Decocted Experience Improves Test-Time Inference in LLM Agents

Maohao Shen[1]*, Kaiwen Zha[1]*, Zexue He[2],*
Zhang-Wei Hong[3] , Siru Ouyang[4] , J. Jon Ryu[1],
Prasanna Sattigeri[3] , Suhas Diggavi[5] , Gregory Wornell[1]

Massachusetts Institute of Technology[1], Stanford University[2],
MIT-IBM Watson AI Lab[3], University of Illinois at Urbana-Champaign[4],
University of California, Los Angeles[5]
maohao@mit.edu, kzha@mit.edu, zexueh@stanford.edu

Abstract

There is growing interest in improving LLMs without updating model parameters. One well-established direction is test-time scaling, where increased inference-time computation (e.g., longer reasoning, sampling, or search) is used to improve performance. However, for complex reasoning and agentic tasks, naively scaling test-time compute can substantially increase cost and still lead to wasted budget on suboptimal exploration. In this paper, we explore *context* as a complementary scaling axis for improving LLM performance, and systematically study how to construct better inputs that guide reasoning through *experience*. We show that effective context construction critically depends on *decocted experience*. We present a detailed analysis of experience-augmented agents, studying how to derive context from experience, how performance scales with accumulated experience, what characterizes good context, and which data structures best support context construction. We identify *decocted experience* as a key mechanism for effective context construction: extracting essence from experience, organizing it coherently, and retrieving salient information to build effective context. We validate our findings across reasoning and agentic tasks, including math reasoning, web browsing, and software engineering.

1 Introduction

Recent progress in large language models (LLMs) has increasingly shifted attention from optimizing model parameters to improving inference-time behavior. A growing body of work shows that performance can often be improved through test-time scaling (Wu et al., 2024; Snell et al., 2025). This shift is especially important for agentic systems, where an LLM must reason over long horizons, interact with tools or environments, and make a sequence of compounding decisions. In such settings, simply allocating more test-time compute is often not sufficient: extensive interaction can increase cost, and still spend substantial budget exploring sub-optimal paths (Shen et al., 2025; Zeng et al., 2025).

In this paper, we argue that *context* constitutes another important axis for efficient scaling. Rather than increasing the output budget, one can instead provide the agent with carefully constructed inputs (i.e., context) to induce more efficient reasoning. While this perspective is intuitively appealing, there has been limited systematic study on how to construct effective context, what characterizes a good context, and how performance scales with respect to it (Mei et al., 2025; Hua et al., 2025). The most straightforward approach to construct context would be to handcraft prompts or demonstrations for a given task, but it is neither scalable nor reliable: handcrafted context must be redesigned for each new task, and it may fail to align with the actual strengths and failure modes of the deployed agent.

*Co-first Authors.

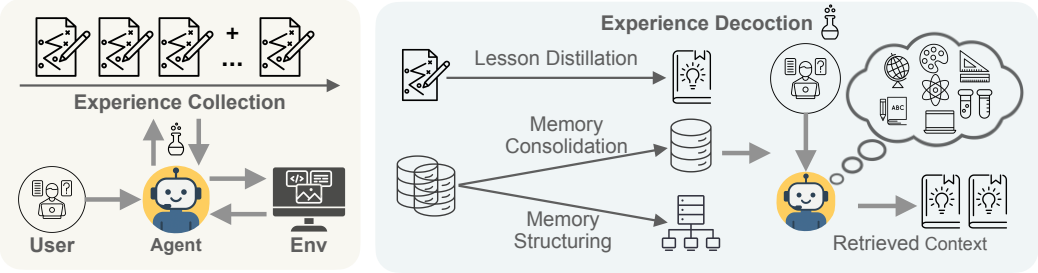


Figure 1: **Experience-Augmented Agent.** The agent accumulates experience from past interactions, decocts it into effective context for improved inference at test time, i.e., distilling lessons from experience, organizing the experience memory, and finally retrieving salient information from it.

A more natural source of high-quality context is *experience*, e.g., through interaction with environments, an agent accumulates trajectories and feedback that contain reusable strategies, workflows, and pitfalls to avoid. Prior work has shown that such prior interaction traces can improve test-time performance (Suzgun et al., 2025; Ouyang et al., 2025; Wei et al., 2025). However, most existing methods focus on specific frameworks for leveraging experience, while a systematic understanding of how raw experience should be transformed into effective context remains limited. It remains unclear **(1) how performance scales with accumulated experience**, **(2) what characterizes effective context theoretically and empirically**, and **(3) what data structures best support context construction**. We address this gap by identifying *decocted experience* as a key ingredient for successful experience-augmented agents: extracting the essence from experience, organizing it coherently, and retrieving salient information to construct effective context. Our main findings are as follows:

1. In Section 3, we study what makes experience-augmented agents scalable along two coupled axes: input context size and accumulated experience size. We show that experience decoction through lesson distillation outperforms raw experience in agentic tasks (Figure 2) and yields better input scalability (Figure 3). We further show that maintaining accumulated experience as memory is critical for scalability: performance exhibits a sweet spot at intermediate memory sizes that can outperform full memory (Figure 4), identifying memory consolidation as a second key factor alongside experience decoction.
2. In Section 4, we investigate how to characterize effective context. Theoretically, we first show that context with higher information gain about the agent's generated output leads to more efficient inference (Proposition 4.1 & Figure 5). Empirically, we show that effective context should balance the trade-off between relevance and diversity, and that this trade-off is associated with the agent's performance improvement (Figure 6).
3. In Section 5, we further study a memory structure tailored for experience-augmented agents. We show that extracting concepts from experience and organizing them into a tree structure across different concept groups improves context construction by encouraging retrieval from more diverse yet still relevant concept groups, leading to better performance than baseline methods (Figure 8).

2 Problem Setup

We study how an agent should *craft context from experience* to efficiently solve new tasks.

Agent. Throughout, we assume that an agent is built upon a frozen LLM $\pi_\theta : \mathcal{C} \to \mathcal{Y}$, where θ denotes pretrained parameters and $\mathcal{C}$ is the space of textual contexts. Given a constructed context $c \in \mathcal{C}$, the model generates $\hat{y} = \pi_\theta(c)$, where $\hat{y}$ can either be a single-turn chain-of-thought (CoT) trace or a trajectory $\hat{y} = (s_1, a_1, \ldots, s_T, a_T)$, where s_t denotes an intermediate state (e.g., environment state) and a_t denotes an action.

Training: Memory Construction from Experience. The goal of this paper is to study how to construct an informative yet succinct context based on *experience*, which is formally defined as follows, and we assume that the agent has access to a set of N problems $\{x_i\}_{i=1}^N$,

where, for each i, m pairs of trajectory and final reward $\mathbf{o}_i := \{(y_i^{(j)}, r_i^{(j)})\}_{j=1}^m$ are given collectively as *experience*. The experience may not be necessarily from the same agent, but all the experiments in this manuscript assume self-experience. We assume the same m for different problems for simplicity. Given a formatting function Ψ, we define $z_i := \Psi(x_i, \mathbf{o}_i)$ for each $i \in [N]$, and call $\mathcal{M} = \{z_i\}_{i=1}^N$ the *raw memory*. Since the size of the raw memory grows linearly in N, the number of previous examples, and thus it may become inefficient to maintain the entire raw memory at test time. Instead, we consider a memory organization mechanism μ that *organizes* $\mathcal{M}$ into a more succinct memory $\tilde{\mathcal{M}} := \mu(\mathcal{M})$.

Test-time: Inference with Memory. Having built the organized memory $\tilde{\mathcal{M}}$, given a new query $x \in \mathcal{X}$, we consider a retrieval mechanism $R : \mathcal{X} \times \tilde{\mathcal{M}} \to 2^{\tilde{\mathcal{M}}}$ which selects a subset of K memory entries relevant to x: $R(x; \tilde{\mathcal{M}}) = \{z_{i_1}, \ldots, z_{i_K}\}$. Here, $K \geq 1$ is a hyperparameter. A context constructor $C : \mathcal{X} \times 2^{\tilde{\mathcal{M}}} \to \mathcal{C}$ then produces a context $c = C(x, R(x; \tilde{\mathcal{M}}))$, which is provided to the agent as a prompt for inference, i.e., $\hat{y} = \pi_\theta(c)$. Throughout the paper, we assume a simple concatenation-based construction.

Scope of Analysis. Given an agent π_θ with past experience $\{(x_i, \mathbf{o}_i)\}_{i=1}^N$, several quantities in the system may grow large in practice, including the number of trajectories per problem (i.e., m), the amount of accumulated experience (i.e., N), and the size of the retrieved context (i.e., K). Our goal is to understand what makes such a system scalable through *experience decoction*: how raw experience should be distilled, organized, and retrieved to construct the context so as to maximize the agent's expected performance on unseen test problems.

2.1 A Minimal Instantiation of the Experience-Augmented Agent

To understand the importance of experience decoction, we begin with a minimal instantiation of the experience-augmented agents that do not rely on sophisticated design choices: **(1) Memory Organization**: We begin with a flat memory and retain only one successful trajectory out of the m trajectories for each entry in order to reduce the retrieved context length, i.e., $\tilde{\mathcal{M}} = \{(x_i, y_i)\}_{i=1}^N$. **(2) Retrieval**: Given a query problem x at test time, we compute embeddings using a fixed encoder $\pi_e : \mathcal{X} \to \mathbb{R}^d$ and define a similarity function $s(x, x_i) = \langle \pi_e(x), \pi_e(x_i) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the standard inner product. The retrieval mechanism selects the top-K entries according to similarity: $R(x, \tilde{\mathcal{M}}) = \mathrm{TopK}_{z_i \in \tilde{\mathcal{M}}}\ s(x, x_i)$. Our analyses in Section 3 and Section 4 are based on this flat experience memory and basic retriever, while in Section 5 we further investigate a more advanced organization system.

2.2 Experimental Setup

Tasks and Datasets. We study three representative tasks: **Mathematical Reasoning (Math Reasoning)**, **Web Browsing (WebShop)**, and **Software Engineering (SWE)**. For math reasoning, we build memory from DAPO-Math (Yu et al., 2025b) and evaluate on AMC, AIME, HMMT, and BeyondAIME using exact-match correctness. For WebShop, we use WebShop (Yao et al., 2022), where an agent searches an online shopping environment to satisfy user requests, and performance is measured by the quality of the final purchase. For SWE, we use SWE-bench (Jimenez et al., 2023), where an agent resolves real GitHub issues by editing code in a repository environment, and success is determined by whether the generated patch passes the evaluation tests. More details are included in Appendix C.1.

Models. We use Seed-OSS-36B-Instruct (ByteDance-Seed, 2025b) as the base language model π_θ in our main experiments. For reasoning tasks, the model generates a single-turn chain-of-thought (CoT) trace followed by a final answer. For agentic tasks, it is equipped with the required external tools and operates in an iterative action–observation loop until termination. We also extend our analysis to another open-source model, GPT-OSS-20B (OpenAI, 2025). We use Qwen3-Embedding-4B (Zhang et al., 2025d) as the embedding model π_e.

Evaluation. We evaluate how leveraging context improves both the effectiveness and efficiency of the LLM agent under a fixed test-time budget. We use two metrics: (1) **Effectiveness:** For a query x, we measure the average quality of multiple attempts as

avg_$m = \frac{1}{m}\sum_{j=1}^{m} r^{(j)}$. (2) **Efficiency:** For reasoning tasks (e.g., Math Reasoning), we measure efficiency by the number of generated output tokens (i.e., the CoT length). For agentic tasks (e.g., WebShop and SWE), we use the number of interaction steps T.

3 What Makes Experience-Augmented Agent Scalable?

As discussed in Section 2, an experience-augmented agent faces scaling challenges along two axes: the input context can become large since retrieved experiences can be lengthy, and the memory itself can grow substantially as experience accumulates. In this section, we argue that *experience decoction* is the key to scalability: locally, it distills raw experience into concise lessons, and globally, it consolidates the memory by removing redundancy.

3.1 Input Size Scaling: How to Construct Effective Context?

Distilled Lessons Are More Effective Than Raw Experience in Agentic Tasks. Given a raw experience memory, i.e., $\tilde{\mathcal{M}} = \mathcal{M} = \{(x_i, \mathbf{o}_i)\}_{i=1}^{N}$, where $\mathbf{o}_i := \{(y_i^{(j)}, r_i^{(j)})\}_{j=1}^{m}$, directly using raw experience as context can be sub-optimal due to lengthy trajectories and large m. This motivates a compression mechanism that we refer to as *lesson distillation*. Specifically, past trajectories are distilled into a single reusable lesson that summarizes transferable reasoning patterns: $\ell_i = \pi_\theta(x_i, \mathbf{o}_i)$, and the distillation is performed by the agent itself. The resulting context becomes $c = \text{Concat}\big(x, \{(x_i, \ell_i)\}_{i=1}^{K}\big)$. To evaluate this idea, for a fixed K, we compare a system built on distilled lessons, i.e., $\{(x_i, \ell_i)\}_{i=1}^{N}$, against the raw experience-based minimal instantiation in Section 2.1. Results are shown in Figure 2. Compared to a vanilla agent without any context, both methods substantially improve effectiveness (avg_m) and efficiency. On reasoning tasks, lesson-based context slightly underperforms raw experience-based context, suggesting that detailed reasoning traces already provide sufficient guidance. In contrast, on WebShop and SWE tasks, lesson-based context consistently outperforms raw experience. Since Agentic trajectories contain extensive environment observations and interaction noise, distillation effectively filters irrelevant information and retains transferable decision-making tricks.

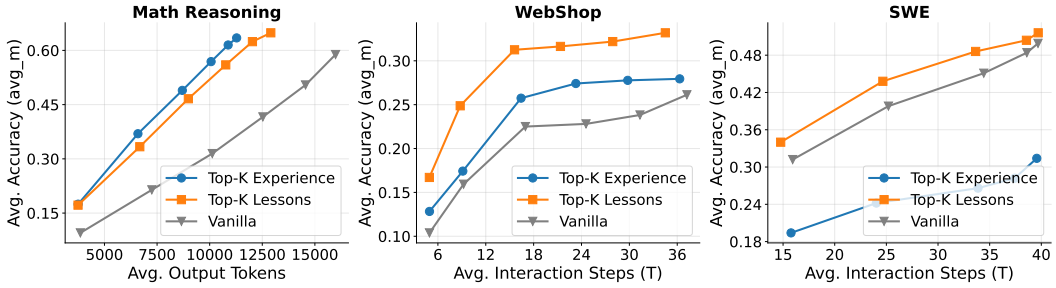


Figure 2: **Raw Experience vs. Distilled Lesson as Context.** Both context construction approaches significantly outperform the vanilla agent without context. Raw experience is slightly stronger for mathematical reasoning, while distilled lessons yield better performance in agentic tasks (WebShop & SWE), where trajectory-level observations are noisier and distillation helps.

Lesson Distillation Enables More Favorable Context Scaling. We further analyze the *input size scaling behavior* of the two approaches by increasing K in Top-K retrieval. Figure 3 reports performance as a function of input token length for both experience-based and lesson-based context construction. First, the lesson-based approach consistently achieves strong performance with a smaller input context size. For a fixed performance level, fewer input tokens are required compared to raw experience. This advantage is particularly pronounced in WebShop, where distilled lessons outperform raw trajectories. Second, as K increases, we observe that the performance of the experience-based approach can degrade, especially when the input context becomes excessively long. This suggests that overly detailed and noisy demonstrations could be misleading.

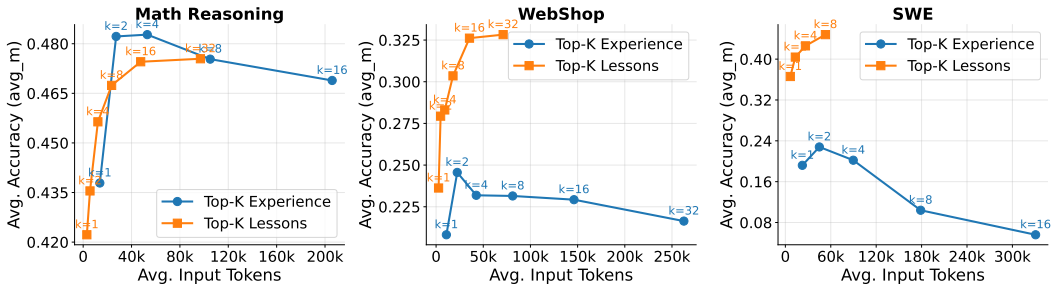


Figure 3: **Input Size Scaling Behavior.** Agent's performance as a function of input context length when increasing K in Top-K retrieval. Distilled lessons achieve better performance with fewer input tokens and remain more robust as context grows. In contrast, raw experience can degrade when the prompt becomes excessively long and noisy. Overall, lesson distillation acts as an effective context compression mechanism that extracts the essential information from experience.

3.2 Experience Size Scaling: How to Construct Effective Memory?

As the experience grows, memory size can also become arbitrarily large. A large memory can introduce redundancy: many entries may be similar and contain overlapping experience. Therefore, this motivates a memory consolidation mechanism, i.e., $\tilde{\mathcal{M}} := \mu(\mathcal{M})$.

Clustering-Based Memory Consolidation. We adopt a clustering-based consolidation strategy. Let each memory entry be associated with an embedding vector $e_i \in \mathbb{R}^d$, derived from the problem statement and the distilled lesson, $e_i = \text{Normalize}\big((1-\alpha)\,\pi_e(x_i) + \alpha\,\pi_e(\ell_i)\big)$, where $\alpha \in [0,1]$ controls the contribution of lesson-level semantics. We partition the memory into N' clusters using standard k-means. For each cluster, we select a single representative entry that is closest to the cluster centroid. The consolidated memory becomes $\tilde{\mathcal{M}} = \{(\tilde{x}_i, \tilde{\ell}_i)\}_{i=1}^{\tilde{N}}$, where $\tilde{N} < N$. We evaluate performance using $\tilde{\mathcal{M}}$ across different sizes.

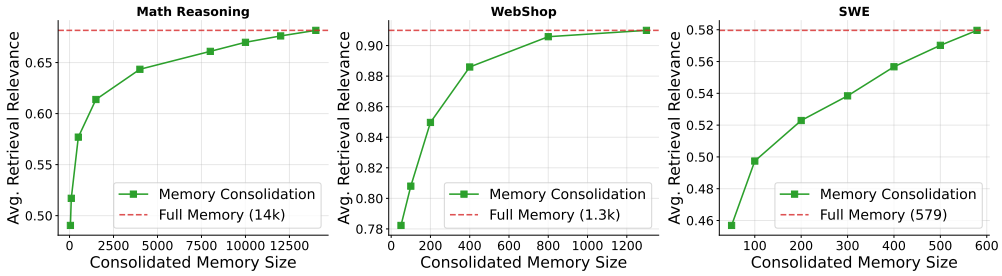
For this study, we consider a new metric *Avg. Retrieval Relevance*, i.e., the relevance score measured by embedding similarity between context retrieved from $\tilde{\mathcal{M}}$ and query sample. From Figure 4 (a), we observe that *retrieval relevance scores increase monotonically* with memory size, as expected, as memory consolidation is essentially a lossy compression.

Memory Consolidation Exhibits a Sweet Spot. Next, we further evaluate agent's performance w.r.t. the memory size in Figure 4 (b), our results reveal several key findings. First, consolidation exhibits a *non-monotonic trend*: when $\tilde{N}$ is too small, aggressive compression removes essential information, causing noticeable performance degradation. When $\tilde{N}$ approaches the full memory size, consolidation provides little benefit. Second, there exists a *sweet spot* at intermediate memory sizes where k-means consolidation achieves improved performance and efficiency compared to full memory. These results highlight a key insight: *higher retrieval relevance does not necessarily yield better task performance*. This reveals a fundamental *relevance–diversity tradeoff*: while semantic similarity helps retrieve related lessons, excessive similarity can reduce the diversity of problem-solving strategies and limit generalization. Overall, moderate memory consolidation improves retrieved context by reducing redundancy while preserving diversity, which is essential for experience-augmented agents.

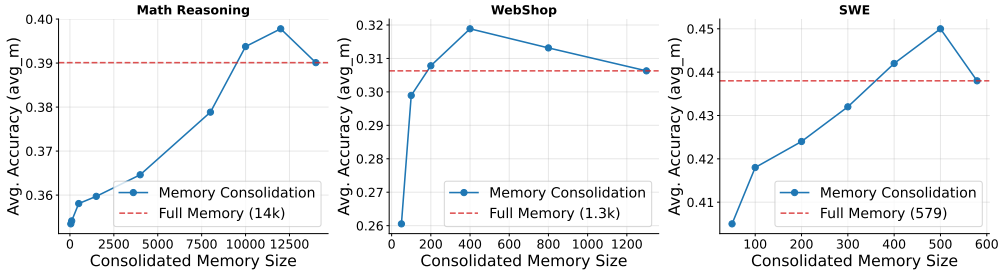
4 What Characterizes an Effective Context?

The previous section demonstrates that constructing context through decocted experience can effectively improve agent's test-time inference. An important question in practice is how we can construct *effective* context for a new task. To answer this question, in this section, we first provide an intuitive theoretical insight on the characterization of good context, and then propose an empirical measure that forecasts the agent's performance improvement.

Theoretical Analysis. In Figure 2, we demonstrate that more informative context (constructed from distilled lesson) may yield better performance with fewer output tokens (or interaction steps). One may naturally ask whether there is a direct theoretical connection



(a) Retrieval Relevance vs. Consolidated Memory Size



(b) Performance vs. Consolidated Memory Size

Figure 4: **Experience Scaling Behavior via Memory Consolidation.** We evaluate experience memory consolidation across varying memory sizes. The red dashed line indicates full memory baseline performance. Memory consolidation achieves a sweet spot at intermediate sizes.

between the informativeness of context and effectiveness of inference. Here, we provide a quantitative yet intuitive relationship between them via information measures.

Fix a query $x \in \mathcal{X}$. Let C denote the random variable corresponding to the context constructed for x, and let Y denote the random trajectory generated by the agent given (x, c). A natural quantity to capture the informativeness of a context c is the *information gain* of context c for query x defined as

$$I(Y; C = c \mid X = x) := H(Y \mid X = x) - H(Y \mid X = x, C = c).$$

We note that the average of the information gain over the randomness in the context corresponds to the *per-query mutual information* $I(Y; C \mid X = x) := \sum_{c \in \mathcal{C}} p(c|x) I(Y; C = c|X = x)$, which is always non-negative. However, the information gain for each context can take either positive or negative values: if $I(Y; C = c \mid X = x) \leq 0$, context fails to reduce uncertainty, indicating that the context may be misleading or noisy; $I(Y; C = c \mid X = x) > 0$ indicates that context is *effective* in that it reduces output uncertainty.

To connect the information gain with inference efficiency, let τ denote the stopping time such that the first EOS token is generated at time $\tau + 1$, such that $Y_{1:\tau}$ corresponds to the sequence of the non-EOS tokens.

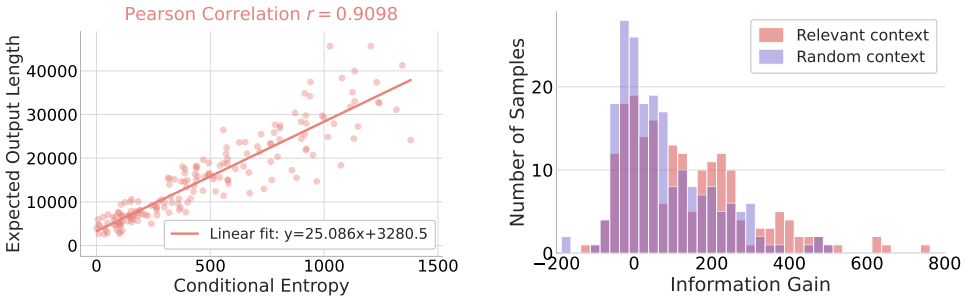
Proposition 4.1 (Context Efficiency Bound)**.** *Fix $X = x$ and $C = c$. Suppose there exists a constant $h > 0$ such that*

$$H\left(Y_{1:\tau} \mid \tau = t, X = x, C = c\right) \geq th, \quad \forall t \geq 0.$$

In words, each token carries at least h bits of uncertainty on average. Then,

$$\mathbb{E}[\tau \mid X = x, C = c] \leq \frac{H(Y \mid X = x, C = c)}{h} = \frac{H(Y \mid X = x) - I(Y; C = c \mid X = x)}{h}.$$

Given base uncertainty $H(Y \mid X = x)$ and the average entropy rate lower bound h, larger information gain $I(Y; C = c \mid X = x)$ leads to a strictly smaller upper bound on the expected trajectory length $\mathbb{E}[\tau \mid X = x, C = c]$. Intuitively, informative context would reduce uncertainty in the output trajectory, effectively contracting the search space that the



(a) Conditional Entropy vs. Expected Output Length.

(b) Distribution of Information Gain.

Figure 5: **Empirical validation of Proposition 4.1.** Figure **(a)** shows a strong linear correlation between the conditional entropy over correct trajectories, $\hat{H}^+(Y \mid X = x, C = c)$, and their expected output length. Figure **(b)** shows that relevant context (retrieved lessons) yields higher information gain on correct trajectories than random context.

model must explore during generation. As a result, the agent can reach solutions using fewer reasoning tokens or interaction steps.

Empirical Validation. We next empirically examine the relationship suggested by Proposition 4.1. While the proposition is stated for the general output trajectory Y, when assessing the usefulness of context in practice, we are primarily interested in successful inference. In particular, reducing uncertainty over arbitrary outputs does not necessarily indicate an effective context. We therefore examine relationship specifically over correct trajectories. Specifically, for each query x, let

$$\mathcal{Y}_x^+ := \{y \in \mathcal{Y} : r(x, y) = 1\}$$

denote the set of trajectories that produce the correct final answer, and define

$$H^+(Y \mid X = x, C = c) := H(Y \mid X = x, C = c, Y \in \mathcal{Y}_x^+).$$

We restrict the estimation to correct trajectories because reducing uncertainty over arbitrary outputs does not necessarily indicate an effective context. We compute a Monte Carlo estimator $\hat{H}^+(Y \mid x, c)$ using the token-level log-probabilities produced by the model, restricted to correct responses.

We first examine the predicted relationship between conditional entropy and output length. Figure 5(a) plots $\hat{H}^+(Y \mid X = x, C = c)$ against the average output length of correct responses across all evaluation samples from the math reasoning benchmarks. The two quantities exhibit a strong linear correlation ($r \approx 0.91$), showing that, among correct responses, lower conditional entropy is associated with shorter generated trajectories.

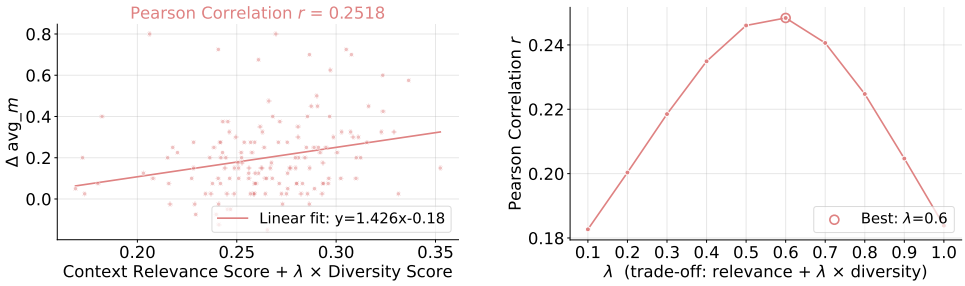
Next, we investigate whether the *information gain* on correct outputs captures the *quality* of the retrieved context. For each problem x, we estimate the empirical information gain

$$\hat{I}^+(Y; C = c \mid X = x) = \hat{H}^+(Y \mid X = x) - \hat{H}^+(Y \mid X = x, C = c),$$

where $\hat{H}^+(Y \mid X = x)$ corresponds to the entropy over correct responses under the no-context setting. Figure 5(b) compares the distribution of $\hat{I}^+(Y; C = c \mid X = x)$ under two conditions: *relevant context* (retrieved lessons) and *random context* (randomly sampled lessons). Relevant context produces substantially higher information gain (mean $\hat{I}^+ = 172.2$ vs. 97.2 for random context) and dominates on 76% of evaluation samples.

Together, these results provide an explanation for the context-scaling behavior observed in Section 3: relevant context tends to reduce uncertainty over correct trajectories more effectively than random context, and such reduced uncertainty is associated with more efficient successful inference.

What Makes a Good Context in Practice? The analysis above suggests that informative context can effectively reduce the length of the trajectory. In practice, however, the measure



(a) Correlation Between Context Quality & Performance Improvement

(b) Trade-off Between Relevance & Diversity

Figure 6: **Correlation between Context Quality and Performance Improvement.** (a) The relationship between the context quality score and the improvement measured in $\Delta\text{avg_}m(x)$. The positive Pearson correlation indicates that higher-quality contexts tend to yield larger performance gains. (b) Pearson correlation as the coefficient λ varies. The correlation peaks around $\lambda = 0.6$, suggesting that balancing relevance and diversity leads to the most informative contexts.

of information gain is not available before generation, so we need a proxy metric for context quality. Here, we hypothesize that effective context should balance two practically *measurable* properties, *relevance* and *diversity*, defined as follows. Recall that $R(x, \tilde{\mathcal{M}}) = \{z_{i_1}, \ldots, z_{i_K}\}$ denotes the set of K retrieved memory entries for a query x. For each $z \in R(x, \tilde{\mathcal{M}})$, let $\pi_e(z) \in \mathbb{R}^d$ denote its embedding representation. **(1) Relevance.** We define the relevance score as the average similarity between the query embedding and the embeddings of the retrieved entries: $\text{Rel}_{R,\tilde{\mathcal{M}}}(x) = \frac{1}{K}\sum_{z \in R(x,\tilde{\mathcal{M}})} \langle \pi_e(x), \pi_e(z) \rangle$. A higher score indicates that the retrieved entries are more semantically aligned with the query. **(2) Diversity.** To measure redundancy within the retrieved set, we compute the negative average pairwise similarity among retrieved entries: $\text{Div}_{R,\tilde{\mathcal{M}}}(x) = -\frac{1}{K(K-1)}\sum_{z \neq z' \in R(x,\tilde{\mathcal{M}})} \langle \pi_e(z), \pi_e(z') \rangle$. A higher score indicates that the retrieved set is less redundant and covers a broader range of concepts or strategies. We define a retrieval quality score by combining these two terms

$$Q_{R,\tilde{\mathcal{M}}}(x) = \text{Rel}_{R,\tilde{\mathcal{M}}}(x) + \lambda\, \text{Div}_{R,\tilde{\mathcal{M}}}(x),$$

where λ controls the relevance–diversity trade-off. For each query x, we also quantify the performance improvement induced by the context:

$$\Delta\text{avg_}m(x) = \text{avg_}m_{\text{context}}(x) - \text{avg_}m_{\text{vanilla}}(x),$$

where $\text{avg_}m_{\text{context}}(x)$ denotes the performance achieved using the context, and $\text{avg_}m_{\text{vanilla}}(x)$ denotes the performance of the same agent with no context.

To validate our hypothesis, we compute the Pearson correlation between the retrieval quality score $Q_{R,\tilde{\mathcal{M}}}(x)$ and $\Delta\text{avg_}m(x)$ across all evaluation samples. Figure 6(a) demonstrates a positive Pearson correlation ($r = 0.25$), indicating that retrieved sets that are both relevant to the query and internally diverse tend to yield larger performance gains. Figure 6(b) further examines the trade-off by varying λ. The correlation peaks around $\lambda = 0.6$, suggesting that the most useful retrieved contexts should balance this trade-off.

5 A Tree-Structured Memory for Experience-Augmented Agents

The analyses in Section 3 and Section 4 identify key properties of effective memory for context construction, and in particular reveal the limitations of a *flat* memory without structure. First, although memory consolidation (i.e., reducing the number of entries) can improve performance when applied at an appropriate level (Figure 4(b)), the optimal operating point is difficult to determine in advance. Second, Figure 6 shows that useful context should balance relevance and diversity, whereas standard similarity-based retrieval over a flat memory may fail to capture deeper conceptual relationships. Together, these observations

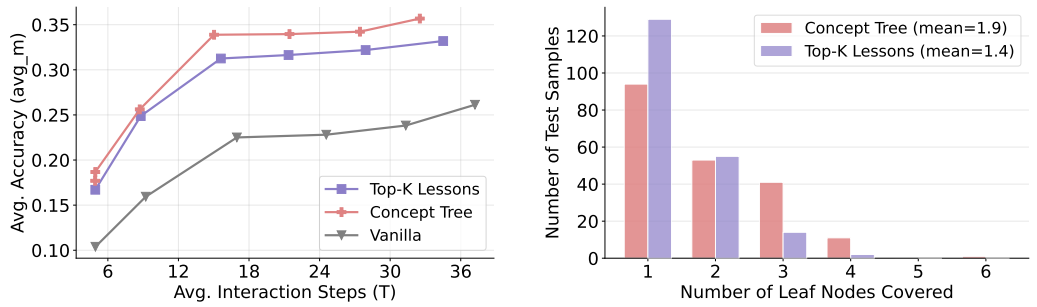
Figure 7: **Visualization of Concept Tree (WebShop).** Hierarchical concept tree constructed from 1,135 experience records on WebShop task. The tree organizes standard lesson-based memory entries into 8 broad topics and 44 concrete concept groups via hierarchical clustering. Each leaf node contains a collection of lessons, with leaf segment widths proportional to the number of records.

suggest that effective memory should (i) adaptively control retrieval granularity without requiring explicit consolidation and (ii) promote diversity beyond surface-level similarity. Motivated by this, we propose a structured memory called a *hierarchical concept tree*, tailored for experience-augmented agents. At a high level, the tree organizes memory entries via embedding-based clustering in a coarse-to-fine manner, enabling concept-level retrieval.

Hierarchical Concept Tree for Structured Memory. Let $\{z_i = (x_i, \ell_i)\}_{i=1}^N$ denote the set of memory entries, where ℓ_i is the distilled lesson for problem x_i. In the hierarchical concept tree, each leaf node corresponds to a subset of entries grouped by a shared concept, represented by a cluster center embedding. The tree is constructed offline in two stages; see the upper half of Algorithm 1 in Appendix. **(1) Concept Extraction.** For each entry $z_i = (x_i, \ell_i)$, we use the base LLM π_θ to extract a short, structured *concept description* $d_i := \pi_\theta(x_i, \ell_i)$, which abstracts the underlying topic, problem pattern, and solution technique. For example, a geometry problem may yield: "`Topic: angle chasing` ‖ `Problem: finding angles in cyclic quadrilaterals` ‖ `Technique: inscribed angle theorem with auxiliary lines`". We then embed d_i using an embedding model π_e, yielding $e_i := \pi_e(d_i)$. **(2) Hierarchical Clustering.** We construct an L-level hierarchical tree $\mathcal{T}$ over the embeddings $\{e_i\}_{i=1}^N$ using recursive bisecting k-means with cosine similarity. Each internal node splits its data into b children, where the branching factor is $b = \lceil (\lfloor N/s \rfloor)^{1/L} \rceil$, with s the target leaf size. A node becomes a leaf when depth L is reached or its size falls below a threshold. Each leaf λ_j stores a subset of entries, representing a coherent concept group.

Concept Tree Retrieval with LLM Re-ranking. At test time, retrieval operates at the leaf level (Algorithm 1, lower half). Given a query x with embedding $e_x = \pi_e(x)$, we compute a *leaf affinity score* $a_j(x)$ for each leaf λ_j as the cosine similarity between e_x and the leaf centroid. We then select the top n_ℓ leaves and merge their entries into a candidate pool Top-$K_{\text{cand}}\{z_i\}_{i=1}^N$. This pool is much smaller than the full memory and remains *structurally diversified*, since different leaves correspond to different conceptual groups. Finally, we prompt π_θ with the query x and the candidate problem statements $\{x_i\}$, ask it to identify the most useful lessons, and format the selected entries into the final context.

Results. We evaluate whether this more advanced memory structure can further improve experience-augmented agents. Specifically, we construct a two-level concept tree ($L = 2$) and visualize it on WebShop in Figure 7. The first level captures broad shopping categories, while the leaf level corresponds to finer-grained subcategories, reflecting the intuition that similar products often require similar shopping strategies. We compare this agent equipped with the concept tree with a flat lesson memory $\tilde{\mathcal{M}} = \{(x_i, \ell_i)\}_{i=1}^N$ using semantic retrieval. Figure 8(a) shows that the concept tree leads to improved inference performance. Figure 8(b) shows that concept-tree-based agent tends to select lessons from a more diverse set of concept groups (the number of the concepts covered by the retrieved context) compared to the semantic-based retrieval baseline. Together with the analysis in Section 4, it suggests the improved performance is due to a better balance between relevance and diversity.



(a) Performance Comparison *(b)* Concept Group Diversity

Figure 8: **Hierarchical Concept Tree vs. Lesson Retrieval** (a) Performance comparing the hierarchical concept tree against Top-K lesson retrieval and vanilla baselines. (b) Concept group diversity comparison: concept tree achieves greater leaf-node coverage than Top-K lesson retrieval.

6 Concluding Remarks

Across reasoning and agentic tasks, our analysis suggest that decocted experience serves as the key ingredient for effective context construction that improves the test-time inference in the LLM agent. Several directions remain for future work. One promising direction is to develop more advanced mechanisms for leveraging experience, including more sophisticated memory structures and closed-loop systems that iteratively collect new experience and update model weights to improve performance over time. Another important direction is continual learning in an agentic system: given a new task or environment, an agent should be able to accumulate new experience, integrate it with existing knowledge, and continually improve without forgetting previously acquired capabilities.

Acknowledgment

This research/project was supported, in part, by the following funding resources: MIT-IBM Watson AI Lab under Agreement No. W1771646, DEVCOM Army Research Laboratory under award # W911NF1720196, and the National Science Foundation under award #2502536. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 15

ByteDance-Seed. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads. https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME, 2025a. 17

ByteDance-Seed. Seed-oss open-source models. https://github.com/ByteDance-Seed/seed-oss, 2025b. 3

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, et al. Scaling agent learning via experience synthesis. *arXiv preprint arXiv:2511.03773*, 2025. 16

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. 15

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, Haofen Wang, et al. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1):32, 2023. 15

Qishuo Hua, Lyumanshan Ye, Dayuan Fu, Yang Xiao, Xiaojie Cai, Yunze Wu, Jifan Lin, Junfei Wang, and Pengfei Liu. Context engineering 2.0: The context of context engineering. *arXiv preprint arXiv:2510.26493*, 2025. 1

Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo J Taylor, and Dan Roth. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale. *arXiv preprint arXiv:2504.14225*, 2025a. 15

Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, et al. Personamem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory. *arXiv preprint arXiv:2512.06688*, 2025b. 15

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. 3, 17

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020. 15

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, Junpeng Ren, Zehao Lin, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhiqiang Yin, Qingchen Yu, Bo Tang, Hongkang Yang, Zhi-Qin John Xu, and Feiyu Xiong. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*, 2025. 15

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Bodhisattwa Prasad Majumder, Katherine Hermann, Amir Yazdanbakhsh, Peter Clark, Hannaneh Hajishirzi, Yejin Choi, Noah A. Smith, Iz Beltagy, and Faeze Brahman. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. 15

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024. 15

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025. 1, 15

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925. 3

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025. 2, 15, 16

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. 15

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation. In *Proceedings of the ACM on Web Conference 2025*, pp. 2366–2377, 2025. 15

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. *arXiv preprint arXiv:2410.14052*, 2024. 15

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 1, 2024. 15

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, et al. Thinking vs. doing: Agents that reason by scaling test-time interaction. *arXiv preprint arXiv:2506.07976*, 2025. 1

Taiwei Shi, Sihao Chen, Bowen Jiang, Linxin Song, Longqi Yang, and Jieyu Zhao. Experiential reinforcement learning. *arXiv preprint arXiv:2602.13949*, 2026. 16

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. 15

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations*, 2025. 1

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025. 2, 16

Yu Wang and Xi Chen. MIRIX: Multi-agent memory system for LLM-based agents. *arXiv preprint arXiv:2507.07957*, 2025. 15

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025. 15

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. 15

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 15

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025. 2, 15, 16

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. In *Advances in Neural Information Processing Systems*, 2024. 1

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments, 2024. 17

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025. 15

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024. 17

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018. 15

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022. 3, 17

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025a. 15

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025b. 3, 17

Guangtao Zeng, Maohao Shen, Delin Chen, Zhenting Qi, Subhro Das, Dan Gutfreund, David Cox, Gregory Wornell, Wei Lu, Zhang-Wei Hong, et al. Satori-swe: Evolutionary test-time scaling for sample-efficient software engineering. *arXiv preprint arXiv:2505.23604*, 2025. 1

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025a. 15

Kai Zhang, Xiangchao Chen, Bo Liu, et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025b. 16

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025c. 15

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025d. 3, 18

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. 16

Appendix

A Related Work **15**

B Proof of Proposition 4.1 **16**

C Experimental Setup **17**
C.1 Datasets 17
C.2 Implementation Details 17
C.3 Prompt Templates 19

D Additional Experiment Results **22**
D.1 Omitted Results in Section 5 22
D.2 Additional Analysis. 24

A Related Work

Context Engineering. Context engineering can be viewed as optimizing the information fed into an LLM at inference time through retrieval, generation, processing, and management, rather than changing the model parameters (Mei et al., 2025). Early context engineering methods typically relied on manually designed prompts for downstream tasks, including few-shot prompting (Brown et al., 2020) and prompt engineering (Sahoo et al., 2024). Chain-of-thought (CoT) prompting (Wei et al., 2022) further shows that multi-step reasoning can substantially improve performance on reasoning tasks. Beyond hand-crafted prompts, self-generated refinement (Madaan et al., 2023; Shinn et al., 2023) treats the model's own reflection as an evolving context, improving generation across a range of tasks. Effective context can also be constructed from external knowledge via retrieval, such as retrieval-augmented generation (RAG) (Lewis et al., 2020; Gao et al., 2023; Qian et al., 2025), which injects relevant knowledge into the prompt for downstream question answering tasks. However, in agentic settings, context is not merely evidence for answering a question, it should also encode reusable strategies and executable workflows. A growing body of recent work (Zhang et al., 2025c; Wang et al., 2024; Ouyang et al., 2025; Wei et al., 2025) argues that agents benefit from dynamically evolving context that contains strategies and workflows, rather than from naive context summarization.

Agent Memory Systems. Transformer-based LLM agents remain constrained by a finite context window, which motivates external memory systems that organize past information while allowing the agent to retrieve relevant content efficiently at test time. Early memory architectures are primarily inference-time systems. MemGPT (Packer et al., 2023) introduces an OS-inspired hierarchy that manages information across working, recall, and archival memory, while MemOS (Li et al., 2025) builds a unified system for representation, organization, and governance across multiple memory forms. Mem0 (Chhikara et al., 2025) focuses on scalable deployment by extracting and consolidating salient facts from multi-session interactions with graph-based memory. MemTree (Rezazadeh et al., 2024) further explores structured memory by organizing information into a tree representation, enabling hierarchical retrieval and integration beyond flat memory stores. MIRIX (Wang & Chen, 2025) builds a multi-agent controller that routes across specialized memory types such as episodic, semantic, and procedural memory. A more recent line of work seeks to train LLMs to *learn* memory management. MemGen (Zhang et al., 2025a) interleaves reasoning with generated latent memory, whereas Mem-α (Wang et al., 2025), MemAgent (Yu et al., 2025a), and Memory-R1 (Yan et al., 2025) explicitly optimize memory construction or memory operations with reinforcement learning. Despite proposing a variety of memory architectures and management strategies, most of the existing literature still centers on conversational recall (Maharana et al., 2024), user preference persistence (Jiang et al., 2025a;b), or long-context question answering (Yang et al., 2018), where the central challenge is remembering factual information rather than reusing problem-solving strategies. In contrast, our work treats memory not merely as a storage-and-recall mechanism, but as a source of *decision-support context* for future reasoning and acting. Accordingly, our focus is not only on how to store

and retrieve information, but on how experience should be distilled, consolidated, and organized so that the retrieved context is most strategically useful at test time.

Leveraging Past Experience for Agents. An emerging research direction studies how agents self-improve from prior interaction experience, either at test time or through weight updates during training. An early representative test-time approach is ExpeL (Zhao et al., 2024), which collects trajectories on training tasks, abstracts them into insights, and retrieves both insights and successful past experiences at test time, showing that agents can improve from accumulated experience without gradient updates. More recent test-time approaches further develop this idea in various forms. Dynamic Cheatsheet (Suzgun et al., 2025) equips LLMs with an adaptive memory of reusable strategies and problem-solving insights extracted from past attempts. ReasoningBank (Ouyang et al., 2025) distills lessons from both successful and failed trajectories into reasoning memory and couples the agent with memory-aware test-time scaling. Under a similar setting, Evo-Memory (Wei et al., 2025) provides a streaming benchmark that evaluates whether agents can retrieve, adapt, and evolve memory using accumulated interactions. In parallel, training-time approaches such as Early Experience (Zhang et al., 2025b) propose reward-free supervision derived from an agent's own experience, where observed future states serve as supervision. Closely related, DreamGym (Chen et al., 2025) proposes scaling agent learning via *experience synthesis*: instead of relying on expensive real-environment rollouts, it trains agents with a reasoning-based experience model that generates synthetic transitions, feedback, and progressively more challenging tasks for online RL. ExRL (Shi et al., 2026) embeds an explicit experience–reflection–consolidation loop inside RL training: a first attempt receives feedback, self-reflection guides a second attempt, and successful improvements are distilled into the base policy to encourage self-improvement. Our work is complementary to this line of research, but focuses on a different question: rather than proposing yet another framework for leveraging experience, we ask what makes past experience actually useful for future inference. Our key insight is that experience becomes effective not simply by being stored, but by being *decocted* into context: extracting the essence from experience, organizing it coherently, and retrieving salient information to to balance relevance with diversity, thereby revealing the principles that make experience-augmented agent effective and scalable.

B Proof of Proposition 4.1

Proof. Consider

$$\begin{aligned} H(Y \mid X = x, C = c) &\overset{(a)}{\geq} \mathbb{E}_{p(\tau|x,c)}\Big[H(Y \mid \tau, X = x, C = c)\Big] \\ &\overset{(b)}{\geq} \mathbb{E}_{p(\tau|x,c)}\Big[H(Y_{1:\tau} \mid \tau, X = x, C = c)\Big] \\ &\overset{(c)}{\geq} h\mathbb{E}_{p(\tau|x,c)}[\tau]. \end{aligned} \tag{1}$$

Here, (a) and (b) follow since conditioning and dropping variables reduce entropy, respectively; (c) follows from the lower-bound assumption on the conditional entropy.

Dividing both sides by $h > 0$, we have

$$\begin{aligned} \mathbb{E}_{p(\tau|x,c)}[\tau] = \mathbb{E}[\tau \mid X = x, C = c] &\leq \frac{H(Y \mid X = x, C = c)}{h} \\ &= \frac{H(Y \mid X = x) - I(Y; C = c \mid X = x)}{h}, \end{aligned}$$

where the last equality follows from the definition of information gain. □

C Experimental Setup

C.1 Datasets

Mathematical Reasoning. We use the DAPO-Math dataset (Yu et al., 2025b) to construct the experience memory. The training set contains 14,116 competition-level mathematical reasoning problems spanning a diverse set of topics, including algebra, geometry, number theory, combinatorics, and probability. For evaluation, we consider six challenging math reasoning benchmarks: AMC 2023, AIME 2024, AIME 2025, HMMT 2024, HMMT 2025, and BeyondAIME (ByteDance-Seed, 2025a), for a total of 260 test problems. These benchmarks are drawn from highly competitive high-school and university-level mathematics contests. The problems are intentionally difficult and require substantially deeper reasoning than standard school-level math benchmarks. Since both the training set and the evaluation benchmarks provide ground-truth final answers, we can automatically verify model outputs and assign binary feedback $r \in \{0, 1\}$ based on whether the generated final answer is correct.

WebShop Agent. WebShop (Yao et al., 2022) is a large-scale interactive web environment that simulates online shopping. In this environment, an agent is given a natural-language instruction describing a target product and must navigate an e-commerce website to identify, configure, and purchase an item that satisfies the user's request. The environment is built from more than 1.18 million real-world Amazon products and 12,087 crowd-sourced shopping instructions, and each episode unfolds over multiple webpage types, including search, results, item, and item-detail pages. At each step, the agent observes the current webpage and can take high-level actions such as issuing a search query, selecting a product from the results list, choosing product options, opening detailed descriptions or overviews, navigating between pages, and finally clicking `Buy` to terminate the episode. Following the WebShop setup in AgentGym (Xi et al., 2024), we use 3,930 training trajectories to construct memory and 200 test episodes for evaluation. Performance is measured by a scalar reward $r \in [0, 1]$ assigned to the final purchased item, which reflects how well it matches the user instruction in terms of product type, attributes, options, and price.

Software Engineering Agent. We consider software engineering tasks based on SWE-bench (Jimenez et al., 2023), a benchmark constructed from real GitHub issues and their corresponding pull requests across 12 popular Python repositories. Each instance provides an issue description and a pre-fix snapshot of the repository, and the agent must generate code edits that resolve the issue in the existing codebase. We adopt the SWE-agent framework (Yang et al., 2024), in which the agent operates in a command-line environment with access to standard bash commands as well as specialized tools for repository navigation, file viewing, content search, and code editing. This setting requires the agent to localize the relevant files, understand the repository structure, implement the fix, and interact with the execution environment to check whether the modification is successful. Following the SWE-bench evaluation protocol, an instance is counted as resolved (i.e., $r \in \{0, 1\}$) only if the generated patch applies successfully and all associated fail-to-pass and pass-to-pass tests succeed, ensuring both issue resolution and preservation of existing functionality. For memory construction, we use the SWE-bench full set excluding SWE-bench Verified, yielding 1,794 training instances, and we evaluate on SWE-bench Verified, a human-validated test set of 500 instances designed for more reliable assessment.

C.2 Implementation Details

Raw Memory Construction. The memory stores the agent's past experience from interactions on the training set. For each training sample, we let the LLM generate $m = 4$ random trajectories using a sampling temperature of 1.0 and an unlimited thinking budget. We then retain only those samples for which the feedback signals across the four attempts are not all zero, ensuring that each memory entry contains at least one positive experience. The raw memory is stored as a dictionary containing the problem statement, the raw experience, and the corresponding feedback signals.

Lesson Distillation. The lesson distillation procedure introduced in Section 3.1 uses the agent itself to extract a reusable lesson from raw experience, i.e., a problem-solving strategy distilled from the $m = 4$ sampled attempts. For successful attempts, the agent is prompted to summarize key insights and common reasoning patterns. For failed attempts, it is prompted to reflect on common mistakes and reasoning flaws that should be avoided in future problems. We use greedy decoding for lesson extraction.

Memory Consolidation. The memory consolidation procedure introduced in Section 3.2 applies standard k-means clustering in the embedding space. The hyperparameter for combining the problem-statement embedding and the lesson embedding is set to $\alpha = 0.5$. Consolidation is applied after lesson distillation, starting from the lesson-based memory $\tilde{\mathcal{M}} = \{(x_i, \ell_i)\}_{i=1}^{N}$, with the goal of compressing it into a smaller memory with $\tilde{N}$ entries. Specifically, the N memory entries are partitioned into $\tilde{N}$ clusters, and only the entry closest to each cluster centroid is retained, while the remaining cluster members are removed.

Context Retrieval. Our default retrieval method is based on semantic similarity. Specifically, we use Qwen3-Embedding-4B (Zhang et al., 2025d) to encode the problem statement of each memory entry into a 2,560-dimensional embedding vector. Given a query, the retriever selects the top-K memory entries with the highest embedding similarity to the query problem. By default, we set $K = 8$ for math reasoning, $K = 4$ for WebShop, and $K = 4$ for SWE. The final context is constructed from the query problem statement, the problem statements of the retrieved memory entries, and their corresponding experiences or lessons.

Inference and Evaluation. The constructed context is fed into the LLM for inference. During evaluation, we set the sampling temperature to 1.0 and generate $m = 8$ random outputs. We evaluate the agent under a fixed test-time budget. For standard reasoning tasks such as math reasoning, we impose a maximum output-token budget, and a problem is counted as solved only if the model produces the correct answer within this token limit. For agentic tasks, we instead define the test-time budget as the maximum number of interaction steps allowed between the agent and the environment. We do not impose an additional token budget within each interaction step, since in agentic settings the primary cost typically arises from repeated rounds of action and feedback, rather than from a long reasoning trace within a single step. The metrics introduced in Section 2.2 are computed by averaging over the $m = 8$ sampled outputs.

Concept Tree. The concept tree introduced in Section 5 is built on top of the lesson-based memory $\{z_i = (x_i, \ell_i)\}_{i=1}^{N}$. For each memory entry, we first perform *concept description extraction* by prompting the base LLM to produce a short structured description. We use greedy decoding in this stage. These concept descriptions are then encoded into embeddings and used to construct the tree via recursive bisecting k-means clustering. For all tasks, we use a two-level hierarchy ($L = 2$). The target leaf size is set to 50 for mathematical reasoning, 20 for WebShop, and 10 for SWE, reflecting the different memory sizes in each domain.

At test time, retrieval is performed at the leaf level. Given a query, we first rank leaves by their affinity to the query embedding and select the top $n_\ell = 5$ leaves. We then pool the memory entries contained in these leaves to form a candidate set for LLM re-ranking. In practice, we set a maximum candidate pool size $K_{\text{cand}} = 300$ so that it is typically within 300 entries, which keeps the re-ranking stage within the LLM context window while still preserving sufficient diversity. Finally, the LLM is prompted with the query and the candidate problem statements, and it is allowed to select a variable number of relevant lessons up to a predefined cap. This design gives the model flexibility to adapt the amount of retrieved context to the different query problems, rather than forcing a fixed number of retrieved lessons for every instance.

C.3 Prompt Templates

Prompt Template: Lesson Distillation (Math Reasoning)

You are a problem-solving strategy synthesizer analyzing problem-solving attempts. Carefully review the problem, and your attempted solutions (can be correct or incorrect). Your goal is to extract a detailed problem-solving strategy from a long reasoning process that can help solve this type of problems.

Problem Statement:
{problem_statement}

Ground Truth Solution:
{gt_solution}

Previous Attempts:
{formatted_rollouts}

Reasoning Instructions:
Extract key insights and reasoning patterns from successful attempts, aggregating different ways to solve the problem.
Reflect Common mistakes, misunderstandings, or reasoning flaws that could lead to incorrect answers (if applicable).
Analyze preconditions and scenarios where the identified strategies are most applicable.
Response Format:
1. Task Description: A one-two sentence summary of the type of problems this strategy applies to.
2. Strategy: A step-by-step detailed problem-solving strategy that could consist of various different ways to tackle similar problems.
3. Pitfalls: Common mistakes or misconceptions to avoid when solving this type of problems (if applicable).

Important:
The strategy should be extremely detailed, covering multiple different ways to solve the problem.

Prompt Template: Lesson Distillation (WebShop)

You are a shopping strategy synthesizer analyzing past web shopping attempts. Review the shopping task and the previous interaction trajectories (can be successful, partially successful, or unsuccessful based on the reward). Your goal is to extract a detailed reusable shopping strategy from a long reasoning process that can help solve this type of problems.

Shopping Task:
{task_instruction}

Past Shopping Trajectories:
{formatted_trajectories}

Instructions:
Extract key insights and reasoning patterns from successful attempts (reward = 1.0).
Reflect Common mistakes, misunderstandings, or reasoning flaws that could lead to sub-optimal attempts (reward < 1.0)
Analyze preconditions and scenarios where the identified strategies are most applicable

Response Format:
1. Task Description: A one-two sentence summary of the type of shopping task this strategy applies to.
2. Action Workflow: A step-by-step detailed problem-solving strategy to complete the purchase, including step-level actions and the summarization of the observations at each step.
3. Pitfalls: Common mistakes or pitfalls to avoid when solving this type of shopping task (if applicable).

Important:
The workflow should include clear action steps (e.g., CLICK, TYPE, SELECT) and observations at each step.

Prompt Template: Lesson Distillation (SWE)

You are a debugging strategy synthesizer analyzing software engineering problem-solving attempts. Review the GitHub issue description and the previous resolution trajectories below (can be successful or unsuccessful based on the result). Your goal is to distill a detailed reusable debugging and fixing strategy from a long reasoning process that can help solve this type of problems.

GitHub Issue:
{issue_text}

Past Resolution Trajectories:
{formatted_trajectories}

Reasoning Instructions:
Extract key insights and reasoning patterns from successful attempts (resolved = True).
Reflect on common mistakes or reasoning flaws from failed attempts (resolved = False).
Focus on actionable details: concrete file paths, class/function names, grep patterns, test commands, and verification steps.

Response Format:
1. Task Description: One-two sentence summary of the issue type and code base area.
2. Strategy: Step-by-step detailed agent workflow structured as an action sequence
(followed by successful attempts).
3. Pitfalls: Common mistakes to avoid. If a failed attempt is available, explain the specific wrong turn and how to avoid it.

Important:
The workflow should include clear action steps and observations at each step.

Prompt Template: Experience-based Inference

You are an expert at solving complex reasoning problems, while leveraging few-shot experience. You have been given several examples from past experience that may help solve the new target problem. Please solve the problem in an efficient manner by leveraging the past experience.

Reasoning Instructions:
{task_dependent_instruction}

Few-shot Experience:
{examples}

Target Problem Statement:
{problem_statement}

Prompt Template: Concept Description Extraction (Math Reasoning)

You are a mathematical concept analyst. Given a problem statement and a lesson/solution, produce a concise concept description that captures the underlying mathematical topic, the core problem pattern, and the key technique(s) used.

Problem:
{problem_statement}

Lesson / Solution:
{lesson}

Respond with EXACTLY three lines (no extra text):
Topic: [1-4 word topic, e.g. “number theory”, “dynamic programming”]

Problem: [1 sentence summarizing the problem pattern]
Technique: [1 sentence summarizing the solution technique]

Prompt Template: Concept Description Extraction (WebShop)

You are a shopping task analyst. Given a task instruction and a lesson from an agent's shopping experience, produce a concise concept description that captures the shopping category, the core task pattern, and the key strategy used.

Task Instruction:
{problem_statement}

Lesson:
{lesson}

Respond with EXACTLY three lines (no extra text):
Topic: [1-4 word shopping/product category, e.g. "clothing, woman", "electronics, laptop"]
Problem: [1 sentence summarizing the shopping task]
Technique: [1 sentence summarizing the shopping strategy and agentic workflow]

Prompt Template: Concept Description Extraction (SWE)

You are a software engineering issue analyst. Given a GitHub issue description and a debugging lesson/solution, produce a concise concept description that captures the repository/library area, the core bug pattern, and the key fix technique used.

Issue:
{problem_statement}

Lesson / Solution:
{lesson}

Respond with EXACTLY three lines (no extra text):
Topic: [1-4 word topic, e.g. "Django ORM query", "NumPy indexing", "pytest fixture"]
Problem: [1 sentence summarizing the bug pattern]
Technique: [1 sentence summarizing the fix approach]

Prompt Template: LLM Re-ranking

You are an expert relevance judge for problem-solving experience retrieval.

Given a **query problem** and a numbered list of **candidate experience entries** (each described by its problem statement), select the most relevant candidates based on problem similarity (the same type of problem that shares similar solution strategies).

Query Problem:
{query_problem_statement}

Candidate Experiences:
{formatted_candidates}

Selection Guidelines:
- You may select as many candidates as you think are relevant, but no more than max_k.
- Order your selections from most relevant to least relevant.
- If for some rare case you find that none of any candidate is relevant, select top_k candidates that could be helpful to the query problem.

Return ONLY a JSON array of integer indices (0-based), Example: [4, 12, 7, 0, 3, 18, 24, 1, 9, 53]
Your response MUST end with the JSON integer array on its own line with no surrounding text or labels. Output (JSON integer array only, on the last line):

D Additional Experiment Results

D.1 Omitted Results in Section 5

Algorithm 1 Hierarchical Concept Tree

```
// Offline: Build Concept Tree
```

Input: Query-lesson pairs $\{(x_i, \ell_i)\}_{i=1}^{N}$ tree depth L; target leaf size s; number of selected leaves n_ℓ; candidate pool size K_{cand}.
Output: Hierarchical concept tree $\mathcal{T}$.
for $i = 1$ **to** N **do**
  $d_i \leftarrow \pi_\theta(x_i, \ell_i)$; // extract concept description
  $e_i \leftarrow \pi_e(d_i)$; // embed concept description
end
$\mathcal{T} \leftarrow \text{BISECTINGKMEANS}\big(\{e_i\}_{i=1}^{N},\ L,\ s\big)$; // build L-level concept tree

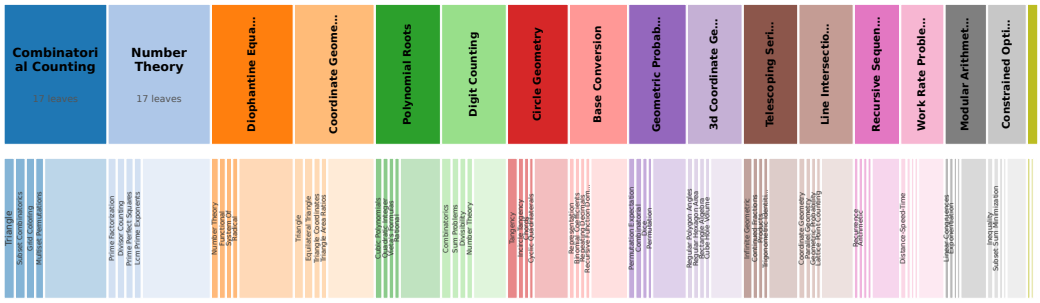
```
// Online: Retrieve & Re-rank
```

Input: Tree $\mathcal{T}$; test query x.
Output: Context c for query x.
$e_x \leftarrow \pi_e(x)$; // embed query
for *each leaf* λ_j *in* $\mathcal{T}$ **do**
  $a_j(x) \leftarrow \text{Affinity}(e_x,\ \lambda_j)$; // compute cosine similarity to leaf nodes
end
$\Lambda^* \leftarrow \text{Top-n}_{\ell_{\lambda_j}}\ a_j(x)$; // select top-n_ℓ leaves
$\mathcal{C}_{\text{cand}} \leftarrow \text{Top-K}_{\text{cand}}\big\{z_i : i \in \bigcup_{\lambda_j \in \Lambda^*} \text{members}(\lambda_j)\big\}$; // pool members of selected leaves
$\mathcal{R}(x) \leftarrow \pi_\theta\big(x,\ \{x_i : z_i \in \mathcal{C}_{\text{cand}}\}\big)$; // LLM re-ranking
$c \leftarrow \text{Concat}\big(x,\ \{(x_i, \ell_i)\}_{i:z_i \in \mathcal{R}(x)}\big)$
return c

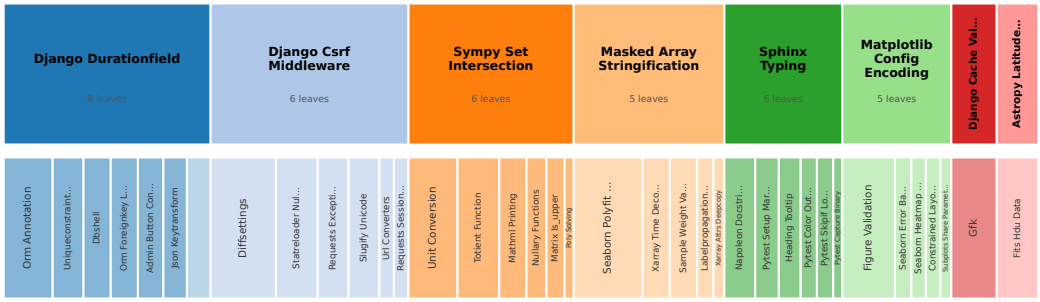
Algorithm of Concept Tree Construction and Retrieval. We first provide the detailed algorithm of the hierarchical concept tree in Algorithm 1. At a high level, the method consists of an offline construction stage and an online retrieval stage. Offline, each lesson memory entry (x_i, ℓ_i) is converted into a short concept description by the base LLM, and these concept descriptions are embedded and organized into a multi-level tree via recursive bisecting k-means clustering. Online, given a test query, we first score leaf nodes by their affinity to the query embedding, then select the top-ranked leaves to form a candidate pool, and finally use the LLM to re-rank candidate entries before constructing the final context. This procedure allows retrieval to operate at the level of concept groups rather than individual memory entries, which provides a more structured way to control retrieval relevance and promote diversity beyond flat similarity-based retrieval.

Concept Tree Visualization. Figures 9(a) and 9(b) visualize the resulting concept trees on the math reasoning and SWE tasks, respectively. On math reasoning, the root-level nodes correspond to broad mathematical topics such as combinatorics, number theory, geometry, and algebra, while lower-level leaves further specialize into finer-grained problem types and strategies, such as modular arithmetic, linear congruences, cyclic quadrilaterals, or triangle area ratios. On SWE, the tree exhibits a different but equally interpretable structure: high-level nodes often correspond to software libraries, subsystems, or bug families, while leaf nodes capture more concrete repair patterns, such as Django middleware issues, SymPy set operations, Matplotlib configuration bugs, or PyTest setup failures.

Results on Math Reasoning and SWE Tasks. Figure 10 further compares the hierarchical concept tree with the top-K lesson retrieval baseline on the math reasoning and SWE tasks. On math reasoning, the concept tree does not yield a clear improvement over flat lesson retrieval. One possible explanation is that math problems and their distilled lessons are already relatively clean, diverse, and semantically focused, so standard embedding-based

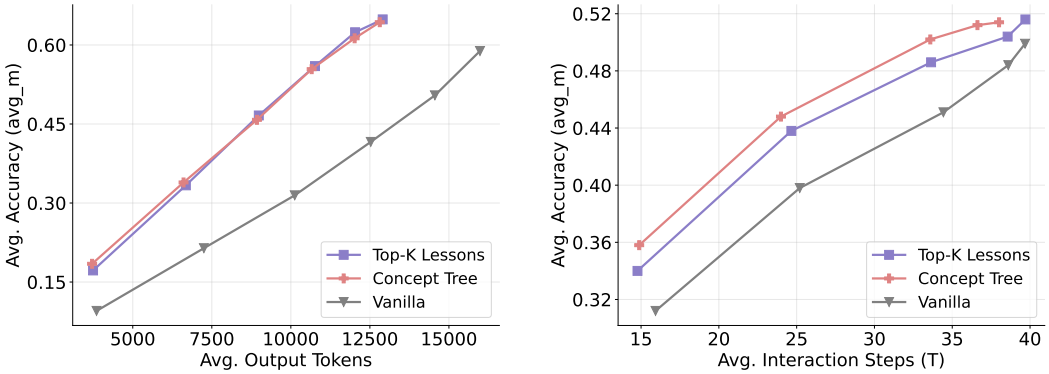


(a) Math Reasoning



(b) SWE

Figure 9: **Visualization of Concept Tree (Math Reasoning & SWE).** (a) Hierarchical concept tree constructed from 13,381 experience records on the math reasoning task. The tree organizes memories into 17 broad topics and 256 concrete concept groups via two levels of embedding-based clustering, with segment widths proportional to the number of records in each cluster. For visual clarity, only the four largest leaf clusters per root category are shown individually, and the remaining leaves are merged. (b) Hierarchical concept tree constructed from 579 experience records on the SWE task. The tree organizes memories into 8 broad topics and 38 concrete concept groups.



(a) Math Reasoning *(b)* SWE

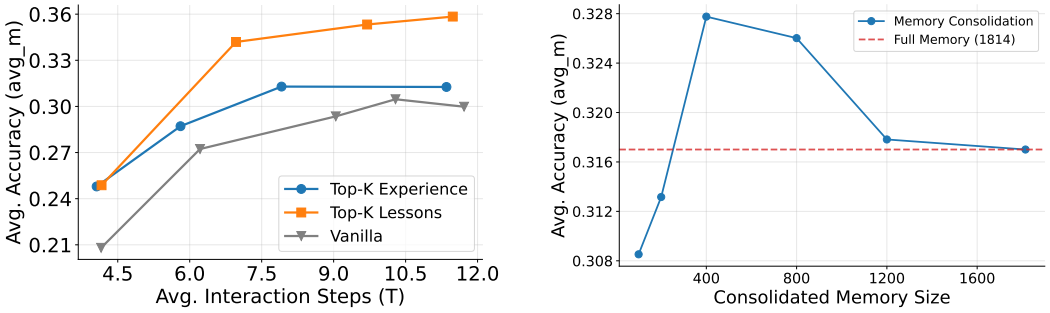
Figure 10: **Hierarchical Concept Tree vs. Lesson Retrieval** Performance comparing the hierarchical concept tree against Top-K lesson retrieval and vanilla baselines.

retrieval over a flat lesson memory is often sufficient to identify useful examples. In contrast, on SWE, the concept tree still delivers a noticeable improvement over the baseline, consistent with our observation on the WebShop task (Figure 8). We hypothesize that this is because experience from agentic tasks often shares deeper conceptual structure, such as reusable

workflows, debugging patterns, or web navigation strategies, that may not be fully captured by surface-level semantic similarity alone.

D.2 Additional Analysis.

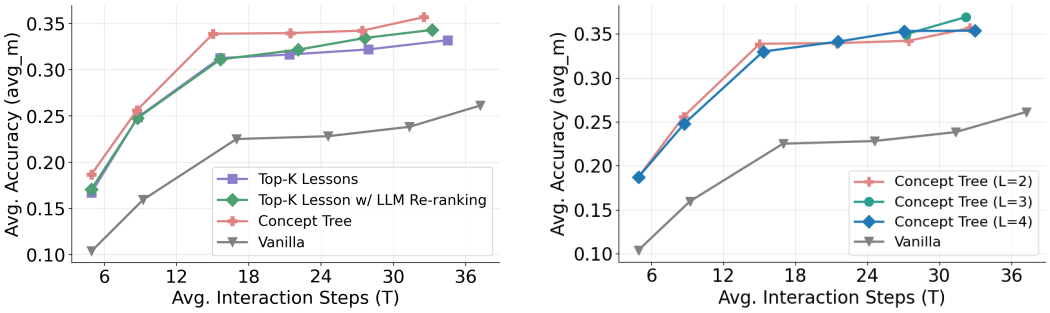
Analysis on Other Base LLMs. We further conduct the analysis in Section 3 with GPT-OSS-20B as the base model, using WebShop as a representative task for illustration. The overall observations remain consistent with the main results: lesson distillation continues to outperform raw experience on the agentic task, and memory consolidation still exhibits a non-monotonic scaling behavior with a sweet spot at intermediate memory sizes. These results suggest that the main conclusions of Section 3 are not specific to Seed-OSS-36B-Instruct, but instead reflect a more general property of experience-augmented agent. In particular, they further support our central claim that effective *experience decoction*, through distilling noisy raw trajectories into reusable lessons and organizing accumulated memory appropriately, is key to improving the scalability and effectiveness of test-time context construction.



(a) Raw Experience vs. Distilled Lesson

(b) Experience Scaling Behavior via Memory Consolidation

Figure 11: **Analysis Results on GPT-OSS-20B (WebShop).** The overall trends are consistent with the main results in Section 3. **(a)** Distilled lessons provide more effective context than raw experience. **(b)** Memory consolidation shows a sweet spot at intermediate memory sizes.



(a) Ablation of LLM Re-ranking

(b) Concept Tree Depth (L)

Figure 12: **Ablation Studies of the Concept Tree. (a)** Comparison with a direct LLM re-ranking baseline shows that the structured memory contributes more substantially to the performance improvement. **(b)** Performance remains relatively stable across tree depths $L \in \{2, 3, 4\}$, indicating limited sensitivity to the choice of depth.

Ablation Studies of the Concept Tree. In this section, we present ablation studies of the concept-tree method. In Figure 12(a), we first examine whether its advantage primarily arises from the structured memory or from the LLM re-ranking stage. Specifically, on

WebShop, we compare against a baseline that applies LLM re-ranking directly without using the tree structure. Although this baseline slightly outperforms Top-K lesson retrieval, it still under-performs the full concept-tree method. This result suggests that the structured memory contributes more substantially to the overall performance improvement.

Next, we ablate the tree depth L by evaluating $L \in \{2, 3, 4\}$. Figure 12(b) shows that performance is relatively insensitive to the choice of depth. One possible explanation is that retrieval is performed at the leaf-node level; therefore, the intermediate hierarchy mainly affects how memory entries are partitioned, rather than directly determining the final retrieved context. Consequently, increasing the tree depth does not necessarily lead to monotonic performance improvements.