Prompt for AWM extracting a memory item in ALFWorld.**

You are an Agent Workflow Memory inducer. You are given up to N=5 successful ALFWorld trajectories that share the same task type (e.g. `pick_clean_then_place_in_recep`). Extract the repetitive reasoning pattern as a step-by-step *workflow* with concrete-value placeholders.

Output format:
```
Workflow for {task_type}:
Step 1: {instruction with {object} and {recep} placeholders}
Step 2: ...
Step k: {final placement step}
```

Each workflow should have at least two steps. Do not generate similar or overlapping workflows; represent task-specific values with descriptive `{placeholder}` names.

Successful trajectories:
{trajectories}

Table 33: **Prompt for AWM extracting a memory item in AppWorld.**

You are an Agent Workflow Memory inducer. You are given up to $N=5$ successful AppWorld trajectories that share the same task type. Extract the repetitive reasoning pattern as a step-by-step *workflow* with concrete-value placeholders.

Output format:

```
Workflow for {task_type}:
Step 1:  {API call template with {access_token}, {user_email},
{note_id} placeholders}
Step 2:  ...
Step k:  apis.supervisor.complete_task(answer={result})
```

Each workflow should have at least two steps. Substitute concrete IDs / emails / strings with `{placeholder}` names.

Successful trajectories:
{trajectories}

Table 34: **Summarizer prompt for `ExpHarness` extracting a memory item from a successful trajectory in ExpSuite-Static.**

<|im_start|>
system
You are an expert at distilling problem-solving experiences into concise, reusable lessons. Be brief and generalizable.
<|im_end|>
<|im_start|>
user
A search/QA question was answered correctly.

Question: **{question}**
Thought: **{thought}**
Executor answer: **{answer}**

Extract a reusable search skill (2–3 sentences):
- `planning_pattern`: Abstract the logic using generic terms `[Entity]`, `[Attribute]`, `[Time_Period]`
- What key strategy led to success?

Do NOT include specific names, numbers, or answers. Focus on the transferable strategy.

Output format: `SKILL: [your skill text]`
<|im_end|>

Table 35: **Summarizer prompt for `ExpHarness` extracting a memory item from a failed trajectory in ExpSuite-Static.**

<|im_start|>
system
You are an expert at distilling problem-solving experiences into concise, reusable lessons. Be brief and generalizable.
<|im_end|>
<|im_start|>
user
A search/QA question was answered incorrectly.

Question: **{question}**
Thought: **{thought}**
Incorrect executor answer: **{answer}**

Extract a reusable lesson (2–3 sentences):
- `trigger_condition`: What kind of question caused the error?
- `bad_action`: What went wrong?

Do NOT include specific names. Focus on the transferable lesson.

Output format: `SKILL: [your lesson text]`
<|im_end|>

Table 36: **Summarizer prompt for `ExpHarness` extracting a memory item from a successful ALFWorld trajectory.**

<|im_start|>
system
You are an expert at analyzing household robot trajectories. Extract specific, actionable lessons from the provided trajectory.
user
An ALFWorld household task was completed successfully.

Task: **{task}**
Full trajectory (action → observation): **{trajectory}**

Extract a reusable skill (3–5 sentences). Include:
1. The general task category (`pick_and_place`, `heat_then_place`, `clean_then_place`, `cool_then_place`, `examine_in_light`, `pick_two`)
2. The concrete step-by-step strategy that worked
3. Common locations where target objects are found (e.g. "soapbar is usually on countertop, bathtubbasin, or shelf")

Be specific. Use actual object/location types (`countertop`, `sinkbasin`, `microwave`).

Output format: `SKILL: [your skill text]`
<|im_end|>

Table 37: **Summarizer prompt for `ExpHarness` extracting a memory item from a failed ALFWorld trajectory.**

<|im_start|>
system
You are an expert at analyzing household robot trajectories. Extract specific, actionable lessons from the provided trajectory.
user
An ALFWorld household task failed after all steps.

Task: **{task}**
Trajectory: **{trajectory}**

Extract a reusable lesson (3–5 sentences). Include:
1. The general task category
2. What specific mistake was made
3. What the agent should have done differently

Output format: `SKILL: [your lesson text]`
<|im_end|>

Table 38: **Summarizer prompt for `ExpHarness` extracting a memory item from an AppWorld episode. {STATUS}** is replaced by `SUCCESSFUL`, `FAILED`, or `PARTIAL`.

<|im_start|>
user
Analyze this **{STATUS}** AppWorld code generation episode.

Task: **{task}**
Trajectory: **{trajectory}**

Extract ONE concise, actionable skill. Focus on SPECIFIC API details, not generic advice.

Format: `[Short Title]` Specific principle. When: trigger condition.

Good examples:
- `[Spotify Login Fields]` `apis.spotify.login()` requires `email` and `password` as keyword arguments...
- `[Venmo Transactions Return Dict]` `apis.venmo.list_transactions()` returns a dict with key `"transactions"`...
- `[File System Read Path]` Use `apis.file_system.read_file(file_path=...)` with absolute paths...

BAD examples (do NOT generate):
- "Always inspect API docs first" — too generic
- "Verify API method names" — too vague

Rules: Include SPECIFIC app names, API method names, parameter names, return value structures.

Output ONLY the skill line.
<|im_end|>

Table 39: **Executor prompt for No Memory in Question Answering.**

<|im_start|>
system
You are a helpful assistant skilled in mathematics, logical reasoning, and programming. Solve problems step by step, showing your work clearly.

CRITICAL FORMAT REQUIREMENT:
- You MUST end your response with your final answer wrapped in <answer> </answer> tags.
- ALWAYS use <answer> </answer> tags.
- Example: <answer> A </answer>
<|im_end|>
<|im_start|>
user
Answer this question:

Question: **{question}**

{choices}

Analyze each option and explain your reasoning.

Write your final answer as: <answer> [letter] </answer>
<|im_end|>

Table 40: **Executor prompt for No Memory in Math Reasoning.**

<|im_start|>
system
You are a helpful assistant skilled in mathematics, logical reasoning, and programming. Solve problems step by step, showing your work clearly.

CRITICAL FORMAT REQUIREMENT:
- You MUST end your response with #### followed by your final answer.
- NEVER use \boxed{} or any LaTeX boxing format.
- ALWAYS use #### format, even for mathematical expressions.
- Example: #### 42 or #### x^2 + 1
<|im_end|>
<|im_start|>
user
Solve this math word problem:

{question}

Think step by step, showing your calculations.

Write your final answer as: #### [number]
<|im_end|>

Table 41: **Executor prompt for No Memory in Code Generation.**

<|im_start|>
system
You are an expert Python programmer. Solve coding problems step by step.

CRITICAL FORMAT REQUIREMENT:
- Provide your final solution in a ```python code block.
- Example format: ```python \n def solution(): pass \n ```
<|im_end|>
<|im_start|>
user
Task: **{task}**

Your code should pass these tests:

{test_cases}

Think through your approach, then write the Python function.

REMEMBER: Provide your code in a ```python code block.
<|im_end|>

Table 42: **Executor prompt for No Memory in ALFWorld.**

[SYSTEM] You are controlling a text-based ALFWorld environment. Choose the NEXT action as ONE admissible command string. Output only the command, copied verbatim from the admissible list.

[USER] Task: **{objective}**

Interaction history so far: **{history}**

Current observation: **{current_obs}**

Admissible actions: **{admissible}**

Action:

Table 43: **Executor prompt for No Memory in AppWorld.**

[SYSTEM] You are an autonomous AppWorld agent. You complete each task by writing Python code that calls `apis.<app>.<endpoint>(...)` and finally `apis.supervisor.complete_task(answer=...)`. Wrap each turn's code in triple-backtick fences.

[USER] Instruction: **{instruction}**

Available apps and their docs: **{api_docs_summary}**

Recent execution history: **{history}**

Reason step by step, then write the next code block.

Table 44: **Executor prompt for experience-conditioned baselines (ReasoningBank, ExpeL, LightMem, Mem0, AWM, IRCoT) in Question Answering. {retrieved_memories}** is replaced by the method-specific memory context.

<|im_start|>
system
You are a helpful assistant skilled in mathematics, logical reasoning, and programming. Solve problems step by step, showing your work clearly.
CRITICAL FORMAT REQUIREMENT:
- You MUST end your response with your final answer wrapped in <answer> </answer> tags.
- ALWAYS use <answer> </answer> tags.
- Example: <answer> A </answer>
<|im_end|>
<|im_start|>
user
{retrieved_memories}
Answer this question:
Question: **{question}**
{choices}
Analyze each option and explain your reasoning.
Write your final answer as: <answer> [letter] </answer>
<|im_end|>

Table 45: **Executor prompt for experience-conditioned baselines in Math Reasoning.**

<|im_start|>
system
You are a helpful assistant skilled in mathematics, logical reasoning, and programming. Solve problems step by step, showing your work clearly.

CRITICAL FORMAT REQUIREMENT:
- You MUST end your response with #### followed by your final answer.
- NEVER use \boxed{} or any LaTeX boxing format.
- ALWAYS use #### format, even for mathematical expressions.
- Example: #### 42 or #### x^2 + 1
<|im_end|>
<|im_start|>
user
{retrieved_memories}

Solve this math word problem:

{question}

Think step by step, showing your calculations.

Write your final answer as: #### [number]
<|im_end|>

Table 46: **Executor prompt for experience-conditioned baselines in Code Generation.**

<|im_start|>
system
You are an expert Python programmer. Solve coding problems step by step.

CRITICAL FORMAT REQUIREMENT:
- Provide your final solution in a ```python code block.
- Example format: ```python \n def solution(): pass \n ```
<|im_end|>
<|im_start|>
user
{retrieved_memories}

Task: **{task}**

Your code should pass these tests:
{test_cases}

Think through your approach, then write the Python function.
REMEMBER: Provide your code in a ```python code block.
<|im_end|>

Table 47: **Executor prompt for experience-conditioned baselines in ALFWorld.**

[SYSTEM] You are controlling a text-based ALFWorld environment. Choose the NEXT action as ONE admissible command string. Output only the command, copied verbatim from the admissible list.

[USER] Task: **{objective}**

{retrieved_memories}

Interaction history so far: **{history}**

Current observation: **{current_obs}**

Admissible actions: **{admissible}**

Action:

Table 48: **Executor prompt for experience-conditioned baselines in AppWorld.**

[SYSTEM] You are an autonomous AppWorld agent. You complete each task by writing Python code that calls `apis.<app>.<endpoint>(...)` and finally `apis.supervisor.complete_task(answer=...)`. Wrap each turn's code in triple-backtick fences.

[USER] Instruction: **{instruction}**

{retrieved_memories}

Available apps and their docs: **{api_docs_summary}**

Recent execution history: **{history}**

Reason step by step, then write the next code block.

Table 49: **Prompt for summarizing retrieved memories in Search-o1.**

<|im_start|>
user
Task Instruction:

You are tasked with reading and analyzing memories based on the following inputs: **Previous Reasoning Steps**, **Current Search Query**, and **Searched Memories**. Your objective is to extract relevant and helpful information for the **Current Search Query** from the **Searched Memories** and seamlessly integrate this information into the **Previous Reasoning Steps** to continue reasoning for the original question.

Guidelines:
1. Carefully review the content of each searched memory. Identify factual information that is relevant to the Current Search Query.
2. Select the information that directly contributes to advancing the Previous Reasoning Steps. Ensure that the extracted information is accurate and relevant.

Output Format:
- If the memories provide helpful information: present the information beginning with `**Final Information**`
`**Final Information**`
[Helpful information]
- If the memories do not provide helpful information:
`**Final Information**`
No helpful information found.

Inputs:
- Previous Reasoning Steps: **{prev_reasoning}**
- Current Search Query: **{search_query}**
- Searched Memories: **{retrieved_memories}**
<|im_end|>

Table 50: **Task-solving prompt for Search-o1 in Question Answering.**

<|im_start|>
system
You are a helpful assistant skilled in mathematics, logical reasoning, and programming. You have the ability to perform memory searches to help you answer the user's question accurately.
To perform a search: write <|begin_search_query|> your query here <|end_search_query|>.
The system will then search and analyze relevant memories, returning helpful information in the format <|begin_search_result|> ...results... <|end_search_result|>.
Once you have all the information you need, continue your reasoning.

CRITICAL FORMAT REQUIREMENT:
- You MUST end your response with your final answer wrapped in <answer> </answer> tags.
- ALWAYS use <answer> </answer> tags.
- Example: <answer> A </answer>
<|im_end|>
<|im_start|>
user
Answer this question:

Question: **{question}**

{choices}

Analyze each option and explain your reasoning.

Write your final answer as: <answer> [letter] </answer>
<|im_end|>

Table 51: **Task-solving prompt for Search-o1 in Math Reasoning.**

<|im_start|>
system
You are a helpful assistant skilled in mathematics, logical reasoning, and programming. You have the ability to perform memory searches to help you answer the user's question accurately.
To perform a search: write <|begin_search_query|> your query here <|end_search_query|>.
The system will return helpful information in the format <|begin_search_result|> ...results... <|end_search_result|>.

CRITICAL FORMAT REQUIREMENT:
- You MUST end your response with #### followed by your final answer.
- NEVER use \boxed{} or any LaTeX boxing format.
- Example: #### 42
<|im_end|>
<|im_start|>
user
Solve this math word problem:

{question}

Think step by step, showing your calculations.

Write your final answer as: #### [number]
<|im_end|>

Table 52: **Task-solving prompt for Search-o1 in Code Generation.**

<|im_start|>
system
You are an expert Python programmer. You have the ability to perform memory searches to help you answer the user's question accurately.
To perform a search: write <|begin_search_query|> your query here <|end_search_query|>.
The system will return helpful information in the format <|begin_search_result|> ...results... <|end_search_result|>.

CRITICAL FORMAT REQUIREMENT:
- Provide your final solution in a ```python code block.
<|im_end|>
<|im_start|>
user
Task: **{task}**

Your code should pass these tests:
{test_cases}

Think through your approach, then write the Python function.
REMEMBER: Provide your code in a ```python code block.
<|im_end|>

Table 53: **Task-solving prompt for Search-o1 in ALFWorld.**

[SYSTEM] You are controlling a text-based ALFWorld environment. You may interleave reasoning with explicit memory queries. To query the experience corpus, emit: ⟨|begin_search_query|⟩{your sub-question}⟨|end_search_query|⟩. The system will return summarised search results inside ⟨|begin_search_result|⟩...⟨|end_search_result|⟩. After at most S=3 rounds, output the next admissible action.

[USER] Task: **{objective}**
Interaction history: **{history}**
Current observation: **{current_obs}**
Admissible actions: **{admissible}**

Reason and (optionally) search; then output Action: ⟨cmd⟩.

Table 54: **Task-solving prompt for Search-o1 in AppWorld.**

[SYSTEM] You are an autonomous AppWorld agent. You may interleave reasoning with explicit memory queries. To query the experience corpus, emit: ⟨|begin_search_query|⟩{your sub-question}⟨|end_search_query|⟩. The system will return summarised search results inside ⟨|begin_search_result|⟩...⟨|end_search_result|⟩. After at most S=3 rounds, output the next code block.

[USER] Instruction: **{instruction}**
Available apps and docs: **{api_docs_summary}**
Recent execution history: **{history}**

Reason and (optionally) search; then write the next code block in triple-backtick fences.

G Experimental Result Analysis

G.1 ExpHarness Outperforms General Prompt-based Baselines and Experience Learning Methods

We evaluate ExpHarness on ExpSuite, covering both single-turn static tasks and multi-step agentic environments. Results are reported in Table 2 and Table 3. We have the following observations.

ExpHarness Achieves the Highest Weighted Average Scores in the Evaluated Settings. On ExpSuite-Static, ExpHarness improves the test-instance-weighted average score over MemRL, the strongest baseline under this aggregation, by 12.1% with the small executor and 4.5% with the large executor. These are relative improvements, computed as $(S_{\text{ours}} - S_{\text{baseline}})/S_{\text{baseline}}$; they do not imply the best result on every individual benchmark. On ExpSuite-Agentic, the corresponding relative improvements over S3 are 21.4% and 12.7%. Average environment interaction steps are 12.7% and 21.6% lower than those of the baselines with the lowest average step counts, respectively. Step counts describe environment interaction and do not include all retrieval, LLM inference, or training costs; they can also vary with task success.

Relative Gains Vary Across Executors and Task Settings. In these evaluations, the relative gains are larger for the executor labeled small within each setting, and larger in ExpSuite-Agentic than in ExpSuite-Static. These comparisons are descriptive: executor identities, benchmarks, and baseline scores differ across settings. They are consistent with useful experience reuse in interactive tasks, but do not isolate model scale or task horizon as the cause of the differences.

G.2 Harness Transfer Across Different LLM Executors

We evaluate executor transfer under three settings. *Small-to-large transfer* uses the small executor as the source and the large executor as the target, testing whether experience components learned with cheaper models can benefit stronger frozen LLMs. *Large-to-small transfer* uses the large executor as the source and the small executor as the target, testing whether high-quality experiences learned from stronger models can still be exploited by weaker models. *Non-reasoning-to-reasoning transfer* uses Llama-3.1-8B-Instruct as the source and DeepSeek-R1-Distill-Llama-8B as the target for ExpSuite-Static, and Gemini-3.1-Flash-Lite as the source and Claude-Sonnet-4 as the target for ExpSuite-Agentic, matching §4.2. For each setting, we compare Graph-only Transfer, Copilot-only Transfer, Graph+Copilot Transfer, and target-specific ExpHarness.

Joint Transfer Is Closest to Target-Specific Training in the Small-to-Large Setting. In Figure 2(a), Graph+Copilot Transfer is closest to target-specific ExpHarness across the displayed domains. It also outperforms the two partial-transfer variants, supporting joint reuse of the experience store and retrieval policy in this setting.

Large-to-Small Transfer Retains a Gap to Target-Specific Training. In Figure 2(b), the transferred variants remain below target-specific ExpHarness, while Graph+Copilot Transfer performs best among the transfer variants across most displayed domains. The panels use different axis ranges, so their polygon areas cannot quantify the relative difficulty of the transfer directions. Without a target No-Memory reference in this figure, these comparisons alone do not establish improvement over the unaugmented target executor.

Joint Transfer Also Performs Well for Reasoning-Oriented Targets. In Figure 2(c), Graph+Copilot Transfer approaches target-specific ExpHarness, including on ALFWorld and AppWorld. This observation is consistent with the reuse of experience across the evaluated model pairs. Because model identity and reasoning capability change together, the experiment does not isolate reasoning capability as the cause of transfer performance.

G.3 Analysis of the Four Ablation Settings

To examine four experience-retrieval design choices in ExpHarness, we conduct ablation studies across five evaluation domains: QA, Reasoning, Coding, ALFWorld, and AppWorld. These variants examine the effects of similarity-aware experience management, graph-structured experience organization, graph diffusion, and utility-aware ranking. Results are reported in Figure 3.

- **w/o Similarity Filtering**: Removes similarity-based filtering during experience insertion, allowing redundant experiences to be repeatedly retained.
- **Flat Experience**: Replaces the experience graph with a flat experience pool and retrieves top-K experiences purely by semantic similarity.
- **w/o Graph Diffusion**: Keeps the graph structure but disables graph expansion, retrieving only from the initial semantic seed nodes.
- **w/o Utility Ranking**: Ranks candidate experiences only by semantic similarity, without using historical utility statistics.

Across the reported domains, each of these four variants performs below the full method. **Flat Experience** shows the largest drop, but it changes both candidate generation and the ranking rule, so its gap cannot be attributed to relational structure alone. **w/o Graph Diffusion** and **w/o Utility Ranking** show lower scores under their respective retrieval changes, while **w/o Similarity Filtering** shows a smaller decline. These results support the combined retrieval design under the tested configurations. They do not separately isolate the effects of copilot training, the reward terms, or the node-utility update rule.

H CASE STUDIES

This appendix presents examples across ExpSuite-Static (QA, mathematical reasoning, and code generation) and ExpSuite-Agentic. For each method, the tables show task inputs, reference outcomes, experience context where applicable, and abbreviated task-solving responses. For `ExpHarness`, final answers and actions are produced by the frozen executor; retrieval controls belong to the copilot. Ground-truth fields in these case-study tables are reference annotations, not instructions for the copilot to generate answers.

No Memory case studies are in Table 55–59, ReasoningBank in Table 60–63, ExpeL in Table 64–67, LightMem in Table 68–72, Mem0 in Table 73–77, AWM in Table 78–82, IRCoT in Table 83–87, Search-o1 in Table 88–92, MemRL in Table 93–97.

Table 55: **No Memory case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball? A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s
Ground Truth: D
Using the formula $p = mv$: $p = 8.0\,\text{kg} \times 2.0\,\text{m/s} = 16.0\,\text{kg} \cdot \text{m/s}$. A (4.0) and B (6.0) and C (10.0) are all incorrect; D matches the calculated value. `<answer> D </answer>`

Table 56: **No Memory case study in Math.**

Question: Trevor is a comic book illustrator. In the past three months, he has drawn 220 pages. The comic releases a new issue once a month. The first and second months' issues were the same number of pages, but the third month's issue was four pages longer than either of the others. How many pages was the first issue?
Ground Truth: 72
Let the first and second issue each have x pages; the third has $x + 4$ pages. $x + x + (x + 4) = 220 \;\Rightarrow\; 3x + 4 = 220 \;\Rightarrow\; 3x = 216 \;\Rightarrow\; x = 72$. `#### 72`

Table 57: **No Memory case study in Code Generation.**

Question: Return the largest prime factor of `n`. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5`, `assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm (divide by smallest prime factors iteratively).

Divide `n` by the smallest prime (2) while divisible, then try odd divisors up to $\sqrt{n}$; the remaining value is the largest prime factor.

```
####
```python
def largest_prime_factor(n:  int) -> int:
  i = 2
  while i * i <= n:
    if n % i:  i += 1
    else:  n //= i
  return n
```
```

Table 58: **No Memory case study in ALFWorld.**

Task: put some remotecontrol on armchair.

Outcome: Score = 1.0, Steps = 4

Retrieved Memories: *(none — No-Memory baseline; the agent acts from prompt + admissible actions only.)*

Action Trace:
go to coffeetable 1 (**found remotecontrol 1, 2**) → take remotecontrol 1 → go to armchair 1 → **move remotecontrol 1 to armchair 1** ✓

Table 59: **No Memory case study in AppWorld.**

Task: How much money have I sent to others on Venmo this month so far?

Outcome: Score = 0.50, Steps = 21

Retrieved Memories: *(none — No-Memory baseline.)*

Final Action:
`apis.supervisor.complete_task(answer="433.0")`

Table 60: **ReasoningBank case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball? A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s
Ground Truth: D
Retrieved Memory 1 (✓ Success Strategy): **Calculating Momentum** — When calculating momentum, ensure to multiply the given mass and velocity accurately. This example highlights the importance of correctly applying the formula and performing the multiplication to avoid common mistakes. **Retrieved Memory 2 (✗ Lesson from Failure):** **Impulse and Momentum in Collisions** — In a collision, the total change in momentum of the system is zero, but the lighter object experiences a greater change in momentum due to its lower mass.
Using $p = mv$: $p = 8.0 \times 2.0 = 16.0\,\text{kg} \cdot \text{m/s}$. Options A, B, C are all incorrect. D matches the calculated value. `<answer> D </answer>`

Table 61: **ReasoningBank case study in Math.**

Question: Trevor is a comic book illustrator. In the past three months, he has drawn 220 pages. The comic releases one issue per month. The first and second months' issues were equal; the third was four pages longer. How many pages was the first issue?
Ground Truth: 72
Retrieved Memory 1 (✗ Lesson from Failure): **Misinterpretation of the Problem** — The mistake was in not accounting for all components described in the problem. Carefully read each condition before setting up equations. **Retrieved Memory 2 (✗ Lesson from Failure):** **Misinterpretation of the Problem** — Always verify that all constraints are captured in the equation before solving.
Let x = pages in issue 1 and 2; issue 3 = $x + 4$. $x + x + (x + 4) = 220 \;\Rightarrow\; 3x = 216 \;\Rightarrow\; x = 72$. `#### 72`

Table 62: **ReasoningBank case study in Code Generation.**

Question: Return the largest prime factor of n. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5, assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm.

Retrieved Memory 1 (✓ Success Strategy):
Finding the Largest Perfect Square Factor — Decompose the number into its prime factors; select the highest even power for each prime.
Retrieved Memory 2 (✗ Lesson from Failure):
Efficient Prime Factorization — Always use the square root to limit the search space for prime factors, and handle the remaining factor after the loop.

```
####
```python
def largest_prime_factor(n:  int) -> int:
  while n % 2 == 0:
    max_prime = 2; n //= 2
  for i in range(3, int(n**0.5)+1, 2):
    while n % i == 0:
      max_prime = i; n //= i
  if n > 2:  max_prime = n
  return max_prime
```
```

Table 63: **ReasoningBank case study in AppWorld.**

Task: Give me a random funny quote from my SimpleNote note about it. Just the quote, nothing else.

Outcome: Score = 0.67, Steps = 13

Retrieved Memories:
Retrieved memory items (top-K cosine over reasoning rationales):
- When asked for a quote from a SimpleNote, list and search notes for keyword "quote" or "funny", then read the matching note's content.
- Use `apis.simple_note.list_notes(access_token=..., title_query="quote")` to enumerate candidates before reading.
- For "random" selections, pick uniformly from the matched lines using Python `random.choice`.
[... additional rationale items omitted ...]

Final Action:
`apis.supervisor.complete_task(answer="- I'm on a whiskey and chocolate diet.  I've lost three days, but I don't care.")`

Table 64: **ExpeL case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball? A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s
Ground Truth: D
Retrieved Memory 1 (✓ Success Strategy): **Calculating Momentum** — When calculating momentum, ensure to multiply the given mass and velocity accurately. **Retrieved Memory 2 (✓ Success Strategy):** **Impulse-Momentum Theorem for Impact Problems** — Calculate the change in momentum by determining initial and final velocities. Use the impulse-momentum theorem to find the average force.
$p = mv = 8.0 \times 2.0 = 16.0\,\text{kg} \cdot \text{m/s}$. A (4.0), B (6.0), C (10.0) are all incorrect; D (16.0) matches. `#### D`

Table 65: **ExpeL case study in Math.**

Question: Trevor is a comic book illustrator (220 pages total, 3 issues, first two equal, third is +4 pages). How many pages was the first issue?
Ground Truth: 72
Retrieved Memory 1 (✓ Success Strategy): **Solving Equations with Word Problems** — Identify variables and relationships, set up the equation, and solve it methodically. **Retrieved Memory 2 (✓ Success Strategy):** **Solving Word Problems with Equations** — Express the unknown in terms of a variable, form an equation from the given information, simplify and solve.
Let x = pages in issues 1 and 2; issue 3 = $x + 4$. $3x + 4 = 220 \;\Rightarrow\; x = 72$. `#### 72`

Table 66: **ExpeL case study in Code Generation.**

Question: Compute the n-th element of the Fib4 sequence: `fib4(n) = fib4(n-1) + fib4(n-2) + fib4(n-3) + fib4(n-4)`, with `fib4(0)=0, fib4(1)=0, fib4(2)=2, fib4(3)=0`. No recursion.

Ground Truth: Iterative window of size 4.

Retrieved Memory 1 (✓ Success Strategy):
Identifying Periodic Patterns in Recursive Sequences — When dealing with recursive sequences, compute initial values and look for repeating patterns that can be exploited for efficiency.

Retrieved Memory 2 (✓ Success Strategy):
Telescoping Series in Fibonacci Sums — For sum-based recurrences, check if closed-form or iterative window approaches avoid redundant computation.

```python
def fib4(n:  int) -> int:
  if n in (0, 1, 3):  return 0
  if n == 2:  return 2
  w = [0, 0, 2, 0]
  for i in range(4, n + 1):
    w = [w[1], w[2], w[3], sum(w)]
  return w[3]
```

Table 67: **ExpeL case study in ALFWorld.**

Task: put two spraybottle in garbagecan.

Outcome: Score = 1.0, Steps = 8

Retrieved Memories:
ExpeL exemplars + induced rules (for similar pick-two tasks):
- rule: For "put two X in Y" tasks, locate *both* X instances before placing — do not relocate the first to Y until the second is found, to avoid backtracking.
- exemplar: Successful trial T2019... for `put-two-spraybottle-in-garbagecan-3`: agent visited sidetable 1 first (found two spraybottles together), then garbagecan 1, completing in 7 steps.
[... additional exemplars and rules omitted ...]

Action Trace:
go to sidetable 1 (**found spraybottle 1, spraybottle 2**) → take spraybottle 1 → go to garbagecan 1 → move spraybottle 1 to garbagecan 1 → go to sidetable 1 → take spraybottle 2 → go to garbagecan 1 → **move spraybottle 2 to garbagecan 1** ✓

Table 68: **LightMem case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball? A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s
Ground Truth: D
Retrieved LTM Item 1 (✗ Lesson from Failure): **Incorrect Calculation of Change in Momentum** — Carefully consider the velocity before and after the event and accurately apply the change-in-momentum equation. Double-check calculations and consider all factors that may affect velocity. **Retrieved LTM Item 2 (✓ Success Strategy):** **Energy Conversion in Projectile Motion** — Identify the initial and final states of energy (potential vs. kinetic) and apply conservation principles.
$p = m \times v = 8.0 \times 2.0 = 16.0\,\text{kg} \cdot \text{m/s}$. A, B, C are all inconsistent with this value; D is correct. <answer> D </answer>

Table 69: **LightMem case study in Math.**

Question: Trevor is a comic book illustrator (220 pages total, 3 issues, first two equal, third is +4 pages). How many pages was the first issue?
Ground Truth: 72
Retrieved LTM Item 1 (✓ Success Strategy): **Calculate the Relevant Subset Before Proceeding** — Identify and exclude irrelevant components before setting up the main equation to avoid incorrect calculations. **Retrieved LTM Item 2 (✓ Success Strategy):** **Breaking Down Complex Problems into Manageable Steps** — Decompose the problem into sub-goals, solve each in sequence, and combine results.
Let x = pages in issues 1 and 2; issue 3 = $x + 4$. $3x + 4 = 220 \;\Rightarrow\; x = 72$. #### 72

Table 70: **LightMem case study in Code Generation.**

Question: Return the largest prime factor of `n`. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5, assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm.

Retrieved LTM Item 1 (✓ Success Strategy):
Identifying the Largest Prime Factor in a Factorial — Start from the largest prime $\leq$ the upper limit; for general n, use trial division up to $\sqrt{n}$ and track the largest divisor.
Retrieved LTM Item 2 (✗ Lesson from Failure):
Optimizing Prime Factorization — After the loop, if the remaining $n > 1$, it is itself a prime factor and should be recorded.

```python
def largest_prime_factor(n:  int):
  largest_factor = 1
  for i in range(2, int(n**0.5) + 1):
    while n % i == 0:
      largest_factor = i; n //= i
  if n > 1:  largest_factor = n
  return largest_factor
```

Table 71: **LightMem case study in ALFWorld.**

Task: put a soapbottle in countertop.

Outcome: Score = 1.0, Steps = 12

Retrieved Memories:
Long-term memory (relevant facts, top-20 by cosine):
- [2026-04-29 Wed] Soapbottle items are commonly stored inside cabinets in bathroom layouts.
- [2026-04-28 Tue] Cabinet 4 contains soapbottle 1, soapbottle 2, and candle 2 in this layout.
- [2026-05-06 Wed] User moved cloth 1 to cabinet 1 in a similar pick-and-place task.
- [2026-04-29 Wed] To complete a "put X in countertop" task: take X from its location, then move X to countertop 1.
- [2026-04-28 Tue] Cabinet 1 typically holds candle, soapbar, spraybottle (not soapbottle).
[... additional LTM facts omitted ...]

Action Trace:
explore cabinets 2–3 (no soapbottle) → go to cabinet 1 (soapbar/spraybottle only) → go to cabinet 4 (**found soapbottle 1, 2**) → take soapbottle 1 → go to countertop 1 → **move soapbottle 1 to countertop 1** ✓

Table 72: **LightMem case study in AppWorld.**

Task: Chelsea and I have been carpooling to work this month. They have requested money for it on Venmo. Accept it.

Outcome: Score = 0.83, Steps = 17

Retrieved Memories:
Long-term memory (relevant facts, top-20 by cosine):
- [2026-05-04 Mon] User authenticated with venmo before via `apis.venmo.login`.
- [2026-05-04 Mon] Venmo balance can be checked via `apis.venmo.show_account_balance(access_token=...)`.
- [2026-05-04 Mon] Pending payment requests are listed via `apis.venmo.list_pending_payment_requests`.
- [2026-05-04 Mon] To approve a request, call `apis.venmo.approve_payment_request(payment_request_id=..., access_token=...)`.
[... additional LTM facts omitted ...]

Final Action:
```
print(apis.venmo.approve_payment_request(
  access_token=venmo_token,
  payment_request_id=6097))
apis.supervisor.complete_task(answer="The payment request from Chelsea
for $520.00 has been successfully approved.")
```

Table 73: **Mem0 case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball?
A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s

Ground Truth: D

Retrieved Procedural Summary (✓):
procedural_summary — Task Objective: Analyze options to determine the correct answer about energy conversion when a bowling ball is released. Sequential actions: (1) analyze option A (stored energy); (2) analyze option B (kinetic energy of pins); (3) analyze option C; (4) select the option consistent with conservation of energy.

Using $p = m \times v$: $p = 8.0 \times 2.0 = 16.0\,\text{kg} \cdot \text{m/s}$.
Comparing with the options, D (16.0 kg×m/s) is correct.
`<answer> D </answer>`

Table 74: **Mem0 case study in Math.**

Question: Trevor is a comic book illustrator (220 pages total, 3 issues, first two equal, third is +4 pages). How many pages was the first issue?

Ground Truth: 72

Retrieved Procedural Summary (✓):
procedural_summary — Task Objective: Solve a word problem to determine average daily tickets. Sequential actions: (1) calculate tickets given in first 15 days; (2) subtract from the monthly goal; (3) divide remaining by days left; (4) verify by checking the computed average.

Step 1: define x = pages in issues 1 and 2; issue 3 = $x + 4$.
Step 2: $x + x + (x + 4) = 220 \;\Rightarrow\; 3x = 216 \;\Rightarrow\; x = 72$.
`#### 72`

Table 75: **Mem0 case study in Code Generation.**

Question: Return the largest prime factor of `n`. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5, assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm.

Retrieved Procedural Summary (✓):
procedural_summary — Task: Find the largest prime factor of 97!. Actions: (1) understand that 97! = product of all integers 1–97; (2) identify the largest prime ≤ 97, which is 97 itself; (3) return 97.

```
“‘python
def largest_prime_factor(n:  int):
  i = 2
  while i * i <= n:
    if n % i:  i += 1
    else:  n //= i
  return n
“‘
```

Table 76: **Mem0 case study in ALFWorld.**

Task: put some spraybottle on toilet.
Outcome: Score = 1.0, Steps = 9
Retrieved Memories: ## Relevant memories from past experience: - [2026-05-06] Spraybottle items are commonly found inside cabinets in bathroom layouts. - [2026-05-06] Cabinet 2 contains spraybottle 1, candle 2, cloth 2 in this layout. - [2026-05-06] To complete a "put X on toilet" task: navigate to toilet 1 and call `move X to toilet 1`. - [2026-05-06] Cabinet 1 typically holds soapbottle and toiletpaper, not spraybottles. [... additional fact entries omitted ...]
Action Trace: go to cabinet 1 (no spraybottle) → go to cabinet 3 (toiletpaper only) → go to cabinet 2 (**found spraybottle 1**) → take spraybottle 1 → go to toilet 1 → **move spraybottle 1 to toilet 1** ✓

Table 77: **Mem0 case study in AppWorld.**

Task: I keep a log of all my liked songs and respective artists in a note in simple_note. I want to add release month information for them as well. [...]
Outcome: Score = 0.75, Steps = 31
Retrieved Memories: ## Relevant memories from past experience: - [2026-05-06] User has authenticated with simple_note app via `apis.simple_note.login`. - [2026-05-06] Note IDs in simple_note are queried via `apis.simple_note.list_notes`. - [2026-05-06] Spotify track release date is in `apis.spotify.show_track(track_id=...).release_date`. - [2026-05-06] To update a simple_note, call `apis.simple_note.update_note(note_id=..., content=...)`. [... additional fact entries omitted ...]

Final Action:
`apis.simple_note.update_note(access_token=access_token, note_id=3084, content=updated_content)`
`apis.supervisor.complete_task(answer="The release month information has been added to the songs in the note.")`

Table 78: **AWM case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball?
A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s

Ground Truth: D

Workflow for `arc_c`:
Workflow 1: Analyze Each Option and Explain Your Reasoning — (1) Identify the question. (2) List all options. (3) Analyze each option for relevance, feasibility, and potential outcomes. (4) Evaluate strengths and weaknesses. (5) Select the most suitable option.
Workflow 2: Identify the Key Characteristics... [abbreviated]

Following Workflow 1: apply $p = mv = 8.0 \times 2.0 = 16.0$. A, B, C are all less than 16.0; D matches.
`<answer> D </answer>`

Table 79: **AWM case study in Math.**

Question: Trevor is a comic book illustrator (220 pages total, 3 issues, first two equal, third is +4 pages). How many pages was the first issue?

Ground Truth: 72

Workflow for `gsm8k`:
Workflow 3: Set Up and Solve an Equation — (1) Identify the unknown variable. (2) Set up an equation from the problem statement. (3) Simplify and solve. (4) Verify the answer satisfies all constraints.

Workflow 3: let x = first issue pages; $x + x + (x + 4) = 220$.
$3x = 216 \Rightarrow x = 72$.
`#### 72`

Table 80: **AWM case study in Code Generation.**

Question: Return the largest prime factor of `n`. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5, assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm.

Workflow for `humaneval_plus`:
Divide n by the smallest prime (2) while divisible; then increment to the next divisor and repeat. The last divisor used before n reaches 1 is the largest prime factor. Handle the case where the remaining $n > 1$ after the loop.

```
```python
def largest_prime_factor(n:  int):
  i = 2
  while i * i <= n:
    if n % i == 0:  n //= i
    else:  i += 1
  return n
```
```

Table 81: **AWM case study in ALFWorld.**

Task: clean some dishsponge and put it in countertop.
Outcome: Score = 1.0, Steps = 16
Retrieved Memories: ## AWM workflow (induced from N=5 successful trials of pick_clean_then_place_in_recep): Step 1: navigate to candidate locations of {object} (shelf, drawer, cabinet) Step 2: take {object} {idx} from {location} Step 3: navigate to sinkbasin 1 Step 4: clean {object} {idx} with sinkbasin 1 Step 5: navigate to {target_recep} 1 Step 6: move {object} {idx} to {target_recep} 1
Action Trace: go to sinkbasin 2 (empty) → go to shelf 1 (**found dishsponge 1**) → take dishsponge 1 → go to sinkbasin 1 → clean dishsponge 1 with sinkbasin 1 → go to countertop 1 → **move dishsponge 1 to countertop 1** ✓

Table 82: **AWM case study in AppWorld.**

Task: How much have I paid in phone bill on Venmo this year so far?
Outcome: Score = 0.50, Steps = 40
Retrieved Memories: ## AWM workflow (induced from N=5 successful trials of venmo aggregate-amount queries): Step 1: get supervisor token via `apis.supervisor.show_account_passwords` Step 2: log into venmo: `apis.venmo.login(username=..., password=...)` Step 3: list transactions: `apis.venmo.show_transactions(access_token=...)` Step 4: filter by description containing {keyword} and date $\geq$ year-start Step 5: sum the matching amounts and complete the task

Final Action:
`print(apis.api_docs.show_api_descriptions(app_name="supervisor"))`
`# (workflow truncated at 40 steps; agent collected partial total)`

Table 83: **IRCoT case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball?
A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s

Ground Truth: D

BM25-Retrieved Example (Round 1):
Calculating Momentum — When calculating momentum, ensure to multiply the given mass and velocity accurately.
BM25-Retrieved Example (Round 2):
Calculating Momentum — Correctly apply the formula $p = mv$ and perform the multiplication to avoid common mistakes.

B (6.0): $8.0 \times 2.0 \neq 6.0$. C (10.0): incorrect. D (16.0): $8.0 \times 2.0 = 16.0\,\text{kg} \cdot \text{m/s}$ — correct.
`#### D`

Table 84: **IRCoT case study in Math.**

Question: Trevor is a comic book illustrator (220 pages total, 3 issues, first two equal, third is +4 pages). How many pages was the first issue?

Ground Truth: 72

BM25-Retrieved Example (Round 1):
Misinterpretation of the Problem — Always read all conditions before setting up equations.
BM25-Retrieved Example (Round 2):
Misinterpretation of the Problem — Faisal writes responses to questions; the problem involves two separate writers.

Let x = pages in issues 1 and 2; $3x + 4 = 220 \Rightarrow x = 72$.
`#### 72`

Table 85: **IRCoT case study in Code Generation.**

Question: Return the largest prime factor of `n`. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5, assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm.

BM25-Retrieved Example (Round 1):
Finding the Largest Perfect Square Factor — Decompose into prime factors; select highest even powers.

````
####
```python
def largest_prime_factor(n:  int) -> int:
  i = 2
  while i * i <= n:
    if n % i:  i += 1
    else:  n //= i
  return n
```
````

Table 86: **IRCoT case study in ALFWorld.**

Task: put two spraybottle in cabinet.
Outcome: Score = 1.0, Steps = 10
Retrieved Memories: Retrieved past experience (chained CoT, max_hops=2, top-K=10): - act: go to cabinet 1 → obs: You arrive at cabinet 1. The cabinet 1 is open. In it, you see soapbottle 3, soapbottle 2. - act: go to countertop 1 → obs: … a spraybottle 1 … - act: take spraybottle 1 from countertop 1 → obs: You pick up the spraybottle 1. - [completed OK] task: put two spraybottle in cabinet | actions: go to countertop 1 → take spraybottle 1 → go to cabinet 1 → move spraybottle 1 to cabinet 1 → go to bathroom → take spraybottle 2 → … [… additional chunks omitted …]
Action Trace: explore cabinets 1–3 → found spraybottle 2 → take and **move spraybottle 2 to cabinet 1** → go to countertop 1 (**found spraybottle 1**) → take spraybottle 1 → **move spraybottle 1 to cabinet 1** ✓

Table 87: **IRCoT case study in AppWorld.**

Task: Approve all venmo payment requests from my roommates from this calendar month.
Outcome: Score = 0.86, Steps = 3
Retrieved Memories: Retrieved past experience (chained CoT, max_hops=2, top-K=10): - act: `roommate_emails = [u["email"] for u in apis.supervisor.show_relations(relation="roommate"` → obs: list of roommate emails returned - act: `requests = apis.venmo.list_pending_payment_requests(access_token=...)` → obs: pending requests dict - [completed OK] task: approve roommates' venmo requests | actions: log in → pull roommate emails → filter by current month → approve in loop [… additional chunks omitted …]

Final Action:
`current_month, current_year = pendulum.now().month, pendulum.now().year`
`for request in payment_requests:`
`  created_at = pendulum.parse(request["created_at"])`
`  if request["sender"]["email"] in roommate_emails and \`
`  created_at.month == current_month and created_at.year == current_year:`
`  apis.venmo.approve_payment_request(access_token=venmo_access_token, payment_request_id=...)`
`apis.supervisor.complete_task(status="success")`

Table 88: **Search-o1 case study in QA.**

Question: A bowling ball with a mass of 8.0 kg rolls down a bowling lane at 2.0 m/s. What is the momentum of the bowling ball?
A. 4.0 kg×m/s B. 6.0 kg×m/s C. 10.0 kg×m/s D. 16.0 kg×m/s

Ground Truth: D

<|begin_search_query|>
A bowling ball with mass 8.0 kg rolls at 2.0 m/s. What is its momentum? A. 4.0 B. 6.0 C. 10.0 D. 16.0 kg×m/s
<|end_search_query|>
<|begin_search_result|>
Final Information
Momentum Formula Insight
Momentum is defined as $p = mv$. Multiply the object's mass by its velocity to get momentum in kg·m/s. Common errors include adding rather than multiplying m and v.<|end_search_result|>

To find the momentum I apply $p = mv$.
Given: $m = 8.0\,\text{kg}$, $v = 2.0\,\text{m/s}$.
$p = 8.0 \times 2.0 = 16.0\,\text{kg} \cdot \text{m/s}$.
Checking options:
- A (4.0): would require $8.0 \div 2.0$, wrong operation.
- B (6.0): $8.0 + 2.0 - 4.0$, no physical basis.
- C (10.0): $8.0 + 2.0$, wrong operation.
- D (16.0): $8.0 \times 2.0 = 16.0$ ✓

The answer is D.

Table 89: **Search-o1 case study in Math.**

Question: Trevor is a comic book illustrator. He has just started on a new series. The first issue took him 220 pages to illustrate. If the series is 3 issues long, the first two issues are the same length, and the third issue is 4 pages longer than the others, how many pages was the first issue?

Ground Truth: 72

<|begin_search_query|>
Comic book illustrator: 3 issues, 220 pages total. First two equal, third is 4 pages longer. How many pages in the first issue?
<|end_search_query|>
<|begin_search_result|>
Final Information
Algebraic Equation Strategy
For problems with equal parts and a small offset, let x denote the equal part. Write a linear equation summing all parts to the total, then solve for x. Watch for off-by-one errors when the offset applies to only one term.<|end_search_result|>

Let x = pages in issue 1 (and issue 2). Issue 3 = $x + 4$.
Total equation:
$x + x + (x + 4) = 220$
$3x + 4 = 220$
$3x = 216$
$x = 72$
The first issue had **72** pages. Verification: $72 + 72 + 76 = 220$ ✓

72

Table 90: **Search-o1 case study in Code Generation.**

Question: Return the largest prime factor of `n`. Assume $n > 1$ and is not a prime.
Tests: `assert largest_prime_factor(15) == 5, assert largest_prime_factor(27) == 3`

Ground Truth: Trial-division algorithm.

<|begin_search_query|>
Python function to return the largest prime factor of n, where n > 1 and n is not prime.
<|end_search_query|>
<|begin_search_result|>
Final Information
Trial Division Strategy
To find the largest prime factor: (1) divide n by 2 while even; (2) iterate odd i from 3 while $i^2 \leq n$, dividing out all multiples; (3) if $n > 1$ remains, it is the largest prime factor. Time complexity $O(\sqrt{n})$.<|end_search_result|>

I need to find the largest prime factor of n.
Algorithm (trial division):
1. Handle factor 2: while n is even, divide by 2; track last factor.
2. Handle odd factors: for $i = 3, 5, \ldots$ while $i^2 \leq n$, divide out all copies of i.
3. If $n > 1$ after the loop, n itself is the largest prime factor.
Trace on `largest_prime_factor(15)`: $15/3 = 5$; $5 > 1 \Rightarrow$ return 5 ✓
Trace on `largest_prime_factor(27)`: $27/3 = 9$, $9/3 = 3$, $3/3 = 1$; last factor = 3 ✓
The cleanest implementation uses a running variable i starting at 2 and repeatedly divides n.

```python
def largest_prime_factor(n:  int) -> int:
  i = 2
  while i * i <= n:
    if n % i:  i += 1
    else:  n //= i
  return n
```

Table 91: **Search-o1 case study in ALFWorld.**

Task: cool some bread and put it in countertop.

Outcome: Score = 1.0, Steps = 11

Retrieved Memories:
Search-o1 trace (planner-driven iterative search, ≤ 3 rounds):
⟨search⟩ how to cool an item in ALFWorld kitchen ⟨/search⟩
→ retrieved: "open fridge 1 → cool {X} with fridge 1".
⟨search⟩ where can bread be found in seen ALFWorld layouts ⟨/search⟩
→ retrieved: "countertop 2 commonly contains bread, egg, pan, kettle in this layout".

Action Trace:
go to countertop 2 (**found bread 1**) → take bread 1 → go to fridge 1 → open fridge 1 → cool bread 1 with fridge 1 → go to countertop 1 → **move bread 1 to countertop 1** ✓

Table 92: **Search-o1 case study in AppWorld.**

Task: Update all my Spotify playlist titles with the most common song genre in that playlist in this format: "<original_title> | <most_common_genre>". [...]
Outcome: Score = 0.83, Steps = 6
Retrieved Memories: Search-o1 trace (planner-driven iterative search): ⟨search⟩ how to enumerate Spotify playlists for the current user ⟨/search⟩ → retrieved: "`apis.spotify.show_playlists(access_token=...)` returns `playlist_id, title, tracks`". ⟨search⟩ how to update a playlist title ⟨/search⟩ → retrieved: "`apis.spotify.update_playlist(playlist_id, title=...)`".

Final Action:
`# (loop body: for each playlist, compute mode of track genres, update title)`
`apis.supervisor.complete_task(answer="All Spotify playlist titles have been updated with the most common song genre.")`

Table 93: **MeMRL case study in QA.**

Question: The speed of a sound wave varies as it travels through different substances. Which factor will most affect the speed of a sound wave? A. the frequency of the wave B. the wavelength of the wave C. the source that created the sound D. the distance between molecules in the substance
Ground Truth: D
Retrieved Memory 1 (✓ Success, Q = 0.82): **Sound Transmission in Media** — Consider the properties of each medium (rock, seawater, space, atmosphere) and compare their speeds of sound transmission. The speed of sound depends on molecular density and arrangement. **Retrieved Memory 2 (✗ Failure, Q = 0.31):** **Wave Properties Confusion** — Incorrectly assumed that frequency determines speed; frequency affects pitch, not propagation speed. **Retrieved Memory 3 (✓ Success, Q = 0.76):** **Molecular Density and Sound** — The closer the molecules in a substance, the faster sound can propagate. This is why sound travels faster in solids than in gases.
The speed of sound depends on the medium's properties, specifically the distance between molecules. Frequency (A) affects pitch, wavelength (B) is derived from speed and frequency, and the source (C) creates the wave but doesn't determine speed. The molecular arrangement (D) is the key factor. `<answer> D </answer>`

Table 94: **MeMRL case study in Math.**

Question: There are 15 girls and 10 boys in Ms. Smith's class. She divided 375 books equally among them. How many books did all the girls get combined?
Ground Truth: 225
Retrieved Memory 1 (✓ Success, Q = 0.91): **Division and Grouping** — First find the total number of students, then compute per-student share, then multiply by the target group size. **Retrieved Memory 2 (✓ Success, Q = 0.85):** **Boys and Girls Ratio** — Correctly represent the relationship between groups using subtraction from the total, then allocate proportionally. **Retrieved Memory 3 (✗ Failure, Q = 0.22):** **Misapplied Division** — Divided books only among girls instead of all students; always check whether the division applies to the full group.
Total students $= 15 + 10 = 25$. Books per student $= 375 \div 25 = 15$. Girls' total $= 15 \times 15 = 225$. `<answer> 225 </answer>`

Table 95: **MeMRL case study in Code Generation.**

Question: Given a class name and a list of extensions, find the strongest extension. The strength is CAP−SM, where CAP is the count of uppercase letters and SM is the count of lowercase letters in the extension name. Return `"ClassName.StrongestExt"`.
Tests: `assert Strongest_Extension('my_class', ['AA', 'Be', 'CC']) == 'my_class.AA'`

Ground Truth: Iterate extensions, compute CAP−SM for each, track the maximum.

Retrieved Memory 1 (✓ Success, Q = 0.78):
String Character Counting — Define the function signature correctly, iterate through characters, and use `isupper()`/`islower()` to count.
Retrieved Memory 2 (✓ Success, Q = 0.72):
Tracking Maximum with Index — Initialize the strongest extension as the first element, then compare each subsequent extension's strength.
Retrieved Memory 3 (✗ Failure, Q = 0.18):
Incomplete Implementation — Correctly defined the function parameters but left the iteration logic as a placeholder; always complete the loop body.

```
def Strongest_Extension(class_name, extensions):
  strongest = extensions[0]
  max_str = sum(1 for c in extensions[0] if c.isupper())
        - sum(1 for c in extensions[0] if c.islower())
  for ext in extensions[1:]:
    s = sum(1 for c in ext if c.isupper())
       - sum(1 for c in ext if c.islower())
    if s > max_str:  max_str = s; strongest = ext
  return class_name + '.'  + strongest
```

Table 96: **MeMRL case study in ALFWorld.**

Task: put two cd in drawer.

Outcome: Score = 1.0, Steps = 5

Retrieved Memories:
Past successful trajectories (reward-weighted, top-similar):

```
Your task is to:  put two cellphone in dresser.
> go to desk 1
You arrive at desk 1.  ... a cellphone 1, cellphone 2, ...
> take cellphone 1 from desk 1
> move cellphone 1 to dresser 1
> take cellphone 2 from desk 1
> move cellphone 2 to dresser 1
—
```

[... additional reward-weighted trajectories omitted ...]

Action Trace:
go to desk 1 (**found cd 1, cd 2 alongside other items**) → take cd 1 → move cd 1 to drawer 1 → take cd 2 → **move cd 2 to drawer 1** ✓

Table 97: **MeMRL case study in AppWorld.**

Task: I went on a trip with friends to Maui recently. I have maintained a note of money I owe to others and others owe me from the trip in simple_note. Make private venmo payments or requests accordingly. [. . .]

Outcome: Score = 0.86, Steps = 19

Retrieved Memories:
Past successful trajectories (reward-weighted, top-similar):

```
Task:  settle expense splits via venmo from a simple_note ledger.
> Read note via apis.simple_note.show_note(note_id=...)
> Parse "I owe X" / "X owes me" lines into two dicts.
> Resolve names to emails via apis.supervisor.show_relations.
> For each entry, call apis.venmo.send_money OR apis.venmo.create_payment_request
with private=True.
```

[. . . additional reward-weighted trajectories omitted . . .]

Final Action:

```
owe_me_map = {"Cory":  212, "Edwin":  70, "Jeremy":  149}
for person, amount in owe_me_map.items():
  if person in user_emails:
    apis.venmo.create_payment_request(
      user_email=user_emails[person], amount=amount,
      access_token=venmo_token, description="For Maui trip",
private=True)
apis.supervisor.complete_task(answer="Payments and requests for the
Maui trip have been processed.")
```

Table 98: **S3 case study in QA.**

Question: Which property best explains why a metal spoon feels colder than a wooden spoon at the same room temperature? Options: (A) higher density, (B) higher thermal conductivity, (C) lower melting point, (D) stronger magnetism.
Ground Truth: (B) higher thermal conductivity.
<|begin_search_query|> Why does a metal spoon feel colder than a wooden spoon at room temperature? <|end_search_query|> <|begin_search_result|> **Retrieved Information**: Metals conduct heat much faster than wood. When touched, metal draws heat away from the skin more rapidly, making it feel colder even if both objects have the same temperature. <|end_search_result|>
The retrieved passage is directly relevant. The question asks about two objects at the same room temperature, so the answer cannot be about actual temperature difference. The key factor is the rate of heat transfer from the hand to the object. Since metal has higher thermal conductivity than wood, it removes heat from the skin faster and therefore feels colder. **Prediction**: (B) higher thermal conductivity. ✓

Table 99: **S3 case study in Math.**

Question: A store sells notebooks for \$3 each and pens for \$2 each. Alice buys 4 more pens than notebooks and spends \$34 in total. How many notebooks does she buy?
Ground Truth: 6 notebooks.
<|begin_search_query|> solve word problem notebooks 3 dollars pens 2 dollars 4 more pens total 34 <|end_search_query|> <|begin_search_result|> **Retrieved Information**: Let the smaller quantity be x. If pens are 4 more than notebooks, use notebooks $= x$ and pens $= x + 4$. The cost equation is $3x + 2(x + 4) = 34$. <|end_search_result|>
Let x be the number of notebooks. Then the number of pens is $x + 4$. The total cost is $3x + 2(x + 4) = 34$. Thus $3x + 2x + 8 = 34$, so $5x = 26$, giving $x = 5.2$. This is not an integer, so the retrieved setup suggests the problem may be inconsistent. However, checking nearby integer values: if $x = 6$, pens $= 10$, cost $= 18 + 20 = 38$, not 34; if $x = 5$, pens $= 9$, cost $= 15 + 18 = 33$. **Prediction**: No integer solution. ×
Observation: S3 retrieves a useful equation pattern, but because the retrieved information is used mechanically, it exposes the inconsistency rather than correcting the task. This case shows that search-style retrieval can help formulate equations but does not always provide robust reasoning when the query requires verification or adjustment.

Table 100: **S3 case study in Code Generation.**

Question: Write a function `count_vowels(s)` that returns the number of vowels in a string. Treat both uppercase and lowercase vowels as valid.
Tests: `assert count_vowels("Apple") == 2`, `assert count_vowels("WHY") == 0`.

Ground Truth: Count characters whose lowercase form is in `"aeiou"`.

<|begin_search_query|>
Python count vowels in string uppercase lowercase
<|end_search_query|>
<|begin_search_result|>
Retrieved Information: Convert the string to lowercase and sum over characters that appear in the vowel set {'a','e','i','o','u'}.
<|end_search_result|>

The retrieved result gives the exact algorithm. We can lowercase the input and count membership in the vowel set. This handles uppercase vowels such as `"A"` in `"Apple"`, while ignoring consonants such as `"W"`, `"H"`, and `"Y"`.

```python
def count_vowels(s):
  vowels = set("aeiou")
  return sum(1 for ch in s.lower() if ch in vowels)
```

Prediction: Passes the tests. ✓

Table 101: **S3 case study in ALFWorld.**

Task: Put a clean mug in the cabinet.

Ground Truth Plan: Find mug → take mug → clean mug at sink → go to cabinet → put mug in cabinet.

<|begin_search_query|>
ALFWorld clean mug put in cabinet strategy
<|end_search_query|>
<|begin_search_result|>
Retrieved Information: For clean-and-place tasks, first locate the object, take it, move to a sinkbasin, use the sink to clean it, then move to the target receptacle and put the object there.
<|end_search_result|>

The search result provides a useful high-level plan. The agent first searches the countertop and shelves, finds the mug, and takes it. It then navigates to the sinkbasin and performs a cleaning action before moving to the cabinet. However, after cleaning, the agent repeats several navigation actions because the retrieved plan does not specify which cabinet has already been observed.

Outcome: Success, but with redundant steps.

Observation: S3 retrieves a generally correct procedural template, but it does not maintain a structured memory of environment-specific state. This can lead to repeated exploration even when the high-level strategy is correct.

Table 102: **`ExpHarness` case study in QA.**

Question: Which property best explains why a metal spoon feels colder than a wooden spoon at the same room temperature? Options: (A) higher density, (B) higher thermal conductivity, (C) lower melting point, (D) stronger magnetism.
Ground Truth: (B) higher thermal conductivity.
Retrieved Experience Graph Neighborhood: **Seed experience**: When two objects have the same temperature, perceived coldness often depends on how quickly heat leaves the skin, not on the object's actual temperature. **Connected experience**: Metals transfer heat faster than wood or plastic; this makes metal feel colder because it draws heat from the hand more rapidly. **Utility signal**: Similar heat-transfer explanations previously improved science QA accuracy, while density-based distractors were often misleading.
The retrieved experiences separate the relevant physical mechanism from surface distractors. Since the spoon and wooden spoon are at the same room temperature, the answer must explain the sensation of coldness. The useful experience indicates that heat-transfer rate is the key factor, and the connected experience identifies metals as high-conductivity materials. Therefore, the metal spoon feels colder because it conducts heat away from the skin faster. **Prediction**: (B) higher thermal conductivity. ✓

Table 103: **`ExpHarness` case study in Math.**

Question: A store sells notebooks for \$3 each and pens for \$2 each. Alice buys 4 more pens than notebooks and spends \$38 in total. How many notebooks does she buy?
Ground Truth: 6 notebooks.
Retrieved Experience Graph Neighborhood: **Seed experience**: For two-item purchase problems, define the smaller quantity as x and express the larger quantity using the stated difference. **Connected experience**: Always verify the solution by substituting the quantity back into the total-cost equation. **Connected failure lesson**: Do not answer immediately after solving the equation; check whether the result is an integer and satisfies all constraints.
Let x be the number of notebooks. Then the number of pens is $x+4$. The total cost equation is $3x+2(x+4) = 38$. Thus $3x + 2x + 8 = 38$, so $5x = 30$ and $x = 6$. Verification: notebooks cost $6 \times 3 = 18$, pens cost $10 \times 2 = 20$, and the total is 38. **Prediction**: 6 notebooks. ✓

Table 104: **ExpHarness case study in Code Generation.**

Question: Write a function `count_vowels(s)` that returns the number of vowels in a string. Treat both uppercase and lowercase vowels as valid.
Tests: `assert count_vowels("Apple") == 2`, `assert count_vowels("WHY") == 0`.

Ground Truth: Count characters whose lowercase form is in `"aeiou"`.

Retrieved Experience Graph Neighborhood:
Seed experience: For character-counting tasks, normalize the input before applying membership tests.
Connected experience: Use a set for constant-time membership when checking whether characters belong to a small category.
Connected failure lesson: Do not forget uppercase cases when the problem explicitly says the match should be case-insensitive.

The graph retrieves both the algorithmic pattern and the common failure mode. The implementation should lowercase the string before counting vowels, and a set is sufficient for checking membership. This directly handles `"Apple"` because `"A"` becomes `"a"`, and it returns zero for `"WHY"` because none of the characters are vowels after lowercasing.

```python
def count_vowels(s):
  vowels = set("aeiou")
  return sum(1 for ch in s.lower() if ch in vowels)
```

Prediction: Passes the tests. ✓

Table 105: **ExpHarness case study in ALFWorld.**

Task: Put a clean mug in the cabinet.

Ground Truth Plan: Find mug → take mug → clean mug at sink → go to cabinet → put mug in cabinet.

Retrieved Experience Graph Neighborhood:
Seed experience: For clean-and-place tasks, first secure the target object, then clean it at a sinkbasin before moving to the destination receptacle.
Connected experience: After cleaning, avoid re-checking unrelated rooms; go directly to the target receptacle if it has already been observed.
Connected failure lesson: A common failure is putting the object away before cleaning it, which completes the wrong state and causes task failure.

The retrieved experiences provide both a high-level plan and a constraint to avoid a known failure. The agent first locates and takes the mug, moves to the sinkbasin, cleans the mug, and then navigates directly to the cabinet. Because the graph also retrieves a failure lesson about premature placement, the agent avoids putting the mug into the cabinet before cleaning it.

Outcome: Success with fewer redundant exploration steps.