Learning How to Remember: A Meta-Cognitive Management Method for Structured and Transferable Agent Memory

Anonymous ACL submission

Abstract

Large language model (LLM) agents increasingly rely on accumulated memory to solve long-horizon decision-making tasks. However, most existing approaches store memory in fixed representations and reuse it at a single or implicit level of abstraction, which limits generalization and often leads to negative transfer when distribution shift. This paper proposes the **M**eta-**C**ognitive **M**emory **A**bstraction method (**MCMA**), which treats memory abstraction as a learnable cognitive skill rather than a fixed design choice. MCMA decouples task execution from memory management by combining a frozen task model with a learned memory copilot. The memory copilot is trained using direct preference optimization, it determines how memories should be structured, abstracted, and reused. Memories are further organized into a hierarchy of abstraction levels, enabling selective reuse based on task similarity. When no memory is transferable, MCMA transfers the ability to abstract and manage memory by transferring the memory copilot. Experiments on ALFWorld, ScienceWorld, and BabyAI demonstrate substantial improvements in performance, out-of-distribution generalization, and cross-task transfer over several baselines.

1 Introduction

Large language model (LLM) agents have recently demonstrated strong performance beyond static question answering (Achiam et al., 2023; Bai et al., 2023; Chen et al., 2025b; Tao et al., 2024), increasingly operating in long-horizon, interactive environments that require sustained decision making and environment feedback (Qiao et al., 2024; Tan et al., 2025; Zhang et al., 2025b; Chhikara et al., 2025). In these settings, a key capability of agents is to accumulate, organize, and reuse memory, which is also known as **procedural memory** (Fang et al., 2025; Cao et al., 2025). Effectively reusing accumulated memories is essential for enabling agents to solve new tasks efficiently and operate robustly in complex, long-horizon environments (Song et al., 2024a).

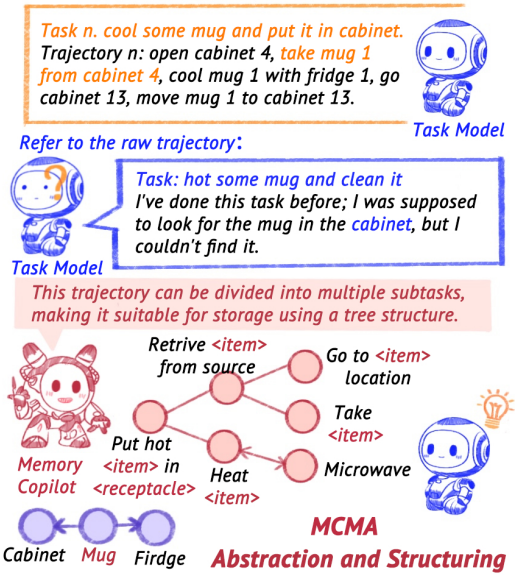

Figure 1: An example of how MCMA works.

Despite the growing use of memory in LLM-based agents, effective memory reuse remains challenging. Retrieval-based approaches recall trajectories (Zheng et al., 2023), sub-tasks (Kim et al., 2024), or fine-grained execution steps (Zhou et al., 2024), but often rely on surface similarity and fail under environment or task changes (Wang et al., 2024b). Summarization (Zhao et al., 2024), abstraction (Wang et al., 2024b), and hierarchical methods (Ye et al., 2025) mitigate noise by organizing memory at higher levels, yet still face an abstraction reuse dilemma. We believe this phenomenon is because fine-grained representations overfit to the seen environments, while overly abstract ones lack actionable guidance. Training-based approaches (Song et al., 2024a; Zeng et al., 2024; Wang et al., 2024a) further internalize experience into model parameters to improve generaliza-

tion, but tightly entangle memory with policy learning, limiting cross-task transfer and risking catastrophic forgetting under domain shifts (Chen et al., 2025a). Overall, the aforementioned methods rely on predefined memory representations and fixed or implicit abstraction levels. As a result, agents fail to learn reusable abstractions or adaptively select appropriate levels of memory abstraction for unseen tasks.

To address these challenges, this paper proposes the **M**eta-**C**ognitive **M**emory **A**bstraction method (**MCMA**), which treats memory abstraction as a learnable cognitive skill, rather than a static design choice. MCMA operates at a cognitive level by learning how memories should be represented, abstracted, and reused, instead of obtaining fixed or predefined abstract memory itself. As shown in Figure 1, MCMA is built around a ***Memory Copilot*** that operates at a meta level, learning to regulate the structure and granularity of memory and managing past successes and failures into reusable abstract memories. To enable this abstraction process to be learned and transferred independently of task-specific behaviors, MCMA decouples memory management from task execution by employing a frozen ***Task Model*** solely for action selection.

MCMA follows a four-stage pipeline: collecting trajectories, generating structured memories, training abstraction strategies, and reusing knowledge and ability. The memory copilot is trained via direct preference optimization (DPO) (Rafailov et al., 2023) to learn **multi-structural memory representations** (e.g., tree, chain, or natural language structures) and, crucially, **abstraction strategies that support memory reuse**. Guided by cognitive theories of memory abstraction, memory is commonly divided into *episodic memory*, which stores concrete, context-dependent experiences, and *semantic memory*, which encodes abstract, organized knowledge (Tulving, 1972). MCMA organizes structured memories into **multiple abstraction levels**, where lower levels retain fine-grained execution details, while higher levels save abstracted knowledge. When no prior memory is reusable, **the memory copilot itself is transferred**, preserving the learned abstraction capability. Experiments on ALFWorld ($\uparrow$ 25.07%), ScienceWorld ($\uparrow$ 7.92%), and BabyAI demonstrate substantial gains in robustness, out-of-distribution generalization, and cross-task transfer.

In summary, our contributions are as follows:

- This paper proposes a meta-cognitive memory abstraction method **MCMA**. By learning structural abstract memory representations and organizing reusable hierarchical abstractions, MCMA transforms memories from inflexible storage into a transferable resource.
- MCMA introduces memory copilot, trained via DPO to distill trajectories into structured abstract knowledge that is more suitable for assisting unseen tasks. Crucially, the memory copilot itself can be transferred across domains, reusing the learned ability to abstract, reflect, and organize new tasks.
- Extensive experiments on ALFWorld, ScienceWorld, and BabyAI, demonstrating substantial improvements in basic performance, out-of-distribution generalization, and cross-task transfer.

2 Related Work

Memory as Content Retrieval or Execution Guidance. Prior work on memory-augmented LLM agents primarily focuses on storing and retrieving past experience to guide future decision making. Retrieval-based methods, including trajectory-level retrieval (Zheng et al., 2023; Kong et al., 2025; Ritchford, 2025; Luo et al., 2025), subtask or case-based reuse (Kim et al., 2024), and step-wise retrieval aligned with intermediate reasoning states (Zhou et al., 2024). While related efforts extract procedural patterns or heuristics from execution histories for planning. These approaches treat memory as reusable content with fixed representations and retrieval granularities. Consequently, retrieved experience often overfits to surface-level similarity and can mislead agents under changes in environment or task, leading to a negative impact.

Hierarchical, Reflective, and Learned Experience Beyond direct retrieval, several works explore hierarchical organization, reflection, or learning from experience to improve agent robustness. Hierarchical and control-based approaches introduce multiple memory levels or regulate memory access to separate high-level planning from low-level execution (Ye et al., 2025; Peiyuan et al., 2024; Zhao et al., 2024; Wang et al., 2024c; Zhang et al., 2025a; Xu et al., 2025), while reflection-based methods refine future behavior through error

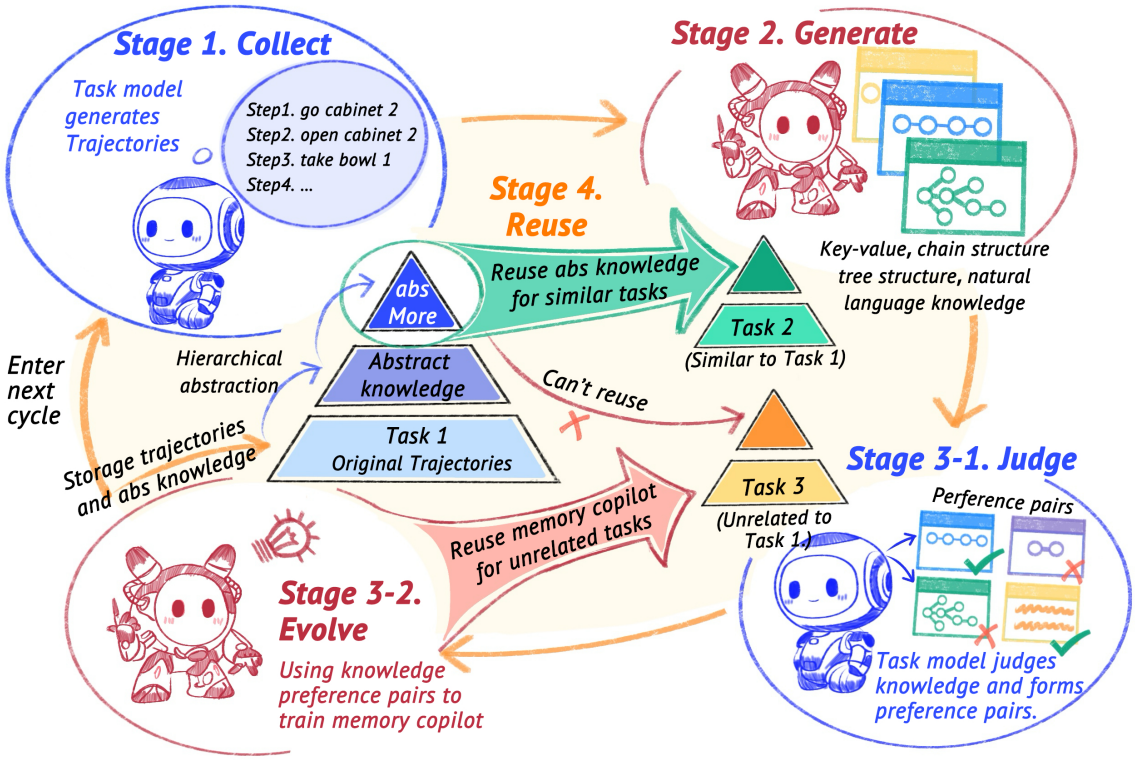

Figure 2: The task model collects raw trajectories, which the memory copilot abstracts into structured knowledge. Preference pairs constructed from downstream task performance are used to train the memory copilot via DPO. Learned memories and the copilot are organized hierarchically to support adaptive memory reuse across tasks.

correction (Song et al., 2024b; Wang et al., 2025a; Fu et al., 2024; Shinn et al., 2023). Large-scale training efforts further internalize experience by adjusting model parameters to improve generalization (Song et al., 2024a; Zeng et al., 2024; Wang et al., 2025b; Yao et al., 2023; Yin et al., 2024). Despite these advances, memory representations and abstraction levels remain largely predefined or implicitly fixed, leaving agents without the ability to learn how experience should be abstracted or which abstraction level is fit for a new task.

3 Methodology

As shown in Figure 2, MCMA consists of two functionally distinct models: a ***Task Model*** and a ***Memory Copilot***. *Stage 1*, the task model collects interaction trajectories by performing the task. *Stage 2*, the memory copilot transforms raw interaction trajectories into structured and reusable memories through selective structure combination and abstraction. *Stage 3*, the task model evaluates these memories based on downstream task performance, constructs preference pairs to train the memory copilot, enabling its continual evolution. *Stage 4*, MCMA organizes the accumulated memories into a hierarchy of abstractions and supports the reuse of either stored knowledge or the memory copilot itself. For example, in ALFWorld, once a subtask is completed, the memory copilot abstracts the trajectory, and when the agent later encounters a similar task, an appropriate level of abstraction is selected for retrieval.

3.1 Trajectory Collection and Preprocessing

The task model collects both successful and failed trajectories from training environments. Raw trajectories are typically long, noisy, and dominated by low-level execution details, making them inefficient for direct reuse. Given a trajectory, execution traces are converted into goal oriented representations via trajectory simplification: **1) Noise pruning**: remove redundant or task-irrelevant steps. **2) Subtask segmentation**: partition the remaining sequence into coherent subtask units based on changes in the goal (*e.g.*, task *"Put a bowl in the dining table"* can be divided into subtasks: *"Take a bowl from the cabinet"*, and *"Place a bowl on the dining table"*.). **3) Consistency checking**: Verify logical dependencies and temporal ordering between subtasks. **4) Structured processing**: Organize subtasks into a tree, with high-level goals as parent nodes and executable skills as leaves.

This process preserves both high-level intent and low-level executability while significantly reducing representation complexity. Appendix A.3 provides a detailed preprocessing procedure and examples.

3.2 Multi-Structure Memory Generation

The memory copilot learns a **meta-level abstraction policy** that jointly determines the *structural composition* and *level* of memory representations. Since memory representation critically affects reuse and transfer across tasks, we avoid assuming a fixed memory storage structure. Instead, for each trajectory τ, the copilot generates a set of candidate *composite structured memories*:

$$\mathcal{M}_\tau = \{m_\tau^1, m_\tau^2, \ldots, m_\tau^N\}. \tag{1}$$

Each candidate m_τ^i is constructed by composing and nesting multiple structural primitives drawn from a predefined set:

$$\mathcal{S} = \{\text{natural text, key–value, chain, tree}\}, \tag{2}$$

rather than selecting a single structure type. Appendix A.4 provides several structural knowledge examples. Intuitively, different structure capture complementary aspects of experience: tree and chain structures encode high-level dependencies, while key-value pairs and natural language retain fine-grained details. Their composition enables expressive yet reusable memory representations. Each composite structured memory is evaluated by its downstream utility $s(m_\tau^i)$ (Equation 6), and the probability of generating a particular composition is defined as:

$$P_\theta(m_\tau^i \mid \tau) = \frac{\exp(\beta\, s(m_\tau^i))}{\sum_{j=1}^{N} \exp(\beta\, s(m_\tau^j))}, \tag{3}$$

where θ denotes the copilot parameters and $\beta > 0$ is a temperature parameter. This distribution is used to optimize memory selection toward representations with higher expected downstream utility.

To control abstraction granularity, we introduce a continuous abstraction parameter $\alpha \in [0, 1]$:

$$m_\tau(\alpha) = \textit{Memory Copilot}_\theta^{\text{abs}}(\tau, \alpha). \tag{4}$$

Smaller α preserves execution details, while larger α promotes higher procedure level abstractions. The memory copilot should learn an optimal α, which maximizes downstream utility via training:

$$\alpha^* = \arg\max_{\alpha \in [0,1]} s\big(m_\tau(\alpha)\big). \tag{5}$$

We collect a diverse set of multi-structure memories with varying abstraction levels to support subsequent memory copilot evolution.

3.3 Memory Copilot Evolution

For each trajectory, multiple knowledge candidates are judged on downstream tasks, producing scores:

$$s(m_i) = \begin{cases} 0, & \text{task fails,} \\ 0.1 + 0.9 \cdot \dfrac{T_{\max} - T_i}{T_{\max} - T_{\min}}, & \text{otherwise,} \end{cases} \tag{6}$$

where T_i denotes execution length. The $Top\text{-}K$ preference pairs with the largest differences in $s(m_i)$ are used for training. This strategy introduces strong supervision signals while preserving diversity among memory representations. We adopt a two-stage procedure consisting of supervised fine-tuning followed by direct preference optimization (DPO) (Rafailov et al., 2023). For a preference pair (m^+, m^-) derived from the same trajectory τ, the DPO objective is:

$$\mathcal{L} = -\log \sigma\Big(\beta\big[\log p_\theta(m^+) - \log p_\theta(m^-)\big]\Big) \tag{7}$$

where $p_\theta(m)$ denotes the likelihood assigned by the memory copilot parameterized by θ to generating a structured memory abstraction m conditioned on trajectory τ. Minimizing this objective encourages the model to assign higher probability to memory representations that lead to more successful and efficient downstream task execution. Success and failure trajectories are processed separately, resulting in two memory copilots with different functions (successful memory summarization and failure memory reflection). During inference, $Top\text{-}N$ relevant trajectories are retrieved using character-level matching on task descriptions, then abstracted by the copilot before being provided to the task model.

3.4 Hierarchical Abstraction and Cross-Task Reusing

After learning instance-level memory representations, memories are organized into a hierarchy:

$$\mathcal{H} = \{H_0, H_1, \ldots, H_L\}, \tag{8}$$

where lower levels store detailed memories, H_0 retaining raw trajectories, low levels (*e.g.*, H_1) encoding episodic memories with execution details, and higher levels (*e.g.*, H_L) capturing abstract knowledge such as scripts ($L = 2$ in our work). We perform similarity clustering based on the vector representation of the task description. High-level abstractions are obtained by merging lower-level

representations while suppressing task-specific details to retain shared goals. For a new task τ_{new}, memory reuse is performed via:

$$m_{\text{reuse}} = \arg \max_{m \in H_\ell, \ell \in [0,L]} \text{sim}(\tau_{\text{new}}, m). \quad (9)$$

High-similarity tasks are supported by low-level, detailed episodic memories, whereas low-similarity tasks use higher-level abstract memories. Abstracted knowledge is incorporated into the prompt to assist the model when solving similar tasks.

Crucially, under extreme distribution shifts where no stored memory is directly reusable, MCMA transfers the memory copilot itself rather than specific trajectories. Let $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{new}}$ denote the training and novel task distributions. The memory copilot learns a transferable abstraction strategy θ^* by optimizing

$$\theta^* = \arg \max_\theta \mathbb{E}_{\tau \sim \mathcal{T}_{\text{train}}} \left[s(\text{Abstraction}_\theta(\tau, \alpha)) \right]. \quad (10)$$

By training, θ^* can be directly applied to new tasks $\tau_{\text{new}} \sim \mathcal{T}_{\text{new}}$ to generate abstract memories, enabling the model to generalize and transfer effectively even in the absence of relevant stored trajectories.

In this way, the memory copilot learns not only what to remember but how experience should be abstracted to maximize its future utility, enabling robust transfer across tasks and domains.

4 Experiment

4.1 Experimental Setup

Datasets. We evaluate MCMA on two long-horizon text-based embodied reasoning benchmarks: ALFWorld (Shridhar et al., 2020b) and ScienceWorld (Wang et al., 2022). ALFWorld evaluates household task completion from textual observations and provides *seen* and *unseen* splits. The *seen* split contains the same task types, objects, and room categories as training but varies object configurations. The *unseen* task instances are executed in rooms that never appear during training. ScienceWorld focuses on multi-step scientific reasoning under complex text-based dynamics and follows a standard *Dev / Test* split. Both benchmarks emphasize long-horizon reasoning and generalization beyond memorizing surface-level execution patterns, requiring effective abstraction, experience reuse, and adaptation to unseen environments.

Metrics. For ALFWorld, we follow the standard evaluation method and report results on both seen and unseen task splits. Performance is measured by task success rate (Acc%) and average execution steps, where higher accuracy and fewer steps indicate better performance. For ScienceWorld, we evaluate on the Dev and Test splits and report the average task reward score, which reflects the progress of sub-goals.

Models. We conduct experiments using two backbone task models: Qwen3-8B (Yang et al., 2025) and Qwen3-32B. The memory copilot uniformly uses Qwen3-4B. In all settings, the task model is frozen during training and evaluation, and all memory-related learning is isolated within the memory copilot.

Baselines. We compare MCMA against several memory configurations: *1) No Memory*, where the agent operates purely reactively without access to prior experience. *2) ReAct* (Yao et al., 2022), adopt a process of fully observing the environment and thinking before making a decision. *3) Raw Trajectory*, which retrieves and injects raw trajectories into memory. *4) TRAD* (Zhou et al., 2024), retrieves relevant expert steps via thought matching and aligning them with localized temporal context for each steps. *5) ExpeL* (Zhao et al., 2024), which extracts high-level natural language insights from past experiences and leverages retrieved successful trajectories as in-context demonstrations during inference. For the ALFWorld task, MCMA uses both the successful memory summary and the failed memory reflection memory copilot. Considering that the ScienceWorld environment is more variable, and summarizing successful memories often has negative effects, this task only used the failed memory reflection memory copilot.

4.2 Main Results

As shown in Table 1, MCMA consistently achieves the strongest performance across all environments and model scales. Compared to all baselines, MCMA significantly improves task success rates while reducing execution steps, indicating more efficient and reliable long-horizon planning.

Consistent gains across seen and unseen tasks. MCMA delivers large and stable improvements over no-memory and prior memory-based baselines in both in-domain and out-of-distribution settings. For example, on ALFWorld with Qwen3-8B, MCMA improves success rates by nearly 25% over

Model	Method	ALFWorld				Science World	
		Seen		Unseen		Dev	Test
		Acc(%) ↑	Step ↓	Acc(%) ↑	Step ↓	Reward ↑	Reward ↑
Qwen3-8B	No Memory	55.00	33.15	56.33	33.06	21.26	19.18
	ReAct	59.29 ↑4.29	31.04 ↓2.11	64.93 ↑8.60	30.58 ↓2.48	21.00 ↓0.26	18.41 ↓0.77
	Raw Tra	67.14 ↑12.14	24.75 ↓8.40	72.39 ↑16.06	25.01 ↓8.05	20.17 ↓1.09	17.61 ↓1.57
	TRAD	64.29 ↑9.29	28.48 ↓4.67	63.43 ↑7.10	29.76 ↓3.30	30.16 ↑8.90	27.42 ↑8.24
	ExpeL	69.28 ↑14.28	24.73 ↓8.42	72.39 ↑16.06	24.16 ↓8.90	26.43 ↑5.17	23.12 ↑3.94
	MCMA*(ours)*	**79.29** ↑24.29	**21.24** ↓11.91	**80.60** ↑24.27	**20.57** ↓12.49	**31.17** ↑9.91	**29.17** ↑9.99
Qwen3-32B	No Memory	62.86	29.85	66.42	30.36	45.10	43.69
	ReAct	63.57 ↑0.71	31.15 ↑1.30	72.39 ↑5.97	28.75 ↓1.61	45.44 ↑0.34	44.05 ↑0.36
	Raw Tra	74.29 ↑11.43	21.92 ↓7.93	78.36 ↑11.94	21.39 ↓8.97	44.82 ↓0.28	41.70 ↓1.99
	TRAD	74.29 ↑11.43	24.77 ↓5.08	79.10 ↑12.68	24.73 ↓5.63	42.77 ↓2.33	42.00 ↓1.69
	ExpeL	78.57 ↑15.71	20.40 ↓9.45	77.61 ↑11.19	21.23 ↓9.13	44.97 ↓0.13	43.78 ↑0.09
	MCMA*(ours)*	**90.71** ↑27.85	**17.78** ↓12.07	**90.30** ↑23.88	**18.00** ↓12.36	**51.95** ↑6.85	**48.60** ↑4.91

Table 1: Performance comparison of the Qwen3 model. We highlight the best and second best results. Green arrows (↑/↓) indicate performance improvement, and Red arrows (↑/↓) indicate decline.

Model	Seen	Unseen
GPT-4o-mini	42.86	48.51
MCMA*(GPT)*	**63.57** ↑20.71	**63.43** ↑14.92
Gemini-2.5-flash	61.43	67.91
MCMA*(Gemini)*	**84.29** ↑22.86	**88.06** ↑20.15

Table 2: Reuse memory copilot of Qwen3-32B on *Gemini-2.5-flash* and *GPT-4o-mini* on ALFWorld task.

the no-memory baseline, with comparable gains on unseen tasks. MCMA maintains similar improvements across seen and unseen scenarios, highlighting the strong generalization of its meta-cognitive memory abstraction.

Improved efficiency and scalability across models. MCMA consistently reduces the average number of execution steps across benchmarks, producing efficient and concise action sequences. These benefits are particularly pronounced for smaller models: on ScienceWorld, MCMA yields nearly a 10% absolute gain on Qwen3-8B, while still providing consistent improvements on Qwen3-32B. This indicates that memory abstraction effectively assists capacity limited model and remains powerful even as model scale increases. We further test the performance of the memory copilot trained for Qwen3-32B on the closed-source model Gemini-2.5-Flash and GPT-4o-mini. As shown in Table 2, the performance improvement is close to that achieved on Qwen3-32B, indicating that MCMA has good transferability and performs well on closed-source models.

4.3 Ablation Study

Component Ablations. To analyze the contribution of individual components in the memory copilot, we conduct ablation studies on ALFWorld, focusing on successful memory summarization (*Sum*) and failure memory reflection (*Ref*). As shown in Table 3, each component independently improves performance: summarization reduces execution steps by abstracting procedural knowledge, while reflection helps avoid recurring errors. Combining both components achieves the best performance across all settings and model scales, yielding higher success rates with fewer steps, particularly under distribution shift. These results suggest that summarization and reflection capture complementary aspects of experience *"learning what to do and what to avoid"*, and are jointly critical for robust generalization.

Memory Representation and Training Ablations. We further examine the impact of memory representation and training by comparing MCMA with variants that store experiences as natural language or chain-structured knowledge generated by DPO-trained memory copilots. As shown in Table 3 (*Nautral Language Know* and *Chain Know*), both representations yield substantial gains over the no-memory baseline (average accuracy

Model	Configuration	Seen		Unseen	
		Acc(%) ↑	Step ↓	Acc(%) ↑	Step ↓
Qwen3-8B	**Natural Language Know**	74.63 ↓4.66	23.37 ↑2.13	72.39 ↓8.21	24.88 ↑4.31
	Chain Know	77.14 ↓2.15	22.57 ↑1.33	75.27 ↓5.23	23.55 ↑2.98
	MCMA *(Sum)*	68.57 ↓10.72	23.69 ↑2.45	72.39 ↓8.21	23.85 ↑3.28
	MCMA *(Ref)*	72.86 ↓6.43	25.15 ↑3.91	71.64 ↓8.96	25.82 ↑5.25
	MCMA *(Sum+Ref)*	**79.29**	**21.24**	**80.60**	**20.57**
Qwen3-32B	**Natural Language Know**	77.86 ↓12.85	20.50 ↑2.72	82.09 ↓8.21	21.75 ↑3.75
	Chain Know	86.14 ↓4.57	19.10 ↑1.32	84.33 ↓5.97	20.24 ↑2.24
	MCMA *(Sum)*	76.43 ↓14.28	21.84 ↑4.06	83.25 ↓7.05	23.97 ↑5.97
	MCMA *(Ref)*	82.14 ↓8.57	22.46 ↑4.68	84.33 ↓5.97	21.95 ↑3.95
	MCMA *(Sum+Ref)*	**90.71**	**17.78**	**90.30**	**18.00**

Table 3: Component and Memory Representation ablation study of Qwen3-32B settings on ALFWorld .

Variant	Acc(%) ↑	Step ↓
MCMA$_{(Base)}$	83.58 ↓7.13	22.53 ↑4.75
MCMA$_{(Gemini)}$	86.56 ↓4.15	19.75 ↑1.97
MCMA$_{(DPO)}$	**90.71**	**17.78**

Table 4: Comparison with MCMA using untrained Qwen3-4B / Gemini-2.5-flash as memory copilot.

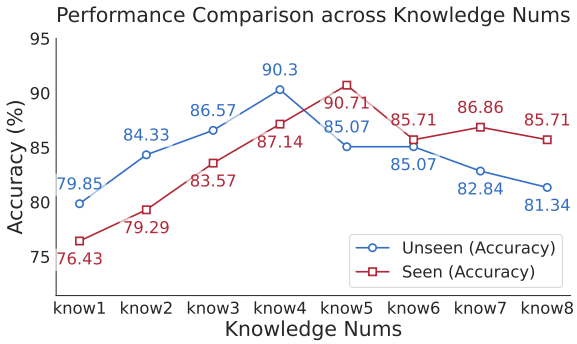

Figure 3: The impact of the number of knowledge provided by MCMA in the ALFWorld task.

64.64%), suggesting that the proposed abstraction strategy is relatively insensitive to specific surface structures. Nevertheless, a consistent performance gap remains compared to the full MCMA (MCMA$_{(Sum+Ref)}$), highlighting the importance of structured organization for effective memory utilization. In addition, when deploying untrained memory copilots Qwen3-4B or more powerful *Gemini-2.5-flash* in unseen settings, performance drops 7.13% and 4.15% (Table 4), indicating that DPO training is essential for equipping the memory copilot with transferable abstraction and generalization capabilities.

4.4 Structured Memory Analyze

The quantity of provided knowledge significantly impacts the model's performance. As illustrated in Figure 3, MCMA achieves optimal results on ALFWorld when 4–5 structural knowledge are provided. Insufficient knowledge fails to equip the task model with enough foresight to adapt diverse scenarios, while excessive knowledge imposes a heavy contextual overhead that disturbs decision-making. Notably, MCMA's structured knowledge is significantly more concise than original traces, it allows for a higher density of knowledge items within the prompt.

In addition, we count all nested structural knowledge. At the top-level, the predominant structures are Tree and Chain (Specifically, Tree structures comprised approximately 70% of these top-level arrangements, while Chain structures accounted for about 20%). This hierarchical organization effectively provides high-level directional guidance to the task model. Furthermore, within both Tree and Chain structures, a wide variety of sub-structures, such as Key-Value pairs and Natural Language descriptions, are nested, enabling a more fine-grained and explicit representation of detailed information. The overall distribution of all knowledge structure types is comprehensively illustrated in Figure 4.a. We further identify tasks characterized by Tree and Chain top-level structure and replace the knowledge representations with other structural formats. As shown in Figure 4.b, this led to a decline in performance, which underscores the effectiveness of the structure selection strategy studied by MCMA.

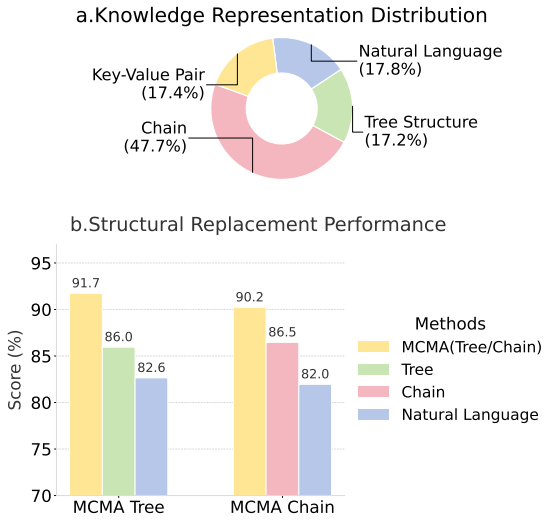

Figure 4: Analyze knowledge structure of MCMA.

4.5 Cross-Task Knowledge Transfer

Previous sections have shown that MCMA generalizes effectively within the same task distribution. This section extends our investigation to evaluate how MCMA performs in entirely novel environments.

To evaluate the effectiveness of hierarchical knowledge transfer under distribution shift, we extend our evaluation to the BabyAI benchmark (Chevalier-Boisvert et al., 2018). BabyAI requires agents to interpret and execute compositional, synthetic natural language instructions in a partially observable 2D gridworld, involving navigation, object manipulation, and environment interaction. Although BabyAI shares high-level similarities with ALFWorld in terms of spatial reasoning and object-centric interaction, it introduces a substantial domain shift, particularly in observation modalities (2D grid states vs. textual descriptions) and action granularity.

Variant	Acc(%) ↑	Reward ↑
Base	16.67	14.24
Abstract Level 1	13.54 ↓3.13	12.32 ↓1.92
Abstract Level 2	**17.71** ↑1.04	**15.33** ↑1.09

Table 5: Apply ALFWorld knowledge on BabyAI.

As shown in Table 5, transfer performance is strongly correlated with abstraction granularity. Fine-grained Level 1 knowledge leads to a 3.13% performance drop, indicating negative transfer caused by mismatched low-level interaction details. In contrast, higher-level Level 2 abstractions improve the baseline by 1.04%, suggesting that abstract knowledge better suppresses domain-specific noise and preserves reusable task semantics. These results demonstrate that higher-level abstractions are essential for effective generalization across distantly related tasks.

4.6 Cross-Domain Copilot Transfer

Method	ScienceWorld	ALFWorld
Base	19.18	56.33
Sci Copilot	29.17 ↑9.99	61.19 ↑4.86
ALF Copilot	27.43 ↑8.25	**71.64** ↑15.31
Mix Copilot	**29.85** ↑10.67	69.40 ↑13.07

Table 6: Copilots transfer across domains.

We further explore a more challenging scenario: *how to reuse memory when a new task has no correlation with existing memories*. MCMA achieves this by transferring the memory copilot, which reuses the capability to organize and abstract knowledge rather than the knowledge itself. We evaluate this via a cross-task transfer between ALFWorld and ScienceWorld. As shown in Table 6, transferring copilots between ALFWorld and ScienceWorld consistently outperforms the baseline. ALFWorld copilot outperforms ScienceWorld copilot in overall performance, we speculate this is because ALFWorld collecting more training data. Notably, the Mix Copilot (trained on mix data collected from ALFWorld and ScienceWorld) proves effective for both domains, particularly providing substantial gains for the task with limited training resources (ScienceWorld). These results provide compelling evidence that MCMA facilitates generalization by learning a transferable meta-cognitive skill for memory management.

5 Conclusion

We propose MCMA, a meta-cognitive memory abstraction method that treats experience reuse as a learnable problem. By decoupling task execution from memory management, MCMA uses a transferable memory copilot trained via preference optimization to abstract and organize interaction trajectories. Experiments on ALFWorld, ScienceWorld, and BabyAI show that MCMA improves task performance, generalization, and cross-task transfer, highlighting the importance of learning how to remember in LLM-based agents.

Limitations

MCMA has achieved good performance and generalization, but it also comes with several limitations. First, MCMA introduces additional computational overhead during training, as constructing preference supervision requires generating and evaluating multiple candidate abstractions for each trajectory. While this cost is incurred offline and the learned memory copilot can be reused across tasks, the overall training pipeline remains more expensive than retrieval-based or single-representation memory methods. Second, although MCMA organizes experience into a hierarchy of abstraction levels to support flexible reuse, the current reuse process relies on a manually designed strategy to select which abstraction level should be applied to a new task. This limits full end-to-end adaptivity and suggests an important direction for future work: learning abstraction-level selection jointly with memory structuring and reuse.

Ethical consideration

Our research explores the reuse of Large Language Models (LLMs) memory. Despite undergoing additional fine-tuning in various experiments, these models retain ethical and social risks inherent in their pretraining data. Notably, open-source LLMs may incorporate private or contentious data during the training phase, thereby raising additional ethical concerns.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. 2025. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *arXiv preprint arXiv:2512.10696*.

Shiqi Chen, Tongyao Zhu, Zian Wang, Jinghan Zhang, Kangrui Wang, Siyang Gao, Teng Xiao, Yee Whye Teh, Junxian He, and Manling Li. 2025a. Internalizing world models via self-play finetuning for agentic rl. *arXiv preprint arXiv:2510.15047*.

Silin Chen, Shaoxin Lin, Xiaodong Gu, Yuling Shi, Heng Lian, Longfei Yun, Dong Chen, Weiguo Sun, Lin Cao, and Qianxiang Wang. 2025b. Swe-exp: Experience-driven software issue resolution. *arXiv preprint arXiv:2507.23361*.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*.

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

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *CoRR*.

Minsoo Kim, Victor Bursztyn, Eunyee Koh, Shunan Guo, and Seung-won Hwang. 2024. Rada: Retrieval-augmented web agent planning with llms. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 13511–13525.

Yi Kong, Dianxi Shi, Guoli Yang, Chenlin Huang, Xiaopeng Li, Songchang Jin, and 1 others. 2025. Mapagent: Trajectory-constructed memory-augmented planning for mobile task automation. *arXiv preprint arXiv:2507.21953*.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. 2025. Agentauditor: Human-level safety and security evaluation for llm agents. *arXiv preprint arXiv:2506.00641*.

Feng Peiyuan, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. 2024. Agile: A novel reinforcement learning framework of llm agents. *Advances in Neural Information Processing Systems*, 37:5244–5284.

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

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

Ewan Ritchford. 2025. Optimizing llm-agents with history-driven task planning. *Journal of Computer Technology and Software*, 4(3).

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024a. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories. *arXiv preprint arXiv:2410.07706*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024b. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, and 1 others. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8416–8439.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*.

Endel Tulving. 1972. "episodic and semantic memory," in organization of memory. *(No Title)*, page 381.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. 2025a. Steca: Step-level trajectory calibration for llm agent learning. *arXiv preprint arXiv:2502.14276*.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024a. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*.

Sai Wang, Yu Wu, and Zhongwen Xu. 2025b. Cogito, ergo ludo: An agent that learns to play by reasoning and planning. *arXiv preprint arXiv:2509.25052*.

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

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

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. 2025. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, and 1 others. 2023. Retroformer: Retrospective large language agents with policy gradient optimization. *arXiv preprint arXiv:2308.02151*.

Shicheng Ye, Chao Yu, Kaiqiang Ke, Chengdong Xu, and Yinqi Wei. 2025. H2r: Hierarchical hindsight reflection for multi-task llm agents. *arXiv preprint arXiv:2509.12810*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025b. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47.

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

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. Synapse: Trajectory-as-exemplar prompting with memory for computer control. *arXiv preprint arXiv:2306.07863*.

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

A Appendix

A.1 Dataset Details

ALFWorld. (Shridhar et al., 2020b) We conduct experiments using the ALFWorld environment (Shridhar et al., 2020b), a framework designed to bridge abstract text-based reasoning and embodied execution. ALFWorld aligns the text-based TextWorld engine (Côté et al., 2018) with the visually grounded ALFRED benchmark (Shridhar et al., 2020a) by utilizing the Planning Domain Definition Language (PDDL) to synchronize state dynamics across modalities. The dataset encompasses six types of compositional household tasks (e.g., *Pick & Place*, *Clean & Place*, *Heat & Place*) distributed across 120 interactive scenes, including kitchens, bedrooms, bathrooms, and living rooms. To evaluate generalization, ALFWorld defines strict data splits: a **Seen** set containing 140 task instances in rooms encountered during training, and an **Unseen** set containing 134 tasks in novel rooms with different layouts and object placements, thereby testing the agent's ability to perform zero-shot transfer in out-of-distribution environments.

ScienceWorld. (Wang et al., 2022) ScienceWorld is a complex interactive text environment designed to evaluate the procedural reasoning capabilities of agents within the context of an elementary school science curriculum. Distinct from static question-answering benchmarks, ScienceWorld simulates a dynamic physical world with underlying engines for thermodynamics, electrical circuits, chemistry, and biological life cycles. The benchmark consists of 30 diverse tasks across 10 topics, ranging from testing electrical conductivity to conducting genetic experiments, requiring agents to synthesize declarative scientific knowledge into multi-step action sequences (e.g., *heating a beaker*, *connecting a battery*). There are 1819 tasks in the test set and 1796 tasks in the dev set. Furthermore, the environment enforces rigorous evaluation through thousands of parametric variations, testing an agent's ability to generalize scientific concepts to novel objects and scenarios.

ALFWorld Task Example

Task Description: Examine the alarmclock with the desklamp.
Initial Observation: You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 1, a drawer 3, . . . and a sidetable 2.

`> go to desk 1`
Observation: You arrive at desk 1. On the desk 1, you see a **alarmclock 1**, a bowl 3, a cellphone 2, . . . and a pencil 2.

`> take alarmclock 1 from desk 1`
Observation: You pick up the alarmclock 1 from the desk 1.

`> go to sidetable 2`
Observation: You arrive at sidetable 2. On the sidetable 2, you see a cellphone 1, a **desklamp 1**, and a keychain 2.

`> use desklamp 1`
Observation: You turn on the desklamp 1.

Goal Achieved.

Figure 5: A simplified trajectory of an agent performing an *Examine in Light* task in ALFWorld.

Figure 6: An example of ScienceWorld.

ScienceWorld Interaction Example

Task Description: Find a living thing. First, focus on the thing. Then, move it to the orange box in the kitchen.

`> look around` **[Score: 0.08]**
Observation: This location is called the outside. You see a **blue jay egg**, a butterfly egg, a dove egg, an axe, and a door to the kitchen.

`> focus on blue jay` **[Score: 0.58]**
Observation: You focus on the blue jay egg.

`> pick up blue jay` **[Score: 0.67]**
Observation: You move the blue jay to the inventory.

`> go to kitchen` **[Score: 0.83]**
Observation: You move to the kitchen. You see a counter, a table, and an **orange box**.

`> move blue jay to orange box` **[Score: 1.00]**
Observation: You move the blue jay to the orange box.

Task Completed.

Figure 7: A step-by-step trajectory for the "Find a living thing" task in ScienceWorld. The score (right) updates incrementally as the agent completes sub-goals: identifying the correct object (focus), acquiring it, navigating, and placing it in the target container.

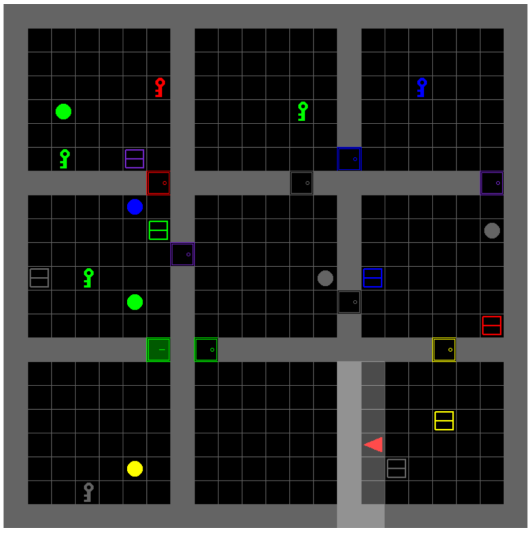

Figure 8: A 2D grid image example of BabyAI.

BabyAI (Chevalier-Boisvert et al., 2018) We evaluate our transfer ability on the **BabyAI** platform (Chevalier-Boisvert et al., 2018), a benchmark designed to study the sample efficiency of grounded language learning agents. BabyAI is built upon the MiniGrid framework, featuring a partially observable 2D gridworld where agents perceive a 7×7 symbolic local view. The platform defines a curriculum of 19 levels of increasing difficulty, ranging from single-instruction tasks (e.g., *GoToObj*) to complex composite missions involving navigation and manipulation (e.g., *BossLevel*).

Figure 8 is an example of BabyAI's 2D grid image. To bridge the modality gap, we employ a deterministic translator that converts the agent's 7×7 egocentric symbolic observation into natural language. The agent is virtually anchored at the bottom-center coordinate $(3, 6)$ of the observable grid. For every salient object i located at (x_i, y_i) (excluding static background elements like walls), we calculate its relative spatial position using Manhattan distances: longitudinal offset $d_{\text{fwd}} = 6 - y_i$ and lateral offset $d_{\text{lat}} = x_i - 3$. These coordinates are mapped to egocentric linguistic templates (e.g., *"A red ball is 3 steps forward and 1 step to the left"*), converting the sparse tensor into a dense descriptive list of visible entities. The final prompt fed to the LLM is constructed by concatenating the static system instructions, the specific *Mission Goal*, a chronological *History of Actions* $a_{0:t-1}$, and the *Current Observation* generated above. The system instruction strictly defines the admissible action space (e.g., *move forward*, *toggle*) and enforces a structured JSON output format. This design ensures the model grounds its decision-making in both the immediate visual context and the temporal trajectory of the episode, allowing for chain-of-thought reasoning before action selection.

A.2 Baseline Details

ExpeL ExpeL (Zhao et al., 2024) is a non-parametric learning framework that enables LLM agents to improve autonomously through experience gathering and knowledge extraction. During a training phase, the agent collects successful and failed trajectories via trial-and-error (utilizing ReAct and Reflexion). It then abstracts cross-task knowledge into natural language *insights* (e.g., guidelines or constraints) and stores successful trajectories in a vector database. At inference time, ExpeL augments the agent's context with these extracted insights and dynamically retrieves the top-k most similar successful past trajectories to serve as few-shot examples, thereby leveraging both abstract rules and concrete experiences to enhance decision-making without gradient updates.

TRAD TRAD (Thought Retrieval and Aligned Decision) (Zhou et al., 2024) is a novel framework designed to enhance Large Language Model

BabyAI Interaction Example

Task Description: Go to the green key.

`> move forward` **[Score: 0.00]**
Observation: You see a **green key** 3 steps forward and 2 steps to the left.

`> turn left` **[Score: 0.00]**
Observation: You turn left. You see a **green key** 2 steps forward and 3 steps to the right.

`> move forward` **[Score: 0.00]**
Observation: You see a **green key** 1 step forward and 3 steps to the right.

`> move forward` **[Score: 0.00]**
Observation: You see a **green key** directly under you and 3 steps to the right.

`> turn right` **[Score: 0.00]**
Observation: You turn right. You see a **green key** 3 steps forward.

`> move forward` **[Score: 0.00]**
Observation: You see a **green key** 2 steps forward.

`> move forward` **[Score: 0.90]**
Observation: You reach the goal.

Task Completed.

Figure 9: A step-by-step trajectory for the "Go to the green key" task in BabyAI. Unlike ScienceWorld, BabyAI typically uses sparse rewards, meaning the score (right) remains 0.00 until the specific goal condition is met at the final step.

(LLM) agents in sequential decision-making tasks by addressing the limitations of traditional trajectory-level retrieval, such as irrelevant context and context window constraints. The framework comprises two core components: *Thought Retrieval* and *Aligned Decision*. The former employs a step-wise retrieval mechanism where the agent generates a "thought"—an abstraction of the current state—to query a memory of expert demonstration steps, ensuring high relevance and minimizing noise. The latter augments these retrieved steps with their temporal neighbors through techniques including Temporal Expansion, Relative Order Marks, and History Alignment, thereby recovering essential contextual dynamics lost in single-step retrieval. Empirical evaluations on benchmarks such as ALFWorld and Mind2Web demonstrate that TRAD significantly outperforms state-of-the-art baselines by effectively balancing context sufficiency with noise reduction.

A.3 Task Preprocess Details

This section details the task preprocessing pipeline. As illustrated in Algorithm 1, raw trajectories are transformed into a hierarchical tree structure Ψ through a recursive coarse-to-fine decomposition strategy. The process initiates with trajectory pruning, which eliminates redundant noise from the raw sequence. Subsequently, the sequence undergoes temporal decomposition to identify distinct sub-goals, followed by a consistency verification step where segments are validated and labeled to ensure logical coherence. This decomposition is applied recursively: segments containing three or more actions are subject to further partitioning, whereas shorter segments are preserved as atomic leaf nodes, thereby yielding a structured representation of long-horizon tasks. A raw trajectory is shown in Figure 10, and a processed example is shown in 11.

Algorithm 1 Hierarchical Task Trajectory Decomposition

Require: Full task trajectory $\mathcal{T}_{\text{raw}}$
Ensure: Hierarchical Task Tree Ψ

Initialization:
$\mathcal{T}_{\text{clean}} \leftarrow \mathcal{A}_{\text{prune}}(\mathcal{T}_{\text{raw}})$
Initialize root node n_{root} with $\mathcal{T}_{\text{clean}}$
Ψ.setRoot(n_{root})
RECURSIVEDECOMPOSE(n_{root})
return Ψ

function RECURSIVEDECOMPOSE(n_{parent})
    $\tau \leftarrow n_{\text{parent}}$.trajectory
    $valid \leftarrow$ False
    while not $valid$ **do**
        $\mathcal{S} \leftarrow \mathcal{A}_{\text{split}}(\tau)$
    end while
    for all subtask $s_i \in \mathcal{S}$, label $l_i \in \mathcal{L}$ **do**
        Create node n_{child} with content s_i and label l_i
        Ψ.addChild($n_{\text{parent}}, n_{\text{child}}$)
        if Length(s_i) ≥ 3 **then**
        else
        end if
    end for
end function

A.4 Examples of Structured Knowledge

In this section, we provide some structured knowledge examples derived from MCMA (tree structure, chain structure, key-value structure, natural language structure, and nest structure. As shown in Figure 14, 15, 16, 17, 18). Figure 13 shows the prompt used for guiding memory copilot to gener-

Raw Task Example

```
Raw Trajectory
STEP 1:go to cabinet 2> You arrive at cabinet 2. The cabinet 2is closed.
STEP 2:open cabinet 2> You open the cabinet 2. The cabinet 2is open. In it, you see a
    bowl 1.
STEP 3:take bowl 1from cabinet 2> You pick up the bowl 1from the cabinet 2.
STEP 4:go to diningtable 1> You arrive at diningtable 1. On the diningtable 1,you see
    a pen 1,a potato 1,and a tomato 1.
STEP 5:move bowl 1to diningtable 1> You move the bowl 1to the diningtable 1.
STEP 6:take bowl 1from diningtable 1> You pick up the bowl 1from the diningtable 1.
STEP 7:go to fridge 1> You arrive at fridge 1. The fridge 1is closed.
STEP 8:open fridge 1> You open the fridge 1. The fridge 1is open. In it, you see
    nothing.
STEP 9:cool bowl 1with fridge 1> You cool the bowl 1using the fridge 1.
STEP 10:go to diningtable 1> You arrive at diningtable 1. On the diningtable 1,you see
    a pen 1,a potato 1,and a tomato 1.
STEP 11:move bowl 1to diningtable 1> You move the bowl 1to the diningtable 1.
```

Figure 10: A task trajectory example in ALFWorld.

Example After Task Process

```
|-- name: Root Task
|-- goal: put a cool bowl in diningtable.
|-- steps_count: 11
|-- trajectory:
|-- subtasks:
| |-- Subtask:
| | |-- name: Take bowl from cabinet
| | |-- goal: put a cool bowl in diningtable.
| | |-- steps_count: 3
| |-- Subtask:
| | |-- name: Place bowl on dining table
| | |-- goal: put a cool bowl in diningtable.
| | |-- steps_count: 2
| |-- Subtask:
| | |-- name: Cool bowl in fridge
| | |-- goal: put a cool bowl in diningtable.
| | |-- steps_count: 4
| | | |-- Subtask:
| | | | |-- name: Take bowl from diningtable
| | | | |-- goal: put a cool bowl in diningtable.
| | | | |-- steps_count: 1
| | | |-- Subtask:
| | | |-- name: Cool bowl using fridge
| | | |-- goal: put a cool bowl in diningtable.
| | | |-- steps_count: 3
| |-- Subtask:
| |-- name: Place cooled bowl on dining table
| |-- goal: put a cool bowl in diningtable.
| |-- steps_count: 2
```

Figure 11: A processed example in ALFWorld.

ate structural knowledge, which outlines the role and instructions, the required JSON structures and the few-shot example provided to the model.

A.5 Task Prompt

Figure 12 shows our task prompt for ALFWorld and ScienceWorld.

A.6 Reuse Memory Copilot on Same Series LLMs

Since the data acquisition for each memory copilot is driven by the specific performance of the task model, the copilot is inherently aligned with the model's capabilities, thereby providing critical assistance in challenging scenarios. However, training a dedicated copilot entails significant computational overhead. To address this, we investigate the transferability of the memory copilot across different models within the same lineage. As evidenced in Tables 7 and 8, the memory copilot optimized for Qwen3-32B demonstrates robust performance when deployed on other LLMs in the same series.

Type	Config	Performance	
		Step ↓	Acc ↑
Base	**Default**	38.67	0.40
Sum	**Default**	31.39 ↓7.28	0.51 ↑11%
	Trained	31.31 ↓7.36	0.55 ↑15%
Reflection	**Default**	35.18 ↓3.49	0.44 ↑4%
	Trained	32.99 ↓5.68	0.51 ↑11%
MCMA	**Default**	30.08 ↓8.59	0.54 ↑14%
	Trained	**28.37** ↓10.30	**0.59** ↑19%

Table 7: Reuse memory copilot trained for Qwen3-32B on Qwen3-4B.

A.7 Hierarchical Abstraction Processing

This section details our approach to handling hierarchical knowledge. The process begins with **semantic embedding generation**, where we encode each knowledge entry by concatenating its name and textual description into a unified semantic vector. To model the relationships between these entries, we proceed with **sparse semantic graph construction**. Specifically, we compute the cosine similarity between knowledge vectors and construct a k-Nearest Neighbor (k-NN) graph, retaining only the top-k ($k = 10$) strongest connections to form a sparse adjacency matrix.

Type	Config	Performance	
		Step ↓	Acc ↑
Base	**Default**	27.93	0.62
Sum	**Default**	24.68 ↓3.25	0.66 ↑4%
	Trained	**22.51** ↓5.42	0.69 ↑7%
Reflection	**Default**	27.35 ↓0.58	0.65 ↑3%
	Trained	25.16 ↓2.77	**0.71** ↑9%
MCMA	**Default**	24.25 ↓3.68	0.66 ↑4%
	Trained	23.70 ↓4.23	0.69 ↑7%

Table 8: Reuse memory copilot trained for Qwen3-32B on Qwen2.5-72B.

Building upon this topology, we employ **hierarchical clustering** to identify latent semantic communities within the graph. The final consolidation is achieved through a two-phase strategy: first, **intra-cluster fusion** merges redundant or highly similar knowledge points within each identified cluster; subsequently, **inter-cluster abstraction** re-aggregates these fused clusters to derive higher-level abstract concepts, thereby forming a structured and concise knowledge hierarchy.

As shown in Figure 19 and Figure 20, we provide the examples of different level of knowledge provided by MCMA. Lower-level knowledge focuses on complete task details, while higher-level knowledge focuses on capturing common problems at higher levels.

Task Prompt

```
### HISTORY OF ACTIONS ###
Here is the history of your actions and observations so far:
{history_str}

### GOAL ###
Your ultimate goal is:
{game_task}

### CURRENT STATE ###
{current_state}

### AVAILABLE ACTIONS ###
You MUST choose one of the following valid actions. Do not make up your own actions.
Admissible actions: {admissible_commands}

Based on all the information, first think step-by-step about what to do next. Then,
    output your final chosen action in the format 'Action: <action>'.
```

Figure 12: Task Prompt for our tasks.

Generate Knowledge Prompt

```
# ROLE AND GOAL
You are an advanced AI assistant specializing in induction and reasoning from multiple
    task execution trajectories. Your core objective is to identify and extract common
    patterns, strategies, or knowledge embedded within these trajectories and to
    summarize this shared knowledge in a flexible, structured format. You are required
    to utilize various structures, including trees, chains, graphs, and natural
    language to represent the inherent logic of the knowledge in the most effective
    way.

# INSTRUCTIONS
1. Analyze Inputs Comprehensively: Scrutinize all provided task trajectories. Each
    trajectory contains an overall goal and a detailed sequence of action steps.
2. Identify Common Patterns: Look for commonalities across different trajectories.
    These commonalities might manifest as:
   (1) Task Decomposition Structure: Is a high-level task consistently broken down
       into similar sub-tasks? (e.g., "place two items" is decomposed into two "place
       single item" sub-tasks).
   (2) Action Sequence: Does completing a sub-task follow a recurring sequence of
       actions? (e.g., `navigate to item -> pick up item -> navigate to destination ->
       place item`).
   (3) Object & Location Relationships: Is there general knowledge about where certain
       types of objects are typically found or should be placed?
3. Select the Optimal Structure: For each piece of summarized common knowledge, choose
    the most appropriate structure for its representation:
   (1) Tree: Use to represent hierarchical relationships, such as task decomposition,
       where a main goal branches into sub-tasks or steps.
   (2) Chain: Use to represent strictly ordered sequences of actions or processes.
   (3) Key-Value: Use to represent properties, attributes, or mappings between
       entities (e.g., an object and its common locations).
   (4) Natural Language: Use to provide an intuitive, human-readable summary of
       complex patterns or insights.
4. Combine and Nest Structures: You have the flexibility to combine and nest these
    structures. For example, a leaf node in a tree structure could itself be a chain
    structure detailing the execution flow for that sub-task.
5. Format the Output: Consolidate all extracted knowledge into a single JSON object.
    This object must contain a list named `knowledge`, where each element represents a
    distinct piece of knowledge and adheres to the JSON structures defined below.

# STRUCTURE FORMATS
...

# EXAMPLE OUTPUT:
Answer: {
  "General Plan for Placing Two Items in a Receptacle": {
    "name": "General Plan for Placing Two Items in a Receptacle",
    "structured_storage": {
    "type": "tree",
    "root": {"name": "put two <item> in <receptacle>",
       "children": [{
          "name": "Process first <item>",
          "children": [{"structured_storage": {"type": "chain","nodes": [{"step": "go
              to location of <item> 1"},{"step": "take <item> 1"},{"step": "go to
              <receptacle>"},{"step": "(optional) open <receptacle> if it is
              closed"},{"step": "move <item> 1to <receptacle>"},...
]}}]}}

# TASK START
Analyze the following input data and generate the JSON output according to the formats
    instructed. Start with 'Answer:' before outputting the official answer.
[input_data]
```

Figure 13: The prompt to guide memory copilot in generating structured knowledge.

Tree Example

```
{
  "name": "Inspect an Object Using a Light Source",
  "description": "A process for examining a specific <object> using a <light source>.
      It begins by navigating to the location of the <light source> and turning it on,
      then proceeds to locate and retrieve the <object> from its <location>, and
      finally returns to the <light source> location to inspect the <object> under the
      illumination.",
  "knowledge": {
    "name": "General Procedure for Observing an Object Under a Light Source",
    "source": ["look at cellphone under the desklamp"],
    "structured_storage": {
      "type": "tree",
      "root": {
        "name": "look at <object> under <light source>",
        "children": [
          {"name": "Navigate to <light source> location",
            "children": [{"structured_storage": {"type": "chain", "nodes": [
                  {"step": "go to location of <light source>" },
                  {"step": "use <light source> to turn it on" }]}}]},
          {"name": "Locate and Retrieve <object>",
            "children": [{"structured_storage": {"type": "chain", "nodes": [
                  {"step": "go to location of <object>" },
                  {"step": "take <object> from <location>" }]}}]},
          {"name": "Return to <light source> location",
            "children": [{"structured_storage": {"type": "chain", "nodes": [
                  {"step": "go back to location of <light source>" }]}}]}
]}}}}
```

Figure 14: Tree structure knowledge examples.

Chain Example

```
{
  "name": "Retrieve an Object with Lighting Assistance",
  "description": "A process for retrieving an object using a light source for
      assistance. It begins by navigating to the light source and turning it on, then
      moving to the object's location to retrieve it, and finally returning to the
      light source.",
  "knowledge": {
    "name": "Core Action Sequence for Observing an Object Under a Light Source",
    "source": ["look at cellphone under the desklamp"],
    "structured_storage": {
      "type": "chain",
      "nodes": [
        {"step": "Navigate to the location of the light source." },
        {"step": "Turn on the light source." },
        {"step": "Navigate to the location of the object." },
        {"step": "Retrieve the object from its location." },
        {"step": "Return to the location of the light source." }
      ]
    }
  }
}
```

Figure 15: Chain structure knowledge examples.

Key-Value Example

```
{
  "name": "Organize Items in Designated Locations",
  "description": "A simple process for placing specific items in their designated
      receptacles. The task involves identifying an <item> and its corresponding
      <receptacle> from a predefined mapping, and then moving the <item> to the
      specified <receptacle> for organization.",
  "knowledge": {
    "name": "Common Object Locations",
    "source": ["look at cellphone under the desklamp"],
    "structured_storage": {
      "type": "key_value",
      "data": {"cellphone": ["armchair"], "desklamp": ["desk"]
      }
    }
  }
}
```

Figure 16: Key-Value structure knowledge examples.

Natural Language Example

```
{
  "name": "Observe an Object Under a Light Source",
  "description": "A structured process where the agent navigates to a light source,
      activates it, locates and retrieves a target object, and then returns to the
      light source to observe the object under its illumination.",
  "knowledge": {
    "name": "Iterative Task Strategy for Observing an Object Under a Light Source",
    "source": ["look at cellphone under the desklamp"],
    "structured_storage": {
      "type": "natural_language",
      "text": "The agent follows a structured, iterative approach to observe an object
          under a light source. It first navigates to the light source, turns it on,
          then locates and retrieves the object, and finally returns to the light source
          to observe it under the light."
    }
  }
}
```

Figure 17: Natural Language structure knowledge examples.

Nested Example (Tree + Chain)

```
{"name": "Inspect an Object Using a Light Source",
 "description": "A process for examining a specific <object> using a <light source>.",
 "knowledge": {
  "name": "General Procedure for Observing an Object Under a Light Source",
  "source": ["look at cellphone under the desklamp"],
  "structured_storage": {
   "type": "tree",
   "root": {
    "name": "look at <object> under <light source>",
    "children": [
     {"name": "Navigate to <light source> location",
      "children": [{"structured_storage": {"type": "chain", "nodes": [
            {"step": "go to location of <light source>" },
            {"step": "use <light source> to turn it on" }]}}]},
     {"name": "Locate and Retrieve <object>",
      "children": [{"structured_storage": {"type": "chain", "nodes": [
            {"step": "go to location of <object>" },
            {"step": "take <object> from <location>" }]}}]},
     {"name": "Return to <light source> location",
      "children": [{"structured_storage": {"type": "chain", "nodes": [
            {"step": "go back to location of <light source>" }]}}]}]}
  }
 }
}
```

Figure 18: Nested structure knowledge examples.

Knowledge of Abstract Level 1 Example

```
{
 "name": "Cool and Replenish Mug in Coffee Machine",
 "description": "An iterative process for preparing a cooled mug for use in a coffee
     machine. The task begins by retrieving the mug from a <container> (e.g., a
     cabinet), moving it to the coffee machine, and then returning it to cool in a
     <cooling_receptacle> (e.g., a fridge). Once cooled, the mug is placed back into
     the coffee machine for use.",
 "knowledge": {
  "name": "Iterative Task Strategy for Mug Placement",
  "source": ["put a cool mug in coffeemachine"],
  "structured_storage": {
   "type": "natural_language",
   "text": "The agent follows an iterative process for placing a cool mug in the
       coffee machine. It first retrieves the mug from the cabinet, moves it to the
       coffee machine, takes it back, cools it in the fridge, and then places the
       cooled mug back into the coffee machine."
  }
 }
}
```

Figure 19: An Example Knowledge of Abstract Level 1.

Knowledge of Abstract Level 2 Example


```
{
  "name": "General Plan for Cooling and Storing an Item",
  "description": "A generic, multi-step process for cooling and storing an object. The
      task begins with retrieving a specific <item> from its <container>, proceeds to
      cool the <item> using a <cooling_source>, and concludes by placing the cooled
      <item> into a designated final <receptacle> for storage.",
  "structured_storage": {
    "type": "tree",
    "root": {
      "name": "cool <item> and put it in <receptacle>",
      "children": [
        {"name": "Retrieve <item>",
          "children": [{"structured_storage": {"type": "chain", "nodes": [{"step": "go to
              location of <item>" },{"step": "open <container> if it is closed" },{
              "step": "take <item> from <container>" }]}}]},
        {"name": "Cool <item>",
          "children": [{"structured_storage": {"type": "chain", "nodes": [{"step": "go to
              <cooling_source>" },{"step": "open <cooling_source> if it is closed" },{
              "step": "cool <item> with <cooling_source>" }]}}]},
        {"name": "Place <item> in <receptacle>",
          "children": [{"structured_storage": {"type": "chain", "nodes": [{"step": "go to
              <receptacle>" },{"step": "move <item> to <receptacle>" }]}}]}
      ]
    }
  }
}
```


Figure 20: An Example Knowledge of Abstract Level 2.