MEMORY AS A CONTROLLED PROCESS: LEARNED ADAPTIVE MEMORY MANAGEMENT FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLM) agents increasingly rely on external memory systems to accumulate experience across tasks. Yet many existing approaches, from graph-structured memories to reflective insight stores, access memory through *fixed, hand-designed heuristics*. We argue that this static view of memory is a core bottleneck for agentic learning because optimal memory behavior is fundamentally context-dependent. The early stages of tasks benefit from minimal retrieval because memory is sparse; recurring goal types benefit from plan reuse rather than generic nearest-neighbor lookup; stuck agents benefit from re-retrieval with alternative queries; and across long task streams, the memory store itself must be consolidated and pruned to remain useful. We present MEMCON (**Mem**ory as a **Con**trolled Process), a framework that models memory operations as a Markov Decision Process and learns an online policy that adaptively decides *when*, *what*, and *how much* to retrieve, when to inject a distilled plan, and when to consolidate or forget. MEMCON is *backend-agnostic*: it wraps compatible memory implementations through a common interface, learns from task-by-task binary feedback with no controller pretraining or additional LLM calls for policy selection, and uses a lightweight tabular contextual bandit with UCB exploration that adapts during deployment. Across 6 benchmarks, 3 agent frameworks, and 3 LLM backbones, MEMCON improves average performance over the evaluated memory baselines; on GPT-4.1-mini, it raises six-benchmark mean success by 2.6–4.2 points over G-Memory while reducing mean input-token use by 10.9–24.1%. Our code is available at https://anonymous.4open.science/r/MemCon-9071/

1 INTRODUCTION

Large Language Models (LLMs) deployed as autonomous agents have demonstrated remarkable capabilities in interactive environments, multi-step planning, and tool use (Yao et al., 2023b;a; Wang et al., 2024b; Xi et al., 2023). A critical enabler of this progress is *memory*, the ability to store experiences from solved tasks and retrieve them on future ones, so that an agent can avoid repeating past mistakes, reuse working strategies, and incrementally accumulate domain knowledge rather than starting from scratch on every episode (Park et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024a; Zhang et al., 2025).

Despite rapid progress in *what* to store (trajectories, reflections, insights, skills, graphs, latent tokens (Zhang et al., 2025; Fu et al., 2026; Packer et al., 2023; Chhikara et al., 2025; Fang et al., 2025; Park et al., 2023)), many existing memory systems treat memory *access* as a static pipeline: a single global retriever is called once per step with a hard-coded `top-`k, a hard-coded graph hop depth, and a hard-coded query template, and uses the same configuration whether memory is empty or contains thousands of trajectories, whether the task is familiar or novel, and whether the agent is progressing or has been executing the same action for five steps. A complementary line of work—most prominently MemGPT (Packer et al., 2023)—makes memory access adaptive by promoting the LLM itself to the role of memory controller (paginating, formulating queries, and deciding when to recall), using LLM inference to select memory operations. MEMCON occupies a third design point: a *learned but lightweight* controller that is adaptive without any extra LLM calls. This static-pipeline view can be a bottleneck for agentic memory systems: static memory systems can *either* retrieve too aggressively (inflating context and cost while hurting accuracy on simple queries) *or* too conservatively (missing

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

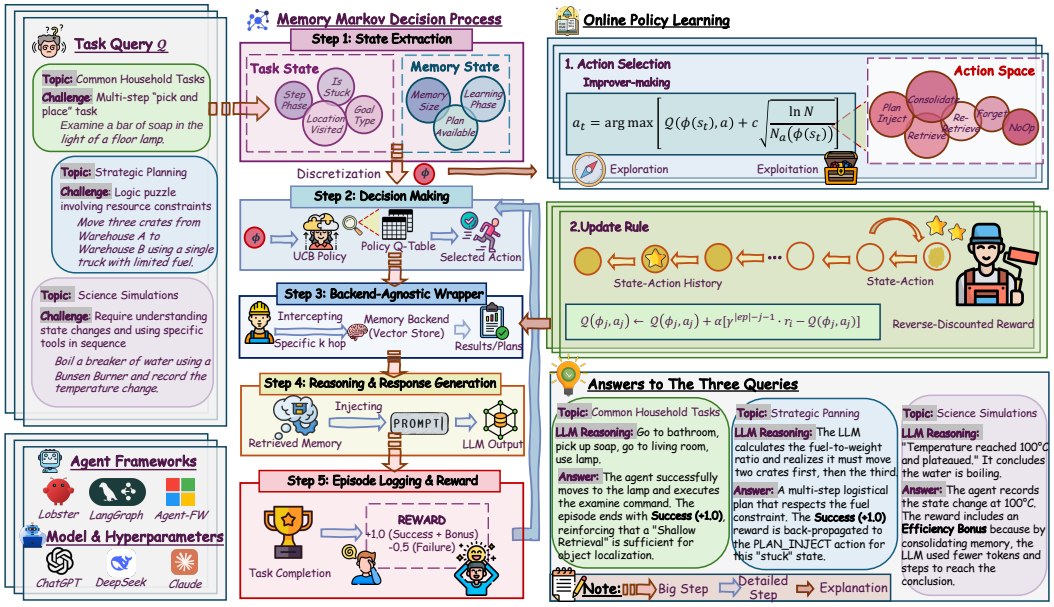


Figure 1: **Overview of MEMCON.** *(Left)* Task streams from ALFWorld, PDDL, and ScienceWorld are executed through three agent frameworks (Lobster, LangGraph, Agent-FW) sharing one LLM backbone. *(Middle)* The *Memory MDP* runs four stages of memory control: extract a compact state $\phi(s)$ from task + memory signals; select an action via the UCB policy over $Q(\phi, a)$; the backend-agnostic wrapper issues retrieval to the inner backend with policy-chosen `top_k`/`insight_k`/hop; retrieved context (optionally plus an injected success plan) is fed to the LLM, and the episode is scored ($+1$ success, -0.5 failure, plus efficiency bonus). *(Right)* Online learning: the action space is {RETRIEVE (varying depth), PLANINJECT, RE-RETRIEVE, CONSOLIDATE, FORGET, NOOP}; after each episode the reverse-discounted reward $\gamma^{|\text{ep}|-j-1} r_i$ updates every visited (ϕ_j, a_j).

key reusable plans and reflections), because any single setting is miscalibrated for at least one of the regimes above.

We argue that effective agent memory requires an adaptive control layer sitting on top of the storage backend. Concretely: (i) *Early tasks* should retrieve *less*, because there is little useful experience to draw on and indiscriminate retrieval only dilutes the prompt. (ii) *Recurring goal types* should prefer *plan reuse*, replaying a distilled, object-generalized template of a prior success, rather than nearest-neighbor retrieval over raw trajectories. (iii) *Stuck agents* that repeat actions should trigger *re-retrieval* with an alternative query rather than re-reading the same top-k that already failed to help. (iv) *Long task streams* require *consolidation and forgetting* so that a growing, noisy memory remains useful rather than overwhelming. These regimes motivate a policy rather than one global setting; the right memory operation is a function of task progress, memory state, and the agent's learning phase.

We introduce MEMCON (**Mem**ory as a **Con**trolled Process), a framework that reformulates memory access as a *sequential decision problem*. MEMCON casts the choice of memory operation (RETRIEVE, PLANINJECT, RE-RETRIEVE, CONSOLIDATE, FORGET, NOOP) together with its parameters (`top_k`, `insight_k`, graph `hop`) as actions in a *Memory MDP*, whose state captures both task progress (goal type, step phase, stuck indicator, locations visited) and memory status (size, plan availability, learning phase). MEMCON learns a policy over this MDP online, during deployment, using a lightweight tabular contextual bandit with UCB exploration (Auer et al., 2002; Li et al., 2010; Sutton & Barto, 2018). The policy is warm-started from human-readable priors, updated via reverse-discounted credit assignment from binary task success, and persists to disk between tasks. Crucially, MEMCON adds **zero additional LLM calls**: policy selection uses a table lookup and does not invoke the backbone.

MEMCON is not a new memory store. It is a thin wrapper that intercepts the abstract `retrieve` and `store` entry points of any *existing* memory backend and decides how to call them. The interface is designed to support flat vector stores, skill libraries (Wang et al., 2024a), summarisation-based memories (Zhong et al., 2024), latent-memory systems (Fu et al., 2026), and graph-structured

memories (Zhang et al., 2025), cleanly separating *what* is stored (backend) from *how* it is accessed (controller) while requiring an adapter for backend-specific capabilities.

We evaluate MEMCON across 6 benchmarks covering both interactive decision-making (ALFWorld (Shridhar et al., 2021), PDDL planning, ScienceWorld (Wang et al., 2022)) and knowledge / web / tool-use QA (TriviaQA (Joshi et al., 2017), WebWalkerQA (Wu et al., 2025), GAIA (Mialon et al., 2023)); 3 agent frameworks (Lobster, LangGraph (LangChain Team, 2024), Microsoft Agent-Framework); 3 LLM backbones (GPT-4.1-mini, Claude Sonnet-4, DeepSeek-V3.2); and compare against memory baselines spanning vector retrieval (MetaGPT (Hong et al., 2024), MemoryBank (Zhong et al., 2024)), skill libraries (Voyager (Wang et al., 2024a)), trajectory summarization (ChatDev (Qian et al., 2024)), generative re-ranking (Generative (Park et al., 2023), ExperienceBank), an adapted rule-memory module (OAgent), graph-based hierarchical memory (G-Memory (Zhang et al., 2025)), and latent memory (LatentMem (Fu et al., 2026)). Across all configurations, MEMCON achieves the best or near-best task success—e.g., 67.9% ALFWorld on GPT-4.1-mini (Lobster), the top score among all 10 memories evaluated in that cell—and improves six-benchmark mean S/A over G-Memory by 2.6–4.2 points on GPT-4.1-mini while reducing mean input-token consumption by 10.9–24.1%.

Contributions.

1. We formalize agent memory management as a *Memory MDP* and introduce a *backend-agnostic wrapper* that decouples the memory **control policy** from the memory **storage backend**, with a small interface for integrating compatible backends (§3.1, §3.3).
2. We propose an *online contextual bandit policy* with UCB exploration, warm-start priors, and reverse-discounted credit assignment that learns during deployment using zero controller pretraining and zero extra LLM calls for policy selection, together with augmented memory operations (§3.2, §3.3).
3. We conduct an extensive evaluation of agent memory on six benchmarks and three agent frameworks, showing consistent accuracy gains *and* token savings, including a component ablation that isolates the learned controller's contribution (§4).

2 RELATED WORK

Memory representations and experience reuse. Agent memory stores trajectories, reflections, rules, procedures, or structured relations: Generative Agents (Park et al., 2023), MemoryBank (Zhong et al., 2024), Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and CLIN (Majumder et al., 2024) provide complementary ways to retain experience. Voyager (Wang et al., 2024a), Agent Workflow Memory (Wang et al., 2024c), MemP (Fang et al., 2025), ProcMEM (Mi et al., 2026), and MemSkill (Zhang et al., 2026a) emphasize reusable procedures; G-Memory (Zhang et al., 2025) and LatentMem (Fu et al., 2026) organize memory for multi-agent systems. These representations address what experience is available. MemCon focuses on the policy that selects memory operations and their parameters. Our comparisons use adapted memory modules in a shared harness rather than reproductions of every original end-to-end system.

Adaptive retrieval and learned memory control. Adaptive access is already an active research direction. MemGPT (Packer et al., 2023) delegates memory operations to an LLM; SimpleMem (Liu et al., 2026) adapts retrieval to query complexity, and xMemory (Hu et al., 2026) expands hierarchical retrieval according to reader uncertainty. Memory-R1 (Yan et al., 2026) fine-tunes memory-management and answer agents with outcome-driven reinforcement learning, while MemAct (Zhang et al., 2026b) learns working-context editing actions. Thus, neither adaptive retrieval nor learning memory operations is unique to MemCon. Our design combines an external cross-task store, execution and memory-state features, and a small online controller trained from terminal task feedback while keeping the LLM frozen. This targets settings where controller fine-tuning or additional LLM inference is undesirable.

Online learning and agent orchestration. UCB and contextual bandits (Auer et al., 2002; Li et al., 2010; Lattimore & Szepesvári, 2020) motivate inexpensive exploration over discrete memory actions. MemCon uses an exponentially weighted terminal-return estimate, rather than a bootstrapped

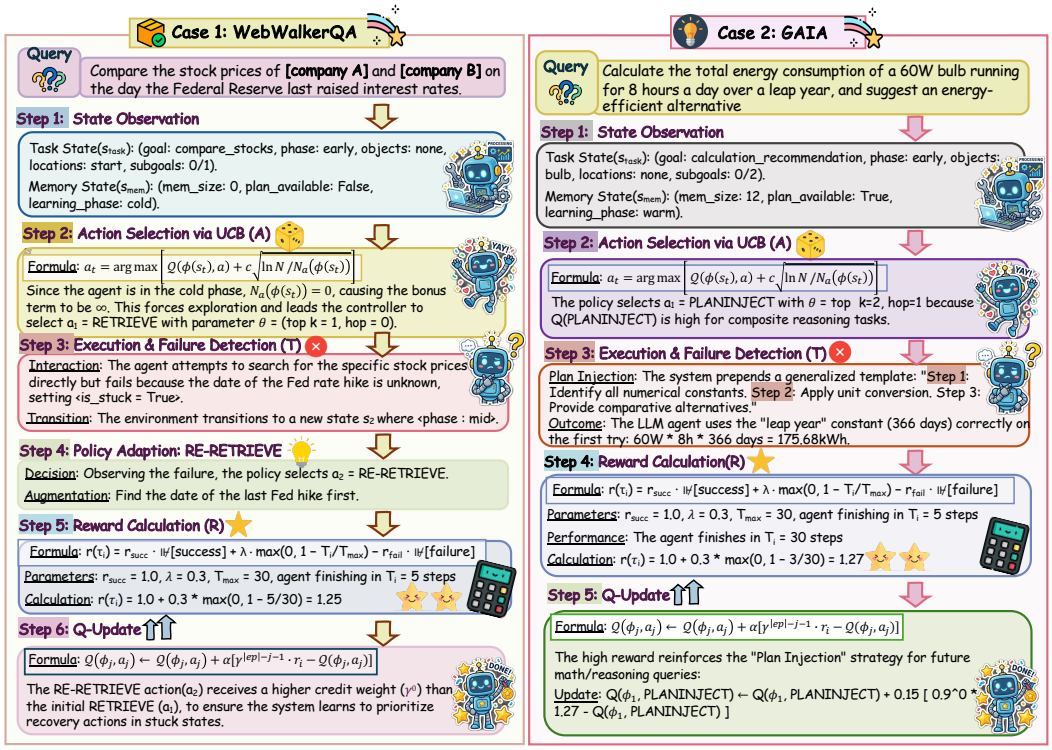


Figure 2: **Illustrative memory-control scenarios.** The left example depicts alternative-query retrieval after a failed attempt; the right depicts plan reuse. These are schematic examples of the intended behavior, not recorded evaluation traces or aggregate evidence of adaptation. In particular, the original QA evaluation used constant interactive-state coordinates.

Q-learning target (Sutton & Barto, 2018); the distinction matters for its guarantees (Appendix E). Agent systems such as MetaGPT (Hong et al., 2024), ChatDev (Qian et al., 2024), and LangGraph (LangChain Team, 2024) provide different orchestration structures. Our evaluation varies the runner while keeping the memory-control formulation fixed.

3 METHOD

This section formalizes each component: the Memory MDP (§3.1), the online policy (§3.2), the backend-agnostic wrapper (§3.3), and two augmented memory operations tailored to long-horizon agentic tasks (§3.3). Figure 1 gives a high-level overview of MEMCON.

3.1 PROBLEM FORMULATION: MEMORY MDP

Consider an LLM agent solving tasks $\{\tau_1, \ldots, \tau_N\}$ sequentially. At each task τ_i, the agent interacts with an environment over T_i steps with access to a memory system $\mathcal{M}$ (the *backend*). Most existing memory backends expose two abstract endpoints, a `retrieve` call (return up to K items relevant to a query) and a `store` call (write the trajectory of the just-finished task), together with optional maintenance hooks for consolidation and eviction. In standard usage these endpoints are invoked with hard-coded hyperparameters (number of items, search depth, query template, consolidation schedule); we instead model the choice of which endpoint to invoke and with what parameters as a sequential decision problem captured by an MDP $\mathcal{M}_{\text{mem}} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$.

State $\mathcal{S}$. The state $s = (s^{\text{task}}, s^{\text{mem}})$ fuses task-progress and memory-status signals:

$$s^{\text{task}} = (\texttt{goal_type}, \texttt{step_phase}, \texttt{is_stuck}, \texttt{objects_held}, \texttt{locations}), \quad (1)$$

$$s^{\text{mem}} = (\texttt{mem_size}, \texttt{plan_available}, \texttt{learning_phase}), \quad (2)$$

where `step_phase` bins the step count, `is_stuck` $\in \{\text{True}, \text{False}\}$ is set when the agent has emitted the same physical action twice in a row, `learning_phase` indicates whether the agent has completed enough tasks to trust learned Q-values, and `plan_available` records whether a distilled success plan exists for the current goal type. We discretize s into a compact hashable key $\phi(s)$ for tabular learning; the exact bins, thresholds, and the discretization rule are given in Appendix C. The coarse key keeps policy selection inexpensive; effective state variation depends on the task interface.

Actions $\mathcal{A}$. Each action $a = (\texttt{op}, \theta)$ specifies a memory operation together with its parameters:

$$\texttt{op} \in \{\textsc{Retrieve}, \textsc{PlanInject}, \textsc{Re-Retrieve}, \textsc{Consolidate}, \textsc{Forget}, \textsc{NoOp}\}; \quad (3)$$

$$\theta = (\texttt{top_k}, \texttt{insight_k}, \texttt{hop}). \quad (4)$$

RETRIEVE returns the top-`top_k` items from the backend together with up to `insight_k` derived rules (when supported) at search depth `hop`; RE-RETRIEVE re-issues a retrieval with an alternative-approach query suffix, used to escape repeated-action loops; PLANINJECT prepends a generalised success plan when one is available (§3.3); CONSOLIDATE and FORGET call any maintenance hooks the backend exposes (e.g., merging or pruning derived rules) and silently no-op on backends that do not implement them; NOOP skips memory access for the current step. We instantiate $\mathcal{A}$ as a small finite set of representative $(\texttt{op}, \theta)$ configurations spanning shallow-to-deep retrieval plus the maintenance and plan operations; the exact action set used in our experiments is listed in Appendix C (Table 9).

Reward $\mathcal{R}$. Rewards are observed at task end based on environment feedback:

$$r(\tau_i) \;=\; r_{\text{succ}} \cdot \mathbf{1}[\text{success}] \;+\; \lambda \cdot \max(0,\; 1 - T_i/T_{\max}) \;-\; r_{\text{fail}} \cdot \mathbf{1}[\text{failure}], \quad (5)$$

where $r_{\text{succ}}, r_{\text{fail}}, \lambda$ and the horizon $T_{\max}$ are fixed scalars (values in Appendix C, Table 8). Successful and more *efficient* trajectories therefore earn larger credit; failed trajectories earn negative feedback that discourages costly or counterproductive memory usage.

Transitions $\mathcal{T}$. The environment transition is driven by the LLM agent and is opaque to the controller. We use a *contextual-bandit approximation with episode-level feedback*: decisions receive discounted terminal rewards without bootstrapping future action values. This is a tractable controller for a sequential problem, not an exact reduction of the MDP. In particular, maintenance may benefit later tasks without receiving that future credit. Appendix E separates idealized stationary-bandit guarantees from the constant-step-size, evolving-memory implementation.

3.2 ONLINE POLICY LEARNING

We learn $\pi : \mathcal{S} \to \mathcal{A}$ online during deployment, with **no controller pretraining** and **no additional LLM calls for policy selection**.

Action selection via UCB. At each memory decision point we apply the Upper Confidence Bound rule (Auer et al., 2002; Li et al., 2010), with the exploration bonus computed using the *state-specific* visit count:

$$a_t \;=\; \arg\max_{a \in \mathcal{A}} \left[Q(\phi(s_t), a) \;+\; c\sqrt{\frac{\ln N(\phi(s_t))}{N_a(\phi(s_t))}} \right], \quad (6)$$

where c is the UCB exploration coefficient (Appendix C, Table 8), $N(\phi(s_t)) := \sum_{a \in \mathcal{A}} N_a(\phi(s_t))$ is the total number of decisions ever taken at state key $\phi(s_t)$, and $N_a(\phi(s_t))$ is the count of action a at that state. Unvisited actions receive an ∞ bonus, forcing the first $|\mathcal{A}|$ visits to each state to span all actions; warm-start priors (below) break ties among such ∞ bonuses to give a sensible initial ordering. This defines the state-specific selection rule; the stationary sample-mean comparison in Appendix E is analyzed separately.

Warm-start priors. Before any learning, Q-values are initialized with interpretable priors that encode mild domain knowledge: RETRIEVE and PLANINJECT receive positive priors (retrieval is usually helpful, plans are useful when available), RE-RETRIEVE a smaller positive prior (useful only when stuck), CONSOLIDATE a neutral prior, and FORGET and NOOP mildly negative priors. Numerical values are listed in Appendix C (Table 8). These priors provide an initial preference before sufficient task feedback is available.

Credit assignment. After each task τ_i, all (s, a) pairs visited during the episode are updated with a reverse-discounted Monte-Carlo return, so that actions taken closer to the terminal outcome receive stronger credit:

$$Q(\phi_j, a_j) \leftarrow Q(\phi_j, a_j) + \alpha \left[\gamma^{|\text{ep}|-j-1} \cdot r_i - Q(\phi_j, a_j) \right], \tag{7}$$

where α is the step size, $\gamma \in (0, 1)$ is the within-episode discount, $|\text{ep}|$ is the total number of memory decisions executed during episode τ_i, and $j \in \{0, 1, \ldots, |\text{ep}| - 1\}$ is the chronological index of the decision being credited (so the last decision receives the undiscounted r_i and earlier decisions receive geometrically attenuated credit). Numerical values for α and γ are in Appendix C. The Q-table is persisted to disk every few updates, so learning carries across runs and task streams.

3.3 Backend-Agnostic Wrapper and Augmented Operations

MemCon is a thin wrapper around *any* memory backend $\mathcal{M}_{\text{inner}}$ that exposes a two-method interface: `retrieve` (return K items for a query) and `store` (write the finished trajectory with its success label); an optional `maintain` hook unlocks the consolidation and eviction actions, and is silently no-op'd otherwise. On `retrieve`, the wrapper builds s_t, invokes the policy to pick $(\texttt{op}^*, \theta^*)$, calls $\mathcal{M}_{\text{inner}}$ with those parameters (unsupported parameters are dropped), and optionally augments the result with one of two augmented operations (described below); on `store`, it computes the episode reward, updates Q-values via Eq. (7), updates the plan index, and delegates persistence to $\mathcal{M}_{\text{inner}}$. This interface separates policy selection from storage and can be adapted to flat vector stores (Hong et al., 2024; Zhong et al., 2024), skill libraries (Wang et al., 2024a), summarisation-based memories (Qian et al., 2024), latent-memory systems (Fu et al., 2026), and graph-structured memories (Zhang et al., 2025). The two augmented operations target failure modes in long-horizon interactive tasks: **generalised plan injection** (a learnable MDP action, PlanInject) extracts the action sequence of a successful task of type g, replaces instance-specific identifiers (e.g. ``shelf 3'' → `[shelf]`) via a regex rewriter, stores the resulting template under g in a lightweight JSON index, and on future tasks of type g prepends the template to the backend's retrieval output (the rewriter degrades to identity on new vocabularies); **goal decomposition** (a deterministic heuristic, *not* an MDP action) uses a task-specific rule to handle composite tasks requiring the same primitive twice with two objects (e.g., "put two cellphones on the desk") by injecting "complete all steps for object 1 first, then repeat for object 2" and additionally retrieving the single-object template when one exists. Both augmentations concatenate text to retrieved context; their parsers and task-type mappings may require domain adaptation; the component ablation (§4.3, Table 2) shows that the learned controller alone already accounts for the majority of MemCon's gains, making both augmentations useful but optional.

Portability and effective actions. Backend-agnostic refers to the interface contract, not to identical functionality or demonstrated gains on every store. The main evaluation wraps G-Memory; Appendix D additionally evaluates five backend pairings. Unsupported parameters can collapse distinct presets into the same effective operation; independently exploring such duplicate arms can waste samples. Backends also differ in whether maintenance calls invoke an LLM. The zero-call claim concerns the controller itself, not the underlying retrieval, summarization, or maintenance implementation.

4 Experiments

We evaluate whether learned memory control improves task success, how its gains relate to token use, and how much improvement comes from the controller versus augmented operations.

4.1 Setup

We evaluate interactive decision-making on **ALFWorld** (Shridhar et al., 2021), **PDDL planning**, and **ScienceWorld** (Wang et al., 2022), and QA/web/tool use on **TriviaQA** (Joshi et al., 2017), **WebWalkerQA** (Wu et al., 2025), and **GAIA** (Mialon et al., 2023). Three runners (Lobster, LangGraph, Microsoft Agent-Framework) and three backbones (GPT-4.1-mini, Claude Sonnet-4, DeepSeek-V3.2) give 54 runner–backbone–benchmark configurations. Benchmark sizes, adapted baseline mechanisms, and prompts appear in Appendices A and F.

Table 1: **Main results with GPT-4.1-mini.** S/A (%) across six benchmarks and three runners. Eight adapted memory baselines and Empty are compared with MemCon on G-Memory. **Bold**: best; underline: second-best distinct value in each runner block, including ties. Avg. is the unweighted mean recomputed from the six displayed scores. SciWorld, Trivia, and WebQA denote ScienceWorld, TriviaQA, and WebWalkerQA. Other backbones and token costs appear in Appendix B.

		Interactive			QA / tool use			
Runner	**Memory**	**ALFWorld**	**PDDL**	**SciWorld**	**Trivia**	**WebQA**	**GAIA**	**Avg.**
Lobster	Empty	43.3	33.3	28.0	69.5	17.9	20.6	35.4
	G-Memory	59.7	31.7	34.0	66.5	19.0	21.2	38.7
	MetaGPT	55.2	33.3	33.0	**71.5**	18.2	16.4	37.9
	Voyager	53.0	31.7	31.0	**71.5**	18.4	16.4	37.0
	Generative	54.5	31.7	36.0	69.5	18.7	18.2	38.1
	ChatDev	32.8	28.3	24.0	69.5	17.7	18.2	31.8
	MemoryBank	57.5	33.3	28.0	69.5	16.3	17.6	37.0
	OAgent	48.5	28.3	34.0	70.5	17.4	19.4	36.4
	ExpBank	59.0	33.3	30.0	69.0	17.8	20.0	38.2
	MEMCON	**67.9**	**35.0**	**38.0**	**71.5**	**20.6**	**22.4**	**42.6**
LangGraph	Empty	31.3	30.0	39.0	67.5	18.6	19.4	34.3
	G-Memory	**68.7**	28.3	38.0	69.0	**20.9**	20.0	40.8
	MetaGPT	57.5	38.3	35.0	69.0	18.3	18.2	39.4
	Voyager	59.7	33.3	40.0	67.5	17.5	20.0	39.7
	Generative	56.0	35.0	36.0	66.0	18.0	20.0	38.5
	ChatDev	39.6	33.3	27.0	67.5	18.1	18.8	34.1
	MemoryBank	57.5	31.7	29.0	**70.0**	18.7	18.8	37.6
	OAgent	53.7	30.0	37.0	68.0	18.1	18.8	37.6
	ExpBank	55.2	35.0	**41.0**	68.0	18.2	20.0	39.6
	MEMCON	**68.7**	**40.0**	40.0	69.0	20.2	**22.4**	**43.4**
Agent-FW	Empty	38.1	33.3	26.0	69.0	18.1	16.4	33.5
	G-Memory	70.2	33.3	28.0	66.0	19.4	20.0	39.5
	MetaGPT	55.2	30.0	**38.0**	68.5	17.9	18.8	38.1
	Voyager	54.5	36.7	36.0	67.5	17.1	19.4	38.5
	Generative	50.0	35.0	37.0	68.0	18.3	18.8	37.9
	ChatDev	32.1	31.7	31.0	66.5	17.8	19.4	33.1
	MemoryBank	59.0	33.3	30.0	**69.5**	18.7	16.4	37.8
	OAgent	63.4	30.0	37.0	69.0	18.1	18.2	39.3
	ExpBank	52.2	36.7	35.0	66.5	17.5	17.0	37.5
	MEMCON	**71.0**	**40.0**	**38.0**	69.0	**20.9**	**23.0**	**43.7**

Comparisons and metrics. The reported MemCon configuration wraps **G-Memory**. Comparing MemCon with G-Memory therefore holds the inner storage design fixed; comparisons with the other memory modules also change the backend. Table 1 includes eight adapted memory baselines and the Empty control; the additional backbone tables also include LatentMem. These are shared-interface adaptations, not replications of the original systems. We report success rate / answer accuracy (**S/A**, %) and average LLM-input tokens per task (**Tok**). Each main-table cell is a single online deployment run. Appendix D adds an independent-stream summary; small main-table differences remain descriptive.

Online protocol. Memory and the policy accumulate experience across tasks, so these scores measure sequential adaptation rather than independent, identically distributed trials with a frozen store. Policy hyperparameters (α, γ, c) are selected per backbone on a held-out 30-task ALFWorld subset; priors, reward constants, and discretization thresholds are shared (Appendix C). In the original QA configuration, interactive features are constant and the effective state reduces to memory size,

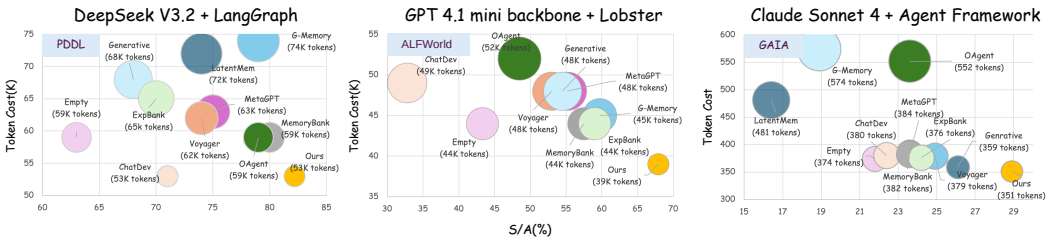


Figure 3: **Token cost vs. task success across memories.** Each panel plots, for one (framework, benchmark) pair, every memory baseline as a bubble whose horizontal position is mean S/A (%), vertical position is mean per-task input tokens (in thousands or units), and area is proportional to per-task token cost. MEMCON illustrates a favorable accuracy–cost trade-off in the displayed configurations. Table 7 reports the full GPT-4.1-mini token comparison; savings are relative to G-Memory and are not universal across baselines.

plan availability, and learning phase. QA results therefore test memory-state adaptation, without establishing that stuck-state recovery operates in those runs.

4.2 MAIN RESULTS

Accuracy across frameworks and backbones. On GPT-4.1-mini, MemCon is best or tied best in **14/18** cells: 6/6 under Lobster, 3/6 under LangGraph, and 5/6 under Agent-FW (11 strict wins). Its six-benchmark mean S/A is 42.6, 43.4, and 43.7, respectively, compared with 38.7, 40.8, and 39.5 for G-Memory. Lobster–ALFWorld improves from 59.7 to 67.9%; Agent-FW–PDDL improves from 33.3 to 40.0%. The exceptions also matter: under LangGraph, MemCon trails the best adapted baseline on ScienceWorld, TriviaQA, and WebWalkerQA. These results favor adaptive control on average without implying dominance in every setting.

The same pattern extends to the other backbones (Appendix B). MemCon leads **15/18** cells on Sonnet-4 and **15/18** on DeepSeek-V3.2. On Sonnet-4, the three deficits to the best baseline are 0.2–0.4 points; GAIA improves over the strongest alternative by 0.8, 2.6, and 2.8 points across runners. On DeepSeek-V3.2, MemCon leads all nine interactive cells; the three exceptions are GAIA, where LatentMem leads by 0.3–0.4 points. Across all backbones, the reported scores are best or tied best in 44/54 cells. Independent-stream results for selected benchmarks are reported in Appendix D.

Accuracy and token use. MemCon reduces input-token use relative to the same G-Memory backend in all 18 GPT-4.1-mini cells (Table 7). On Lobster–ALFWorld, accuracy rises by 8.2 points while input tokens fall from 45K to 39K (13.3%); on Agent-FW–ALFWorld, accuracy rises from 70.2 to 71.0% while tokens fall from 43K to 37K (14.0%). Averaging the six displayed benchmark costs gives approximate reductions of 10.9%, 11.0%, and 24.1% for Lobster, LangGraph, and Agent-FW. These are descriptive averages over heterogeneous tasks, not pooled workload costs. MemCon is not always the cheapest memory: several simpler QA baselines use fewer tokens, and some no-memory runs are cheaper. The evidence supports improved efficiency relative to G-Memory rather than a universal minimum-cost claim.

Generalization to QA and tool use. On GPT-4.1-mini GAIA, MemCon reaches 22.4–23.0% versus 20.0–21.2% for G-Memory, while TriviaQA results are tied with or slightly below the best adapted baseline in several settings. For 165 GAIA tasks, a 1.2-point difference corresponds to approximately two additional correct answers. Thus, the QA evidence is consistent with useful control over retrieval even with a reduced state, but does not establish a large or statistically stable gain. No backbone weights are updated during these runs.

4.3 COMPONENT ANALYSIS AND SENSITIVITY

Table 2 reports the complete five-variant component ablation. The learned UCB controller improves ALFWorld from 59.7 to 64.9%, accounting for 5.2 of the full 8.2-point gain (about 63%). The plan-injection and goal-decomposition configurations reach 66.4 and 67.2%, and the full system reaches 67.9%. On GAIA, all controller variants achieve 22.4%, compared with 21.2% for the static

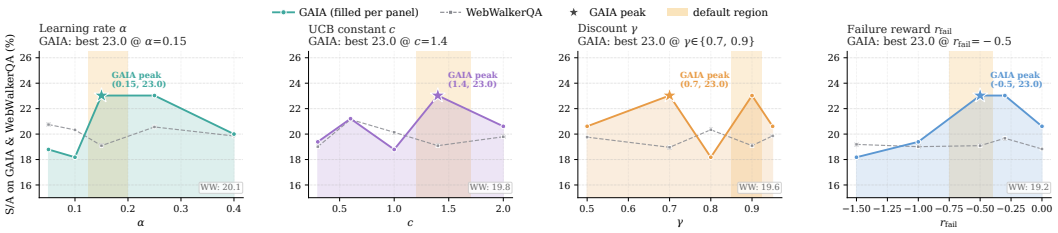


Figure 4: **Single-knob sensitivity (continuous policy hyperparameters).** Each panel sweeps one parameter—learning rate α, UCB coefficient c, discount γ, or the signed failure reward $-r_{\text{fail}}$—with the others fixed. Solid colored lines show GAIA S/A; gray dashed lines show WebWalkerQA. Shading marks the default region and stars mark GAIA peaks. Backbone: GPT-4.1-mini; runner: Lobster.

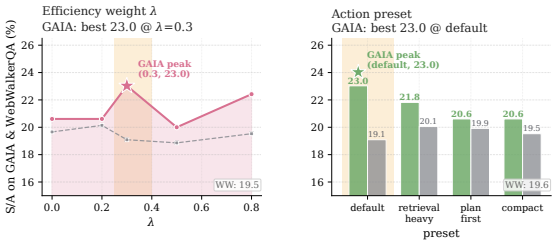


Figure 5: **Efficiency weight and action preset.** Left: varying λ favors the default 0.3; removing or overweighting the bonus reduces GAIA accuracy. Right: the default nine-action preset has the highest GAIA score; retrieval-heavy has the highest WebWalkerQA score.

Variant	ALFWorld	GAIA
Static backend	59.7	21.2
+ learned UCB	64.9	22.4
+ plan injection (g)	66.4	22.4
+ goal decomposition	67.2	22.4
+ all (= MemCon)	**67.9**	**22.4**

Table 2: **Component ablation on GPT-4.1-mini.** Lobster–ALFWorld success and GAIA S/A (%) for all five reported variants. The learned controller supplies the largest single gain; the full configuration further improves ALFWorld.

backend. Thus, learned access control contributes without relying on the augmented operations, while the full configuration offers additional gains on interactive tasks.

Figures 4 and 5 jointly show all six sensitivity panels. Intermediate policy settings perform well; removing the efficiency bonus or giving it excessive weight reduces accuracy by roughly two points. These local sweeps hold other parameters fixed and complement the component comparison.

Additional experiments. Appendix D extends the evaluation to five memory backends and additional adaptive-access baselines, examining how the controller performs beyond the main G-Memory pairing. The 400-task experiments study memory maintenance over longer deployments, where consolidation and forgetting can influence subsequent tasks. Additional analyses of state design, transfer, and initialization examine the sensitivity and transferability of the controller. A five-seed evaluation across independent task streams complements the single-deployment results by assessing variation across runs. Together, these studies complement the main accuracy and token-cost comparisons with a broader assessment of backend portability, long-horizon behavior, and deployment variability.

5 Conclusion

We introduced MemCon, a framework that treats agent memory as a controlled decision process. A lightweight online UCB controller chooses memory operations and parameters from task and memory state while keeping the LLM frozen. Across the reported benchmarks, frameworks, and backbones, MemCon improves average success. On GPT-4.1-mini, it also reduces input-token use relative to G-Memory. The component analysis attributes most of the ALFWorld gain to learned control. Together with the additional backend and long-stream experiments, these findings motivate separating memory storage from access policy and improving credit assignment for long-term maintenance.

AI USE STATEMENT

In this work, generative AI tools were employed to support method implementation and translation. We also used these tools to assist with creating and modifying software code, summarizing and analyzing existing literature, and retrieving or searching for relevant information. Generative AI tools were not used to generate or refine research hypotheses, design or evaluate research methodologies or experiments, clean or restructure datasets, conduct qualitative or thematic data analysis, generate synthetic data, develop theoretical models or conceptual frameworks. These applications were not relevant to the present work. All code, written content, and references were independently reviewed. The authors take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the assistance of generative AI tools.

ETHICS STATEMENT

This paper primarily focuses on agent systems for mathematical research. There are several potential societal consequences of our work, none of which we feel must be specifically highlighted here.

REPRODUCIBILITY STATEMENT

The system developed in this work is publicly accessible online, and references are provided for all baseline methods considered in our experiments. Further technical details regarding the implementation and experimental setup are provided in the Appendix.

REFERENCES

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002. doi: 10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

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. URL https://arxiv.org/abs/2504.19413.

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

Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. LatentMem: Customizing latent memory for multi-agent systems. *arXiv preprint arXiv:2602.03036*, 2026. URL https://arxiv.org/abs/2602.03036v2.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246, 2006. doi: 10.1613/jair.1705. URL https://doi.org/10.1613/jair.1705.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.00352.

Zhanghao Hu, Qinglin Zhu, Runcong Zhao, Di Liang, Hanqi Yan, Yulan He, and Lin Gui. Beyond RAG for agent memory: Retrieval by decoupling and aggregation. *arXiv preprint arXiv:2602.02007*, 2026. URL https://arxiv.org/abs/2602.02007.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017. URL https://aclanthology.org/P17-1147/.

LangChain Team. LangGraph. LangChain Blog, 2024. URL https://www.langchain.com/blog/langgraph.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. doi: 10.1017/9781108571401. URL https://doi.org/10.1017/9781108571401.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 661–670, 2010. doi: 10.1145/1772690.1772758. URL https://doi.org/10.1145/1772690.1772758.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023. URL https://arxiv.org/abs/2304.11477.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. SimpleMem: Efficient lifelong memory for LLM agents. *arXiv preprint arXiv:2601.02553*, 2026. URL https://arxiv.org/abs/2601.02553.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. In *Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=xS6zx1aBI9.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. ProcMEM: Learning reusable procedural memory from experience via non-parametric PPO for LLM agents. *arXiv preprint arXiv:2602.01869v1*, 2026. URL https://arxiv.org/abs/2602.01869v1.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983.

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. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://aclanthology.org/2024.acl-long.810/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/7a92bcdede88c7afd108072faf5485c8-Abstract.html.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024a. URL https://arxiv.org/abs/2305.16291.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024b. doi: 10.1007/s11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022. URL https://aclanthology.org/2022.emnlp-main.775/.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c. URL https://arxiv.org/abs/2409.07429.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. WebWalker: Benchmarking LLMs in web traversal. *arXiv preprint arXiv:2501.07572*, 2025. URL https://arxiv.org/abs/2501.07572.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023. URL https://arxiv.org/abs/2309.07864.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-R1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://arxiv.org/abs/2210.03629.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025. URL https://arxiv.org/abs/2506.07398.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a. URL https://arxiv.org/abs/2602.02474.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 19149–19164, 2026b. doi: 10.18653/v1/2026.findings-acl.956. URL https://aclanthology.org/2026.findings-acl.956/.

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. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

A Experimental Setup Details

We detail the reported benchmark subsets, adapted memory modules, and tuning protocol.

A.1 Benchmarks

Interactive decision-making (long-horizon, multi-step). **ALFWorld** (Shridhar et al., 2021) contains 134 household tasks across 6 types (*put, clean, heat, cool, examine, puttwo*); it couples a TextWorld front-end with embodied AI2-THOR back-ends. **PDDL Planning** consists of 100 classical planning tasks spanning *blocksworld, barman, gripper, tyreworld*, closely related to PlanBench (Valmeekam et al., 2023; Liu et al., 2023; Helmert, 2006). **ScienceWorld** (Wang et al., 2022) contains 100 elementary-science experimental tasks (boiling water, melting ice, finding animals by property, etc.).

Knowledge / web / tool-use QA. **TriviaQA** (Joshi et al., 2017) (200 questions, short-answer open-domain QA); **WebWalkerQA** (Wu et al., 2025) (200 multi-hop web traversal questions requiring site navigation and information extraction); **GAIA** (Mialon et al., 2023) (165 general-AI-assistant questions spanning tool use, web search, and document reading).

A.2 Agent Frameworks

The runners share task loaders, tools, and evaluation protocols. **Lobster** calls an OpenAI-compatible chat API directly; **LangGraph** (LangChain Team, 2024) provides graph-structured workflows; **Microsoft Agent-Framework** provides pipeline orchestration. Interactive tasks use a 30-step horizon; QA uses benchmark-default budgets. Temperature and tool protocols are shared across memory comparisons.

A.3 Memory Baselines

The table labels identify adapted modules on a shared retrieve/store interface: **MetaGPT** uses vector retrieval over trajectories; **Voyager** summarizes trajectories before cosine-similarity retrieval; **Generative** uses LLM relevance re-ranking; **ChatDev** summarizes phases every 10 steps without cross-task retrieval; **MemoryBank** adds temporal decay; **OAgent** induces rules by comparing successful and failed trials; **ExperienceBank** uses LLM-scored experience retrieval; and **G-Memory** combines vector retrieval, graph traversal, and scored insight rules. The additional backbone tables also include a **LatentMem**-labeled module. These labels denote experimental adaptations rather than verified reproductions of the cited systems, particularly for OAgent and LatentMem, whose original descriptions do not establish implementation equivalence. The **Empty** control disables memory. MemCon wraps G-Memory, making this pair the controlled comparison that holds the storage design fixed.

A.4 Hyperparameter Selection Protocol

MemCon uses the default nine-action set, warm-start priors, and the reward in Eq. (5). The per-backbone policy settings (α, γ, c) are chosen by grid search on a held-out 30-task ALFWorld subset (Lobster); priors, reward constants, and bins are shared across backbones. Appendix C lists the values. The Q-table and plan index persist across tasks. The main comparison tables use one online deployment stream per configuration; Appendix D adds the five-seed summary.

B PER-BACKBONE S/A TABLES AND TOKEN COST

This appendix reports the full per-backbone S/A tables for the two backbones omitted from the main paper (Claude Sonnet-4 and DeepSeek-V3.2) and provides the available GPT-4.1-mini average per-task token costs (Tok). Tables 5 and 6 are synthetic layout previews, excluded from the empirical comparisons.

B.1 CLAUDE SONNET-4 (S/A ONLY)

Table 3: **Claude Sonnet-4 results.** S/A (%) on the same six benchmarks. This table adds the adapted LatentMem-labeled module to the baselines in Table 1. **Bold**: best; underline: second-best distinct value, including ties. Avg. is recomputed from displayed scores.

		Interactive			QA / tool use			
Runner	**Memory**	**ALFWorld**	**PDDL**	**SciWorld**	**Trivia**	**WebQA**	**GAIA**	**Avg.**
Lobster	Empty	14.2	18.0	19.0	81.5	17.1	24.2	29.0
	G-Memory	31.3	58.0	25.0	82.0	16.3	20.6	38.9
	MetaGPT	6.7	8.0	12.0	82.5	15.9	24.9	25.0
	Voyager	10.4	14.0	10.0	81.5	16.7	24.9	26.3
	Generative	6.7	15.0	15.0	81.5	16.4	25.4	26.7
	ChatDev	11.9	17.0	14.0	81.5	16.6	26.7	28.0
	MemoryBank	9.0	12.0	8.0	81.5	16.5	22.4	24.9
	OAgent	28.4	42.0	27.0	81.5	17.4	24.2	36.8
	ExpBank	5.2	11.0	5.0	81.5	16.2	23.6	23.8
	LatentMem	30.6	67.0	**82.0**	81.5	15.8	17.6	49.1
	MEMCON	**32.3**	**68.9**	81.6	**84.4**	**19.4**	**27.5**	**52.4**
LangGraph	Empty	17.2	15.0	20.0	82.5	15.9	25.4	29.3
	G-Memory	46.3	33.0	37.0	81.5	17.4	19.4	39.1
	MetaGPT	16.4	24.0	21.0	**83.0**	15.4	23.0	30.5
	Voyager	22.4	22.0	33.0	82.5	17.1	25.4	33.7
	Generative	17.9	26.0	31.0	82.0	16.6	21.8	32.6
	ChatDev	12.7	28.0	20.0	82.5	16.1	22.4	30.3
	MemoryBank	26.9	25.0	39.0	82.0	15.8	22.4	35.2
	OAgent	53.0	33.0	58.0	80.5	16.9	17.6	43.2
	ExpBank	17.9	22.0	21.0	82.0	16.3	23.0	30.4
	LatentMem	53.7	61.0	68.0	82.5	18.0	23.0	51.0
	MEMCON	**55.4**	**63.2**	**69.3**	82.8	**19.2**	**28.0**	**53.0**
Agent-FW	Empty	12.7	21.0	17.0	83.0	17.1	21.8	28.8
	G-Memory	20.9	23.0	33.0	81.5	**17.5**	18.8	32.5
	MetaGPT	15.7	19.0	19.0	81.0	**17.5**	23.6	29.3
	Voyager	10.4	14.0	14.0	81.0	9.5	24.9	25.6
	Generative	20.2	20.0	26.0	82.0	17.2	26.1	31.9
	ChatDev	12.7	16.0	19.0	82.0	15.6	22.4	28.0
	MemoryBank	11.9	20.0	20.0	81.5	**17.5**	23.6	29.1
	OAgent	23.9	30.0	28.0	81.5	10.2	23.6	32.9
	ExpBank	6.0	6.0	7.0	82.0	16.4	24.2	23.6
	LatentMem	58.2	70.0	66.0	81.5	15.1	16.4	51.2
	MEMCON	**60.6**	**70.7**	**67.1**	**84.3**	17.1	**28.9**	**54.8**

B.2 DEEPSEEK-V3.2 (S/A ONLY)

Table 4: **DeepSeek-V3.2 results.** S/A (%) on the same six benchmarks, including the adapted LatentMem-labeled module. **Bold**: best; underline: second-best distinct value, including ties. Avg. is recomputed from displayed scores.

		Interactive			QA / tool use			
Runner	**Memory**	**ALFWorld**	**PDDL**	**SciWorld**	**Trivia**	**WebQA**	**GAIA**	**Avg.**
Lobster	Empty	64.2	63.0	67.0	75.5	17.6	28.5	52.6
	G-Memory	79.8	69.0	77.0	73.5	18.5	24.9	57.1
	MetaGPT	75.4	73.0	76.0	73.0	18.0	27.9	57.2
	Voyager	84.3	78.0	86.0	72.0	19.0	27.3	61.1
	Generative	82.8	79.0	88.0	73.5	21.0	25.4	61.6
	ChatDev	64.9	69.0	69.0	71.5	19.0	29.1	53.8
	MemoryBank	81.3	81.0	84.0	75.5	20.1	27.3	61.5
	OAgent	82.8	76.0	84.0	67.0	17.2	27.3	59.1
	ExpBank	76.9	73.0	78.0	75.0	18.5	26.7	58.0
	LatentMem	77.6	72.0	80.0	71.0	17.2	**29.9**	58.0
	MEMCON	**86.7**	**83.6**	**90.2**	**77.8**	**22.0**	29.6	**65.0**
LangGraph	Empty	66.4	63.0	66.0	71.5	18.9	21.8	51.3
	G-Memory	84.3	79.0	83.0	71.5	16.4	29.1	60.6
	MetaGPT	88.8	75.0	85.0	72.5	19.3	30.3	61.8
	Voyager	80.6	74.0	80.0	73.5	18.1	27.3	58.9
	Generative	79.8	68.0	80.0	71.5	19.5	26.1	57.5
	ChatDev	64.2	71.0	62.0	76.0	19.4	28.5	53.5
	MemoryBank	84.3	80.0	88.0	70.5	17.2	28.5	61.4
	OAgent	88.8	79.0	91.0	69.5	18.1	26.7	62.2
	ExpBank	82.1	70.0	78.0	72.5	17.7	28.5	58.1
	LatentMem	86.6	74.0	85.0	70.5	17.9	**33.1**	61.2
	MEMCON	**89.8**	**82.2**	**94.0**	**77.5**	**22.5**	32.8	**66.5**
Agent-FW	Empty	62.7	68.0	68.0	72.0	19.9	27.3	53.0
	G-Memory	77.6	69.0	80.0	71.0	20.5	15.8	55.7
	MetaGPT	81.3	75.0	83.0	75.0	17.9	24.9	59.5
	Voyager	81.3	83.0	86.0	73.5	18.1	27.3	61.5
	Generative	80.6	80.0	83.0	74.0	18.6	24.9	60.2
	ChatDev	61.9	63.0	65.0	73.5	18.3	17.0	49.8
	MemoryBank	82.8	83.0	88.0	74.0	19.4	27.9	62.5
	OAgent	82.1	74.0	85.0	74.5	16.6	25.4	59.6
	ExpBank	80.6	82.0	87.0	72.5	19.1	25.4	61.1
	LatentMem	83.6	76.0	89.0	75.0	16.4	**31.4**	61.9
	MEMCON	**85.4**	**84.2**	**91.3**	**77.8**	**21.5**	31.0	**65.2**

B.3 GPT-6-LUNA

Table 5: **GPT-6-Luna** Each S/A (%) is $100k/N$ for an integer success count k and the task count N below, rounded to one decimal. Avg. is the unweighted mean of the six displayed scores. Bold/underline mark the highest/second-highest distinct placeholder values, including ties.

		Interactive			QA / tool use			
Runner	**Memory**	**ALFWorld**	**PDDL**	**SciWorld**	**Trivia**	**WebQA**	**GAIA**	**Avg.**
	Tasks (N)	134	100	100	200	200	165	—
Lobster	Empty	69.4	68.0	72.0	<u>78.5</u>	21.0	<u>32.1</u>	56.8
	G-Memory	84.3	73.0	83.0	76.5	22.0	26.7	60.9
	MetaGPT	79.9	77.0	79.0	77.0	22.0	30.9	61.0
	Voyager	<u>88.8</u>	84.0	89.0	76.5	22.5	30.3	65.2
	Generative	87.3	83.0	<u>92.0</u>	77.0	<u>25.0</u>	27.3	65.3
	ChatDev	70.1	75.0	75.0	74.5	22.5	30.9	58.0
	MemoryBank	86.6	<u>85.0</u>	89.0	78.0	23.5	30.3	<u>65.4</u>
	OAgent	<u>88.8</u>	80.0	89.0	70.5	21.0	30.9	63.4
	ExpBank	82.8	78.0	81.0	78.0	22.5	30.3	62.1
	LatentMem	83.6	77.0	86.0	75.5	19.5	<u>32.1</u>	62.3
	MEMCON	**91.0**	**88.0**	**96.0**	**82.0**	**25.5**	**33.3**	**69.3**
LangGraph	Empty	71.6	69.0	71.0	75.0	22.0	24.2	55.5
	G-Memory	88.8	<u>85.0</u>	89.0	74.5	20.5	32.7	65.1
	MetaGPT	93.3	79.0	90.0	75.0	23.5	32.7	65.6
	Voyager	86.6	79.0	85.0	76.5	22.5	30.3	63.3
	Generative	84.3	71.0	86.0	74.5	<u>24.0</u>	28.5	61.4
	ChatDev	69.4	77.0	67.0	<u>79.5</u>	23.5	30.3	57.8
	MemoryBank	89.6	84.0	92.0	73.5	20.0	31.5	65.1
	OAgent	<u>94.8</u>	<u>85.0</u>	<u>95.0</u>	74.0	21.0	29.7	<u>66.6</u>
	ExpBank	85.8	74.0	82.0	76.5	20.5	31.5	61.7
	LatentMem	91.0	78.0	90.0	73.5	21.5	<u>36.4</u>	65.1
	MEMCON	**95.5**	**88.0**	**98.0**	**81.0**	**26.5**	**37.0**	**71.0**
Agent-FW	Empty	67.9	73.0	73.0	76.0	<u>24.0</u>	29.7	57.3
	G-Memory	84.3	74.0	84.0	74.0	<u>24.0</u>	18.8	59.9
	MetaGPT	87.3	80.0	87.0	<u>78.0</u>	21.5	28.5	63.7
	Voyager	87.3	<u>87.0</u>	90.0	76.5	21.0	29.7	65.3
	Generative	86.6	85.0	87.0	<u>78.0</u>	22.5	27.9	64.5
	ChatDev	67.2	68.0	69.0	76.5	21.5	20.6	53.8
	MemoryBank	<u>88.8</u>	<u>87.0</u>	<u>93.0</u>	77.5	23.5	30.3	<u>66.7</u>
	OAgent	<u>88.8</u>	79.0	90.0	77.5	20.0	29.1	64.1
	ExpBank	85.8	<u>87.0</u>	92.0	76.0	22.0	29.1	65.3
	LatentMem	88.1	80.0	<u>93.0</u>	77.5	20.0	<u>35.2</u>	65.6
	MEMCON	**90.3**	**90.0**	**94.0**	**82.0**	**25.0**	**35.8**	**69.5**

B.4 QWEN-3.8-27B

Table 6: **Qwen-3.8-27B**. Each S/A (%) is $100k/N$ for an integer success count k and the task count N below, rounded to one decimal. Avg. is the unweighted mean of the six displayed scores. Bold/underline mark the highest/second-highest distinct placeholder values, including ties.

		Interactive			QA / tool use			
Runner	**Memory**	**ALFWorld**	**PDDL**	**SciWorld**	**Trivia**	**WebQA**	**GAIA**	**Avg.**
	Tasks (N)	134	100	100	200	200	165	—
Lobster	Empty	65.7	65.0	69.0	<u>76.5</u>	18.5	29.7	54.1
	G-Memory	82.1	70.0	79.0	74.5	19.5	25.5	58.4
	MetaGPT	77.6	74.0	77.0	74.5	19.0	28.5	58.4
	Voyager	<u>86.6</u>	80.0	87.0	73.5	20.0	28.5	62.6
	Generative	84.3	80.0	<u>90.0</u>	74.5	<u>22.5</u>	26.1	<u>62.9</u>
	ChatDev	67.2	71.0	71.0	72.5	20.0	29.7	55.2
	MemoryBank	83.6	<u>82.0</u>	85.0	<u>76.5</u>	21.5	27.9	62.8
	OAgent	85.1	77.0	86.0	68.0	18.0	28.5	60.4
	ExpBank	79.1	75.0	79.0	76.0	20.0	27.9	59.5
	LatentMem	79.9	74.0	82.0	72.5	18.0	<u>30.3</u>	59.5
	MEMCON	**88.1**	**85.0**	**92.0**	**79.5**	**23.5**	**30.9**	**66.5**
LangGraph	Empty	67.9	65.0	68.0	73.0	20.5	22.4	52.8
	G-Memory	86.6	<u>81.0</u>	85.0	72.5	18.0	30.3	62.2
	MetaGPT	90.3	76.0	86.0	73.5	20.5	30.9	62.9
	Voyager	82.1	75.0	82.0	74.5	19.5	27.9	60.2
	Generative	81.3	69.0	82.0	72.5	<u>21.0</u>	26.7	58.8
	ChatDev	66.4	73.0	63.0	<u>77.0</u>	<u>21.0</u>	29.1	54.9
	MemoryBank	85.8	<u>81.0</u>	90.0	71.5	18.5	29.7	62.8
	OAgent	<u>91.0</u>	<u>81.0</u>	<u>93.0</u>	71.0	19.0	27.3	<u>63.7</u>
	ExpBank	83.6	71.0	80.0	74.0	18.5	29.7	59.5
	LatentMem	88.1	75.0	87.0	72.0	19.0	<u>33.9</u>	62.5
	MEMCON	**91.8**	**84.0**	**95.0**	**79.0**	**24.0**	**34.5**	**68.1**
Agent-FW	Empty	64.9	70.0	69.0	73.0	<u>21.5</u>	27.9	54.4
	G-Memory	79.9	70.0	82.0	72.0	<u>21.5</u>	17.0	57.1
	MetaGPT	83.6	76.0	84.0	<u>76.0</u>	19.0	25.5	60.7
	Voyager	83.6	<u>85.0</u>	87.0	74.5	19.5	27.9	62.9
	Generative	82.8	82.0	85.0	75.0	20.0	25.5	61.7
	ChatDev	64.2	64.0	67.0	74.5	20.0	18.2	51.3
	MemoryBank	84.3	84.0	90.0	75.0	21.0	29.1	<u>63.9</u>
	OAgent	84.3	75.0	87.0	<u>76.0</u>	18.0	26.7	61.2
	ExpBank	82.1	83.0	89.0	74.0	20.0	26.1	62.4
	LatentMem	<u>85.8</u>	77.0	<u>91.0</u>	<u>76.0</u>	18.0	<u>32.7</u>	63.4
	MEMCON	**87.3**	**87.0**	**92.0**	**79.0**	**23.0**	**33.3**	**66.9**

B.5 TOKEN COST (TOK PER TASK)

Table 7 reports the average number of LLM-input tokens consumed per task for the GPT-4.1-mini main results (this appendix reports token costs for GPT-4.1-mini only). The token-cost vs. S/A trade-off is summarized visually in Figure 3 of the main paper.

Table 7: **GPT-4.1-mini input-token cost per task.** K denotes thousands of tokens. Avg. is the unweighted mean of the displayed costs, rounded to the nearest thousand. **Bold**: lowest; underline: second-lowest distinct value, including ties. MemCon reduces input-token use relative to G-Memory in all 18 configurations; simpler baselines can be cheaper.

		Interactive			QA / tool use			
Runner	**Memory**	**ALFWorld**	**PDDL**	**SciWorld**	**Trivia**	**WebQA**	**GAIA**	**Avg.**
Lobster	Empty	44K	**98K**	**31K**	58	153	**377**	**29K**
	G-Memory	45K	155K	44K	288	459	703	41K
	MetaGPT	48K	154K	33K	57	157	421	39K
	Voyager	48K	154K	39K	**56**	154	407	40K
	Generative	48K	148K	34K	57	155	382	38K
	ChatDev	49K	102K	32K	57	153	395	31K
	MemoryBank	44K	143K	38K	57	**151**	404	38K
	OAgent	52K	162K	36K	146	244	566	42K
	ExpBank	44K	154K	44K	57	154	399	40K
	MEMCON	**39K**	148K	**31K**	58	164	440	36K
LangGraph	Empty	51K	101K	**31K**	57	**152**	406	31K
	G-Memory	42K	157K	42K	267	434	720	40K
	MetaGPT	49K	156K	41K	57	156	398	41K
	Voyager	47K	145K	38K	57	155	387	38K
	Generative	49K	154K	42K	57	154	**384**	41K
	ChatDev	46K	**100K**	33K	**56**	154	411	**30K**
	MemoryBank	47K	137K	36K	57	154	410	37K
	OAgent	51K	161K	38K	242	309	477	42K
	ExpBank	45K	160K	42K	57	154	395	41K
	MEMCON	**41K**	142K	32K	61	172	456	36K
Agent-FW	Empty	46K	102K	33K	57	154	407	**30K**
	G-Memory	43K	154K	41K	300	428	691	40K
	MetaGPT	46K	168K	37K	57	152	407	42K
	Voyager	49K	140K	37K	57	152	411	38K
	Generative	49K	164K	39K	**56**	**150**	**356**	42K
	ChatDev	49K	**100K**	**32K**	**56**	152	405	**30K**
	MemoryBank	42K	143K	44K	57	151	389	38K
	OAgent	44K	157K	38K	222	236	470	40K
	ExpBank	46K	158K	38K	**56**	154	393	40K
	MEMCON	**37K**	111K	33K	**56**	186	462	**30K**

C HYPERPARAMETERS AND IMPLEMENTATION DETAILS

This appendix lists the default hyperparameters for the main comparisons; supplementary variants override them as described in Appendix D. Table 8 consolidates the constants used in §3–§4 into one place; Table 9 gives the structural action set $\mathcal{A}$ which has a different layout. These are the reported experimental settings; the table distinguishes per-backbone policy settings from shared constants.

Table 8: **Default MEMCON hyperparameters for the main comparisons.** The four groups—policy, warm-start priors, reward shape, and state discretisation/runtime—contain the policy and state constants. Policy hyperparameters (α, γ, c) are selected per backbone via a small grid search on a held-out 30-task ALFWorld split (Lobster); all other constants are shared across backbones.

Symbol	GPT-4.1-mini	Sonnet-4	DeepSeek-V3.2	Description
Policy hyperparameters (per backbone, used in Eqs. (6), (7))				
α (step size)	0.15	0.12	0.18	Terminal-return update step size
γ (discount)	0.9	0.92	0.88	Within-episode discount factor for reverse credit assignment
c (UCB coeff.)	1.4	1.2	1.6	UCB exploration coefficient
warm-start	enabled	enabled	enabled	Initialise unseen (s, a) with priors below
Warm-start priors $Q_0(s, a)$ for unseen (s, a) (same across backbones)				
RETRIEVE		$+0.5$		retrieval is usually helpful
PLANINJECT		$+0.3$		plans help when available
RE-RETRIEVE		$+0.1$		useful only when stuck
CONSOLIDATE		0.0		neutral
FORGET		-0.1		mildly risky
NOOP		-0.2		doing nothing usually hurts
Reward shape (Eq. (5))				
r_{succ}		1.0		success bonus
r_{fail}		0.5		failure penalty (subtracted)
λ		0.3		efficiency-bonus weight
$T_{\max}$		30 steps (interactive); benchmark default (QA)		step horizon
State discretisation bins $\phi(s)$ (used in ϕ for tabular Q)				
`step_phase`		early (<8), mid (8–17), late (≥ 18)		step-count bin for s^{task}
`learning_phase`		cold (task index ≤ 15); warm (task index >15)		task-index bin for s^{mem}
`is_stuck` trigger		2 consecutive identical actions		sets the stuck flag
held-objects bin		$\min(\texttt{hold}, 2)$		cap on objects-held count
locations bin		$\min(\lfloor \texttt{visited}/3 \rfloor, 4)$		coarse unique-location bin
mem-size bin		$\min(\lfloor \texttt{mem_size}/10 \rfloor, 5)$		coarse memory-size bin
Persistence and runtime				
`persist_path`		`./memcon/policy_q.json`		Q-table file
flush interval		every 5 updates		Q-table write frequency
LLM temperature		0 (interactive); benchmark default (QA)		sampling temperature
seed averaging		none in the main comparison tables		matches online-deployment setting

Discretised state key. The MDP state key used by the controller is

$$\phi(s) = \langle \texttt{goal_type}, \texttt{step_phase}, \texttt{is_stuck}, \min(\texttt{hold}, 2), \min(\lfloor \texttt{visited}/3 \rfloor, 4), \min(\lfloor \texttt{mem_size}/10 \rfloor, 5), \texttt{plan_available}, \texttt{learning_phase} \rangle. \tag{8}$$

with the bin definitions in Table 8. This coarse discretization bounds the representation size, but sample requirements depend on actual state revisitation and action gaps. In the original QA configuration, `goal_type`, `step_phase`, `is_stuck`, held objects, and locations are constant; only memory size, plan availability, and learning phase vary. Consequently, the effective QA state is smaller than the full tuple.

Action space. Table 9 lists the default action set $\mathcal{A}$ for the main comparisons (preset `"default"`). Additional presets (`"retrieval_heavy"`, `"plan_first"`, `"compact"`) are available in the released code; the action-preset ablation in §4.3 (right panel of Figure 5) shows that the default preset leads on GAIA and on the two-benchmark average, while retrieval-heavy leads on WebWalkerQA.

Table 9: Default action space $\mathcal{A}$ for the main comparisons. Backends that do not implement maintenance hooks silently no-op on CONSOLIDATE/FORGET (§3.3).

Idx	Operation	`top_k`	`insight_k`	`hop`
0	RETRIEVE (shallow)	1	3	1
1	RETRIEVE (medium)	2	5	1
2	RETRIEVE (deep)	3	8	2
3	PLANINJECT	1	3	–
4	RE-RETRIEVE (alt. query)	2	5	2
5	CONSOLIDATE	–	–	–
6	FORGET	–	–	–
7	RETRIEVE (insight-only)	1	2	0
8	NOOP	–	–	–

LLM, agent framework, and benchmark settings. Agent frameworks use their default configurations, modified only to share identical task loaders and evaluators. Interactive benchmarks: ALFWorld (134 tasks, 6 types), PDDL (100 tasks across blocksworld/barman/gripper/tyreworld), ScienceWorld (100 tasks). QA benchmarks: TriviaQA (200 questions), WebWalkerQA (200), GAIA (165). Each main-table number is a single realistic deployment run, with the Q-table persisted across tasks; LLM temperature, run protocol, and seed handling follow the values listed in Table 8.

D ADDITIONAL EXPERIMENTS

This appendix collects additional experiments reported in the rebuttal. S/A denotes success rate or answer accuracy in percent, and K denotes thousands of input tokens. The uncertainty convention for $\pm$ and the operational convergence criterion were not specified; those summaries and the stated resampling-test results are retained as reported.

D.1 MEMORY BACKENDS AND ADAPTIVE ACCESS POLICIES

The backend comparison holds the wrapper and its hyperparameters fixed while varying the inner store. Table 10 reports the unwrapped and wrapped results together with the number of behaviorally distinct supported actions, $|\mathcal{A}_{\text{eff}}|$. The reported gains are positive for each of the five backends, although the quality of the underlying store still substantially affects final performance.

Table 10: **Wrapping five memory backends.** Each benchmark cell gives the original backend $\rightarrow$ backend with MEMCON, followed by the reported gain in percentage points.

Inner backend	$\lvert\mathcal{A}_{\text{eff}}\rvert$	ALFWorld	SciWorld	GAIA
MetaGPT (flat vector)	6	55.2 $\rightarrow$ 59.7 (+4.5)	33.0 $\rightarrow$ 36.0 (+3.0)	16.4 $\rightarrow$ 17.6 (+1.2)
ChatDev (summary)	7	32.8 $\rightarrow$ 37.3 (+4.5)	24.0 $\rightarrow$ 27.0 (+3.0)	18.2 $\rightarrow$ 19.4 (+1.2)
Voyager (skills)	7	53.0 $\rightarrow$ 58.2 (+5.2)	31.0 $\rightarrow$ 34.0 (+3.0)	16.4 $\rightarrow$ 17.6 (+1.2)
MemoryBank (decay)	8	57.5 $\rightarrow$ 63.4 (+5.9)	28.0 $\rightarrow$ 32.0 (+4.0)	17.6 $\rightarrow$ 18.8 (+1.2)
G-Memory (graph)	9	59.7 $\rightarrow$ 67.9 (+8.2)	34.0 $\rightarrow$ 38.0 (+4.0)	21.2 $\rightarrow$ 22.4 (+1.2)
Mean gain		+5.7	+3.4	+1.2

Table 11 compares access policies with the same memory store and Lobster harness. The SimpleMem-style, xMemory-style, and MemGPT-style rows are adaptations of the corresponding access patterns, rather than reproductions of the complete systems. Query-adaptive retrieval and uncertainty-guided expansion improve the reported task scores over fixed retrieval; MEMCON attains the highest scores in this comparison with fewer input tokens.

Table 11: **Adaptive access with GPT-4.1-mini and Lobster.** Policies share the same store. Extra calls count LLM calls for memory-access control. Tok is the input-token count reported for this comparison.

Access policy	Extra calls	ALF	SciW	GAIA	Tok
G-Memory, fixed top-k	0	59.7	34.0	21.2	45K
Query-adaptive k (SimpleMem-style)	1	62.7	35.0	21.8	47K
Uncertainty-guided expansion (xMemory-style)	≥ 1	64.2	36.0	21.8	52K
LLM controller (MemGPT-style)	1	63.4	35.0	20.6	58K
MEMCON	0	**67.9**	**38.0**	**22.4**	**39K**

D.2 MAINTENANCE OVER A LONGER TASK STREAM

The 400-task comparison separates a static backend, a controller without maintenance, the full controller, and a variant that credits maintenance across subsequent tasks with a length-5 trace and discount $\eta = 0.8$. The full controller is one point below the no-maintenance variant in the first block, but the reported gap increases to eight points in the last block (Table 12). The cross-task-credit variant reaches 57.3% overall, compared with 56.8% for the full controller. These comparisons describe the behavior of the evaluated variants; observing later state changes alone does not assign future rewards to earlier maintenance decisions.

Table 12: **Success over a 400-task stream.** Results are shown in consecutive blocks of 100 tasks. The final column compares the full controller with its no-maintenance variant.

Task block	Static	No maintenance	Full	Cross-task credit	Full − no maintenance
1–100	44.0	47.0	46.0	46.0	−1.0
101–200	48.0	55.0	56.0	56.0	+1.0
201–300	47.0	57.0	61.0	62.0	+4.0
301–400	45.0	56.0	64.0	65.0	+8.0
Overall	46.0	53.8	56.8	57.3	+3.0

Table 13 records store size, the reported insight score, retrieval precision, token use, and maintenance frequency. At task 400, the full controller retains fewer insights and has higher reported precision@3 than either reference configuration. The frequency of consolidation or forgetting rises from 3.2% in the first block to 14.6% in the fourth block.

Table 13: **Store diagnostics for the long stream.** The first three rows describe the store or task cost at task 400; the last row compares maintenance frequency in blocks 1 and 4.

Metric	Static	No maintenance	Full
Insights / mean score	412 / 1.87	398 / 1.92	176 / 3.41
Retrieval precision@3	41.2%	43.8%	62.5%
Input tokens / task	68K	61K	44K
$P(\text{Consolidate} \cup \text{Forget})$	—	$0\% \to 0\%$	$3.2\% \to 14.6\%$

D.3 WARM-START INITIALIZATION

The initialization comparison includes the specified priors, zero initial values, and sign-flipped priors. The largest differences occur in the first 30 tasks; the reported late-stream rates are closer, while the cold and adversarial starts require more tasks to reach the reported convergence point.

Table 14: **Initialization on ALFWorld.** Success rates for the early and later portions of the 134-task stream, with overall S/A and reported tasks to convergence.

Initialization	Tasks 1–30	Tasks 31–134	Overall	Convergence
Warm start	46.7	74.0	67.9	30
Cold start ($Q_0 = 0$)	33.3	73.1	64.2	52
Adversarial priors	26.7	71.2	61.2	68
Static G-Memory	43.3	64.4	59.7	—

D.4 AGGREGATE POLICY BEHAVIOR

Table 15 reports the action frequencies for the 400-task stream and several state slices. Re-retrieval is more frequent in the stuck slice, plan injection is more frequent when a plan is available, and maintenance is more frequent in the larger-memory slice. These frequencies characterize the reported policy behavior, without identifying the causal contribution of an individual state coordinate or action.

Table 15: **Action frequencies by state slice (%).** Shal/Med/Deep are shallow, medium, and deep retrieval; Plan is plan injection; ReRet is re-retrieval; Maint combines consolidation and forgetting. Cold/warm denote task index ≤ 15/> 15.

State slice	Shal	Med	Deep	Plan	ReRet	Maint	NoOp
All decisions	18.4	21.7	15.2	22.6	9.8	6.1	6.2
Cold	24.1	18.6	11.3	8.2	11.5	9.4	16.9
Warm	16.9	22.5	16.2	26.4	9.4	5.2	3.4
Not stuck	19.6	22.8	15.7	23.9	4.3	6.4	7.3
Stuck	8.1	12.4	11.6	12.7	51.3	2.8	1.1
Plan unavailable	26.8	31.4	21.9	0.0	9.1	6.3	4.5
Plan available	8.3	9.9	7.1	59.4	10.6	3.9	0.8
Memory < 10	27.3	19.1	8.4	14.2	10.7	1.4	18.9
Memory ≥ 30	13.6	20.8	19.4	24.1	8.7	11.9	1.5

The four contrasts in Table 16 summarize the behavior patterns motivating the controller. The reported permutation test uses 10,000 samples. The contrasts are associations between state slices and selected actions; in particular, plan availability can also constrain which actions are feasible.

Table 16: **Reported state-slice contrasts.** Frequencies and contrasts are in percentage points. The final column reproduces the reported permutation-test values.

Action probability	Compared slices	Contrast	p
$P(\text{Shal} \cup \text{NoOp})$	Cold 41.0 / warm 20.3	$+20.7$	< 0.001
$P(\text{Plan})$	Available 59.4 / unavailable 0.0	$+59.4$	< 0.001
$P(\text{ReRet})$	Stuck 51.3 / not stuck 4.3	$+47.0$	< 0.001
$P(\text{Maint})$	Memory ≥ 30: 11.9 / < 10: 1.4	$+10.5$	< 0.001

D.5 PLAN GENERALIZATION BEYOND A FIXED VOCABULARY

The lexicon-free variant removes trailing integer indices, replaces task-argument tokens with slots, and collapses consecutive identical action templates. It improves the reported PDDL score while matching the lexicon-based result on ScienceWorld and GAIA (Table 17).

Table 17: **Plan-generalization variants.** PDDL tests vocabulary held out from the household-object lexicon; S/A is reported in percent.

Generalizer	ALFWorld	SciWorld	PDDL	GAIA
None (verbatim replay)	64.2	36.0	33.3	22.4
Lexicon-based	67.9	38.0	35.0	22.4
Lexicon-free	67.2	38.0	38.3	22.4

D.6 Feature instantiation and state granularity

In the original QA configuration, the interactive coordinates are constant: objects held and locations are zero, the stuck indicator is false, the step phase is early, and the goal type is "other." The remaining effective coordinates are memory size, plan availability, and learning phase. The richer feature variant uses answer-type keywords, tool-call counts, repeated tool calls, unique URLs, and open files to instantiate the same semantic roles for QA and tool use. Table 18 compares the number of visited states and task scores.

Table 18: **Richer state features for QA and tool use.** Original → richer-feature results. The ALFWorld code path is unchanged.

Benchmark	Visited states	S/A (%)
ALFWorld	$312 \rightarrow 312$	$67.9 \rightarrow 67.9$
TriviaQA	$9 \rightarrow 54$	$71.5 \rightarrow 72.0$
WebWalkerQA	$14 \rightarrow 96$	$20.6 \rightarrow 21.3$
GAIA	$11 \rightarrow 88$	$22.4 \rightarrow 23.6$

The discretization comparison varies the numbers of memory-size, location, and step-phase bins. Finer bins increase the number of visited states and the reported time to convergence on the standard stream. The 402-task fine-bin result is shown separately: its longer stream changes the evaluation conditions, so it does not establish an advantage over a default-bin controller run for the same duration. The final column in Table 19 gives the relative visited-state count, not a measured regret value.

Table 19: **Discretization granularity.** Bin counts are memory/location/step. Conv. is the reported tasks to convergence; NR means not reached. The last row uses a 402-task stream.

Granularity	Bins	$\lvert\Phi\rvert$	Conv.	ALF	SciW	GAIA	$\lvert\Phi\rvert/312$
Coarse	3/3/2	68	14	63.4	35.0	21.8	$0.22\times$
Default	6/5/3	312	30	67.9	38.0	22.4	$1.00\times$
Fine	12/9/5	1,140	78	65.7	36.0	21.8	$3.65\times$
Very fine	24/17/7	3,420	NR	61.2	33.0	20.6	$10.96\times$
Fine, 402 tasks	12/9/5	1,206	91	69.4	39.0	—	$3.87\times$

D.7 Including previous memory operations in the state

Adding the last operation yields slightly higher reported interactive scores, while adding a retrieval counter or a full operation-count histogram increases the state space further and lowers scores in this comparison. The largest state representation does not reach the reported convergence point within the stream.

Table 20: **History-augmented state representations.** The base state has eight coordinates. The retrieval count is binned; the histogram records counts by operation.

State design	$\lvert\Phi\rvert$	ALF	SciW	GAIA	Conv.
Base ϕ	312	67.9	38.0	22.4	30
$\phi \oplus$ last operation	704	68.7	39.0	22.4	48
$\phi \oplus$ last operation $\oplus$ retrieval count	1,486	67.2	37.0	21.8	72
$\phi \oplus$ full operation-count histogram	5,930	62.7	34.0	20.6	NR

D.8 TASK-ORDER AND LLM-SAMPLING VARIATION

The five-seed comparison resamples both task order and LLM sampling, reflecting the dependence of online learning on the arrival sequence. Table 21 reports the aggregate scores and the paired-bootstrap comparison against the best baseline in this table. The reported ALFWorld variability components are $\sigma_{\text{stream-order}} = 1.8$ and $\sigma_{\text{LLM-sampling}} = 1.1$. The GAIA comparison has reported $p = 0.087$, so it does not meet a 0.05 threshold.

Table 21: **Five-seed summary with task-order and sampling variation.** S/A (%) and the supplied $\pm$ summaries. The final row gives the reported paired-bootstrap comparison with the best baseline shown.

Memory	ALFWorld	SciWorld	GAIA
G-Memory	59.7 ± 2.8	34.0 ± 3.1	21.2 ± 1.4
MemoryBank	57.5 ± 3.4	28.0 ± 2.7	17.6 ± 1.6
ExperienceBank	59.0 ± 3.0	30.0 ± 2.9	20.0 ± 1.5
MEMCON	67.9 ± 2.1	38.0 ± 2.4	22.4 ± 1.2
Paired-bootstrap p	0.004	0.011	0.087

D.9 CONTROLLER TRANSFER ACROSS DOMAINS

Table 22 compares learning from scratch with transferring an ALFWorld controller, either freezing its values or continuing online adaptation. An additional transferred variant omits goal type from the state key. The reported adapted controllers reach the stated convergence point in fewer target-domain tasks than the from-scratch controller. The goal-type-agnostic variant obtains the highest reported ScienceWorld score and matches the adapted full-state controller on PDDL and GAIA.

Table 22: **Transfer from ALFWorld.** S/A (%) and reported tasks to convergence on each target benchmark. A zero denotes the frozen-transfer setting; dashes denote non-learning references.

	SciWorld		PDDL		GAIA	
Setting	**S/A**	**Conv.**	**S/A**	**Conv.**	**S/A**	**Conv.**
No memory (Empty)	28.0	—	33.3	—	20.6	—
Static G-Memory	34.0	—	31.7	—	21.2	—
MEMCON from scratch	38.0	34	35.0	29	22.4	41
Zero-shot transfer (Q frozen)	35.0	0	33.3	0	21.8	0
Transfer + online adaptation	39.0	17	36.7	15	23.0	26
Goal-type-agnostic ϕ, transfer + adaptation	40.0	12	36.7	11	23.0	19

The additional slice-level observation reports re-retrieval in 51.3% of stuck states on ALFWorld and 47.4% under frozen transfer to ScienceWorld. This is an observed frequency comparison, rather than evidence that unmatched entries in a full-tuple tabular representation automatically share values across domains.

E IDEALIZED BANDIT ANALYSIS AND ITS SCOPE

This appendix separates a stationary-bandit reference model from the implemented controller. The reference model explains why a compact state and action space can limit exploration cost. It does not provide a regret or convergence guarantee for MEMCON in an evolving memory environment. In particular, the implementation uses a constant-step-size estimate and episode-end feedback, rather than the sample means and immediately available observations in the classical UCB1 analysis.

Bounded episode returns. With the reward constants in Table 8, the terminal reward satisfies $r_i \in [-0.5, 1.3]$. A memory decision at index j in an episode containing m_i decisions receives

$$G_{i,j} = \gamma^{m_i - j - 1} r_i, \qquad 0 < \gamma \leq 1, \quad 0 \leq j < m_i. \tag{9}$$

Multiplication by a number in $[0, 1]$ moves a reward toward zero, so every such return remains in $[-0.5, 1.3]$. We therefore use the uniform range bound $W = 1.8$; reverse discounting does not increase this range.

A stationary reference model. Fix a finite set of state keys Φ and an action set $\mathcal{A}$. For each pair (ϕ, a), assume that successive observed returns are independent draws from a fixed distribution supported on an interval of width at most W, with mean $\mu(\phi, a)$. Equivalently, one can associate an independent reward sequence with each pair and reveal its next entry when that pair is selected. Assume that previous returns are available before the next selection. Define

$$\mu^*(\phi) = \max_{a \in \mathcal{A}} \mu(\phi, a), \qquad \Delta(\phi, a) = \mu^*(\phi) - \mu(\phi, a).$$

Here the comparator maximizes a stationary per-decision return at a fixed state. It is not the optimal policy of the original sequential memory management problem, where actions can change future states and rewards.

Initialize each action once at a given state. At local decision t, let $N_a(t-1)$ be its observed sample count and $\widehat{\mu}_a(t-1)$ the arithmetic mean of those samples. The reference UCB1 rule is

$$a_t \in \arg\max_{a \in \mathcal{A}} \left\{ \widehat{\mu}_a(t-1) + W \sqrt{\frac{2 \ln t}{N_a(t-1)}} \right\}. \tag{10}$$

Rescaling bounded returns to $[0, 1]$ and applying the standard UCB1 argument (Auer et al., 2002; Lattimore & Szepesvári, 2020) gives, for a deterministic local horizon $n \geq |\mathcal{A}|$,

$$\mathbb{E}[\mathcal{R}_n(\phi)] \leq \sum_{a : \Delta(\phi, a) > 0} \left\{ \frac{8 W^2 \ln n}{\Delta(\phi, a)} + \left(1 + \frac{\pi^2}{3}\right) \Delta(\phi, a) \right\}, \tag{11}$$

where $\mathcal{R}_n(\phi) = \sum_{t=1}^{n} \Delta(\phi, a_t)$. Briefly, after approximately $8W^2 \ln n / \Delta^2$ samples of a suboptimal action, selecting it requires a deviation of an empirical mean beyond its confidence radius. Hoeffding's inequality and a union bound make the remaining expected number of such selections finite. The bound applies to the sample-mean rule above with the specified exploration coefficient. It does not automatically apply after replacing the sample mean by an exponentially weighted estimate or changing the coefficient.

Summing statewise exploration costs motivates a small number of contexts and actions. It does not determine an optimal discretization: coarsening can merge states whose useful actions differ, creating approximation error absent from the bound. Similarly, two action configurations that collapse to the same backend behavior have the same mean, but need not have zero gap. If that behavior is suboptimal, separate aliases can each incur exploration cost. Sharing their statistics would require an explicit identification of equivalent actions.

What the constant-step-size update estimates. Consider one fixed state–action pair in isolation, and index its updates by k. Under the additional idealization that its observed returns $G_1, G_2, \ldots$ are independent and identically distributed, with mean μ and variance $\sigma^2 \leq W^2/4$, the update in Eq. (7) is

$$Q_{k+1} = (1 - \alpha) Q_k + \alpha G_{k+1}, \qquad 0 < \alpha < 1. \tag{12}$$

For a deterministic initialization Q_0, expanding this recursion yields

$$\mathbb{E}[Q_k] - \mu = (1-\alpha)^k (Q_0 - \mu), \tag{13}$$

$$\mathrm{Var}(Q_k) = \frac{\alpha\sigma^2}{2-\alpha}\left[1 - (1-\alpha)^{2k}\right]. \tag{14}$$

Consequently,

$$\mathbb{E}[(Q_k - \mu)^2] \le (1-\alpha)^{2k}(Q_0 - \mu)^2 + \frac{\alpha W^2}{4(2-\alpha)}. \tag{15}$$

The mean forgets its initialization geometrically, while nonzero return variance leaves a nonzero estimation-variance floor. This is a tracking estimate, not almost-sure convergence to a fixed value at constant α. The warm-start values in the implementation are initial estimates; they are not Bayesian pseudo-counts. Their direct contribution after k updates is $(1-\alpha)^k Q_0$.

For comparison, an arithmetic sample mean corresponds to a local step size $\alpha_k = 1/k$ and converges under independent stationary sampling when that pair is observed infinitely often. More general stochastic approximation results require suitable noise conditions and diminishing step sizes whose sum diverges and squared sum converges *on each pair's own update sequence*. Conditions on a global clock alone do not ensure these properties on a sparsely visited subsequence. UCB can explore actions when a state recurs; it cannot ensure that the environment revisits every state indefinitely.

Limits for sequential memory control. Several aspects of deployment violate the reference assumptions. The memory store and downstream policy evolve, so the return distribution for the same coarse state–action key can change. Several decisions in one episode share the same terminal reward, creating dependent feedback, and that feedback is unavailable until the task ends. The compact key is not established to be a sufficient Markov state. These facts prevent the stationary UCB1 bound from certifying the implemented controller or implying convergence within any fixed number of tasks.

Finally, Eq. (7) credits a maintenance action only with the current task's reward. A benefit appearing in a later task is not propagated back to that action. Including the resulting memory size or plan availability in later states can help later decisions, but does not repair this missing cross-task credit. Long-term maintenance guarantees would require an explicit treatment of such delayed effects. The results above therefore provide conditional reference calculations and clarify the behavior of the estimator; empirical performance and long-term memory quality must be assessed separately.

F PROMPT TEMPLATES

This appendix lists representative solver, augmentation, and adapted baseline-module templates used by MEMCON and its evaluation harness.

F.1 BENCHMARK SOLVER SYSTEM PROMPTS

All memory methods (including MEMCON) share the identical solver system prompt per benchmark, so that differences in reported performance isolate the memory component.

ALFWorld system prompt

```
You are now in a household environment called Alfworld, and your
tasks include locating objects, heating or cooling items, and other
similar activities.
NOTE:
- You must strictly follow the syntactic structure of the steps
(where 'a' and 'b' are variables):
    1.  take a from b.
    2.  go to a.
    3.  open a.
    4.  put a in/on b.  (always write "in/on" together, never "in"
or "on" alone.)
    5.  clean a with b.
    6.  heat a with b.
    7.  cool a with b.
    8.  use a.
    9.  think:  xxx
- You must check carefully whether your output command is consistent
with the allowed commands above.  Any output not among the listed
commands is rejected.
```

ScienceWorld system prompt

```
You are a helpful assistant to do scientific experiments in a
text-based environment.  In the environment, there are several
rooms:  kitchen, foundry, workshop, bathroom, outside, living room,
bedroom, greenhouse, art studio, hallway.  You should explore the
environment and find the items you need to complete the experiment.
You can teleport to any room in one step.
For each turn, choose "Thought" or "Action".  If "Thought", output
"Thought:  ...  \n Action:  ...".  If "Action", output only "Action:
...".  Only one Action per response.
Available actions:  open / close / activate / deactivate / connect /
disconnect / use / look around / examine / look at / read / move /
pick up / pour / mix / teleport to / focus on / wait / wait1.
CRITICAL RULES: use EXACT object names as shown in observations;
"focus on OBJ" is typically required at the end; read the task
description literally.
```

PDDL Blocksworld system prompt

```
You are a robot with four actions: pickup, putdown, stack, and
unstack. Blocks can be stacked on top of each other; the arm holds
at most one block; the table holds the rest.
Actions:
- think: xxx (format: 'think: ...')
- pickup <block>: pick up a clear block from the table if the arm
is empty.
- putdown <block>: put the held block on the table.
- stack <block> <block>: stack held top-block onto a clear
bottom-block.
- unstack <block> <block>: pick the top-block off a bottom-block
when the arm is empty.
You must strictly follow these actions; no other actions are
allowed.
```

Analogous solver system prompts are used for *barman*, *gripper*, and *tyreworld* (see `tasks/prompts/pddl_prompt.py` in the released code).

TriviaQA / WebWalkerQA / GAIA. The QA harness requests concise final answers ("Answer: X" for multiple choice, a single string for open-answer tasks). These output conventions do not by themselves specify the complete system prompts, tool prompts, or task preprocessing.

F.2 MemCon-Specific Prompt Fragments

The following augmentation fragments are assembled from stored trajectories and concatenated to retrieved context without a separate LLM call to generate them.

Injected by PlanInject / Retrieve

```
[Proven plan for '{goal_type}' tasks]
   1. {step_1_generalized}
   2. {step_2_generalized}
   ...
   k. {step_k_generalized}
   Adapt object/location names to your current task.
```

Here each step is produced by generalizing a successful trajectory: numbered location and object instances are replaced with categorical placeholders (e.g., `"go to shelf 3"` → `"go to [shelf]"`, `"take mug 1 from diningtable 2"` → `"take [mug] from [diningtable]"`). The regular-expression rewriter covers the standard ALFWorld/ScienceWorld vocabulary (shelves, cabinets, drawers, containers, common foods, tools, etc.).

Injected for ALFWorld *puttwo* and analogous composite goals

```
[TWO-OBJECT TASK: Complete all steps for object 1 first, then repeat
for object 2]
   1. Find & take first object → put it at target.
   2. Find & take second object → put it at target.
[Reference: single-put plan]
   1. {step_1}
   2. {step_2}
   ...
```

The second block reuses the generalized plan for the simpler single-object variant (*put*) when one has been learned, providing an additional hint to the LLM agent.

F.3 Baseline Memory-Module Prompts

We summarize the memory-write/re-rank prompts used by the baselines we re-implemented on top of the shared two-method memory interface. Full text of these prompts is in the released code. These templates document adapted modules; they do not establish reproduction of every learned component or prompt in the cited systems.

Trajectory-summarization system prompt

```
You are a helpful assistant that writes a description of the task
resolution trajectory.
1) Try to summarize the trajectory in no more than 6 sentences.
2) Your response should be a single line of text.
```

Relevance-scoring system/user prompts

```
System: You are an agent designed to score the relevance between
two pieces of text.
User: You will be given a successful case and an ongoing task. Do
not summarize either case; evaluate how relevant and helpful the
successful case is for the ongoing task, on a scale of 1--10.
Success Case: {trajectory}
Ongoing task: {query}
Score:
```

Phase-based summarization system prompt

```
You are an agent skilled in summarization. Your task is to
generate phase-based summaries from given execution records of an
agent's task. These summaries help the agent efficiently utilize
existing information, avoid redundant computations, and ensure task
continuity.
1. Phase-based summarization: organize records into logical
phases and extract key steps.
2. Task relevance: explain what has been completed and what
remains.
3. Clarity and conciseness: precise language, no unnecessary
details.
If intermediate states are incorrect or irrelevant, filter or
correct them to make the summary more accurate.
```

Rule-comparison prompt for the adapted OAgent module

```
You are an advanced reasoning agent that derives general rules from
examples. You will receive one successful trial and one failed
trial.
Goal: compare the positive and negative examples to extract
insights. The insights must be concise and expressed as high-level
reasoning principles.
Output format: each line is one of
   AGREE <EXISTING RULE NUMBER>: <EXISTING RULE>
   REMOVE <EXISTING RULE NUMBER>: <EXISTING RULE>
   EDIT <EXISTING RULE NUMBER>: <NEW MODIFIED RULE>
   ADD: <NEW RULE>
```