EVOLVING BELIEF MEMORY FOR CONTINUAL LEARNING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning agents operating in environments with shared latent structure may need to carry forward reusable propositions whose evidential support remains incomplete at task boundaries. Most existing non-parametric approaches preserve raw interaction histories or compress experience into persistent solutions, strategies, or memories, leaving the evolving belief status of environment-level propositions largely implicit. We introduce Evolving Belief Memory (EBM), a non-parametric continual-learning framework that couples task-directed proposition proposal and evidence-based validation with a persistent belief state. EBM represents each belief as an environment-level proposition together with its current status and associated interaction evidence. We evaluate EBM on adapted continual Database and Poker environments across three backbone models. EBM achieves the highest mean Database reward among the compared non-oracle methods across all evaluated backbones, while remaining competitive on Poker, where performance exhibits substantially greater run-to-run variability. Persistent-state replay shows that the learned belief state benefits a fresh memoryless solver even without raw cross-task interaction history. Analyses on a controlled Database stream further show that reusable beliefs accumulate across tasks and support later task solving, while ablations support the utility of persistent belief storage and evidence-based validation. These results suggest that explicitly maintaining revisable, evidence-grounded beliefs offers a useful way to represent and carry forward learning state in continual language agents.

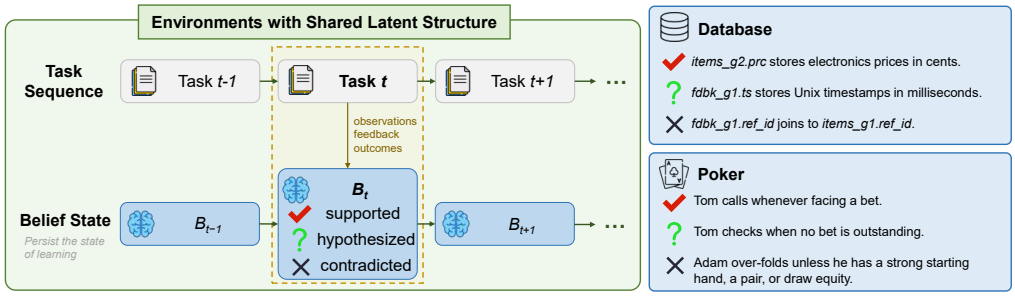

Figure 1: **Evolving Belief Memory for continual learning agents.** Left: Across tasks sharing latent environmental structure, the agent maintains and continually updates a belief memory. Right: Examples of the agent's belief state about the environment at task t.

1 INTRODUCTION

Continual learning requires language agents to learn from experience accumulated over a stream of interactions (Kirkpatrick et al., 2017; De Lange et al., 2022; Wang et al., 2023; Mao et al., 2025). Existing benchmarks instantiate this paradigm in diverse ways, ranging from sequences of tasks drawn from the same or different domains (Wei et al., 2025; Harrington et al., 2026) to ordered task sequences that require the learning and transfer of cumulative skills (Zheng et al., 2025). More recently, a line of work has considered distinct tasks situated within a common environment

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.

with shared latent structure, where experience from one task can reveal properties relevant to others (Asawa et al., 2026). Such properties may require evidence from several interactions to establish. This raises a question for continual learning: *how should an agent carry forward propositions about the environment whose status may change as new evidence arrives across tasks?*

Consider an illustrative database example: an agent suspects that two tables use the same identifier convention. Matching values observed during one task provide partial support, while a later task may expose incompatible formats or motivate a targeted verification query. Keeping the proposition together with its evidence and current status gives subsequent interactions a specific claim to reassess as further evidence becomes available.

Thus, an agent may need to preserve not only past experience, but also an explicit belief state about the environment. Most existing non-parametric approaches do not explicitly maintain such an evolving belief state. Full-history in-context learning retains past experience, but leaves the learning state implicit, requiring reusable propositions and their current status to be reconstructed repeatedly from an ever-growing history. Memory-based methods instead compress experience into persistent solutions, strategies, insights, skills, or playbooks (Chhikara et al., 2025; Zhang et al., 2026a; Suzgun et al., 2026), thereby primarily preserving consolidated products of learning rather than an explicit state that tracks propositions together with their evidence. Concurrent work such as BeliefMem represents uncertainty through competing interpretations whose confidence scores change as new evidence arrives (Liao et al., 2026). We study how agents retain and revisit unresolved environment propositions, updating their status as additional evidence accumulates across tasks.

We introduce **Evolving Belief Memory (EBM)**, a non-parametric continual-learning framework that couples task-directed proposition proposal and evidence-based validation with a persistent, evidence-tracked *belief state*. The belief state contains environment-level propositions, their current status, and associated interaction evidence.

EBM realizes this process through three components. Before solving each task, a *proposer* uses the current request, interaction history, and current belief state to propose additions or revisions. Each proposal can reference existing evidence or request an environment interaction for verification. A *validator* assesses the resulting evidence and controls how the proposal changes the belief state. A *solver* then uses the current belief state and interaction history to solve the task, producing further experience for later updates.

We evaluate EBM on adapted continual Database and Poker environments from Asawa et al. (2026), which involve relatively deterministic schema and semantic properties in Database and uncertain behavioral regularities in Poker. Across three backbone models—Qwen-3.8-27B, Claude-Sonnet-4.6, and Claude-Opus-4.8—EBM achieves the highest mean Database reward among the compared non-oracle methods and competitive results on Poker, where run-to-run variability is substantially larger. Persistent-state replay shows that its retained belief state can help a memoryless solver without raw cross-task interaction history. On a controlled Database stream that exposes most reusable properties early, frozen belief checkpoints support later tasks, while new propositions and verification probes become less frequent in the reuse-focused phase. Component ablations further support the utility of persistent belief state and validation.

Our contributions are threefold:

- We study the acquisition, revision, and reuse of environment propositions across streams of distinct tasks that share latent environment structure.
- We propose **Evolving Belief Memory (EBM)**, which combines task-directed proposition proposal and evidence-based validation with a persistent belief state consisting of environment propositions, their current status, and associated evidence.
- We evaluate EBM on continual Database and Poker environments, measure the utility of its retained belief state through replay, and analyze proposition acquisition and reuse within a controlled Database stream.

2 Related Work

2.1 Continual Learning Tasks and Benchmarks

Recent benchmarks increasingly evaluate language agents as continual learners over sequences of tasks rather than as static solvers of independent instances. However, these benchmarks differ in what structure makes experience reusable across the task stream. Evo-Memory constructs streams by sequentializing instances from existing benchmarks (Wei et al., 2025), and related work similarly recasts existing tasks as sequential adaptation problems (Joshi et al., 2025; Harrington et al., 2026). AgentCL argues that such sequentialization alone does not necessarily induce meaningful continual learning, and instead constructs controlled compositional streams in which earlier sub-solutions, evidence, or workflows are deliberately reusable in later tasks (Shu et al., 2026). LifelongAgentBench uses skill-grounded, interdependent task sequences to evaluate skill acquisition, transfer, and retention (Zheng et al., 2025), while related benchmarks explicitly organize task families around reusable skills, procedures, or workflows (Zhang et al., 2026b; Guan et al., 2026; Zhong et al., 2026).

Continual Learning Bench (CL-Bench) studies a complementary source of cross-task structure: task instances are embedded in a persistent but initially unknown environment whose latent properties can be progressively inferred through interaction (Asawa et al., 2026). Depending on the domain, this shared structure may correspond to database conventions, codebase organization, system dynamics, or opponent behavior and strategy. We focus on this setting, where *distinct tasks expose shared latent properties of the underlying environment*, and continual learning primarily concerns acquiring, revising, and reusing propositions about those properties across task instances.

2.2 Continual Learning from Agent Experience

A central challenge in continual learning is to turn sequential experience into information that improves future behavior (Kirkpatrick et al., 2017; De Lange et al., 2022). While some approaches adapt model parameters across tasks (Wu et al., 2026), recent work increasingly studies non-parametric adaptation with frozen foundation models and persistent external state. In-context learning retains past trajectories directly, but leaves the learning state implicit, requiring reusable structure and its current status to be reconstructed from an increasingly long history. Other approaches distill experience into reflections and insights (Shinn et al., 2023; Zhao et al., 2024), workflows and skills (Wang et al., 2024; Shu et al., 2026), persistent memories (Chhikara et al., 2025), and evolving strategies or playbooks (Suzgun et al., 2026; Zhang et al., 2026a). Harness Continual Learning further broadens the persistent learning state to agent components such as memory, skills, tools, and routing policies (Kang et al., 2026). These approaches offer different representations and update mechanisms for making experience available to future decisions.

BeliefMem retains competing conclusions with evidence references and updates their confidence through evidence merging (Liao et al., 2026). It selects candidates with observational support while preserving uncertainty about them. EBM separates task-directed proposition generation from the evidence verdict governing memory updates. Each eligible proposal is assessed against non-empty evidence; the verdict and proposed operation jointly determine the stored statement and status. In particular, an eligible ADD receiving an INCONCLUSIVE verdict can create a HYPOTHESIZED belief. Subsequent revalidation, replacement, and invalidation follow operation-specific status transitions (Appendix C).

The reported evaluation protocols also differ. BeliefMem's ALFWorld setup constructs memory from training expert trajectories before evaluation and retrieves it during test episodes (Liao et al., 2026, Appendix B.1). EBM acquires and revises memory online across the evaluation task stream. We do not report a direct comparison with BeliefMem under this same online protocol.

Several recent methods on ARC-AGI-3 (ARC Prize Foundation, 2026; Brown, 2026; Courtis et al., 2026; Lehmann et al., 2026) iteratively infer environment structure through interaction. These approaches similarly use iterative hypothesis formation and evaluation to construct and refine a world model through interaction within a game, whereas EBM uses this process to update a persistent belief state across distinct task objectives.

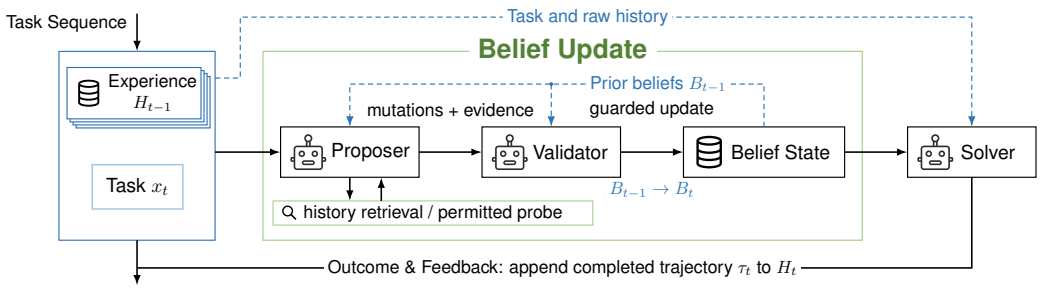

Figure 2: **EBM's belief-update and task-execution phases.** The proposer supplies mutation–evidence pairs; the validator controls their incorporation into memory. Updated beliefs guide the solver, whose completed trajectory is retained for subsequent tasks.

3 METHODOLOGY

Evolving Belief Memory (EBM) maintains an explicit belief state across distinct tasks $x_1, \ldots, x_T$ that expose shared latent properties of an environment $\mathcal{E}$. Each task produces a trajectory $\tau_t = (x_t, h_t, r_t)$, where $h_t = \big((a_{t,k}, o_{t,k})\big)_{k=1}^{L_t}$ is the ordered sequence of solver actions and observations, and r_t is the task outcome or feedback.

A *proposer* and *validator* update beliefs before each task, and a *solver* uses them during execution (Figure 2; Algorithm 1). EBM persists both experience history and belief state.

Experience History. The experience history $H_t = (\tau_1, \ldots, \tau_t)$ retains completed raw trajectories as evidence for current-task reasoning and belief updates.

Belief State. The belief state $B_t = \{b_1, \ldots, b_{N_t}\}$ stores atomic propositions about reusable environment properties, each with its current status and accumulated interaction evidence. A HYPOTHESIZED belief has insufficient evidence for either support or refutation; a SUPPORTED belief has usable supporting evidence within its scope; and a CONTRADICTED belief has been directly refuted. Statuses reflect the agent's assessment rather than objective truth and may change with new evidence. Database drift additionally uses STALE (Appendix A).

Proposer. Given the current request x_t, experience H_{t-1}, and prior beliefs B_{t-1}, the proposer outputs belief mutations paired with evidence:

$$\Delta B_t = P(x_t, H_{t-1}, B_{t-1}) = \{(m_{t,i}, e_{t,i})\}_{i=1}^{K_t}. \tag{1}$$

Each mutation $m_{t,i}$ introduces a new proposition or proposes a revision; $e_{t,i}$ is its associated evidence.

Here P includes evidence acquisition from eligible history records or permitted verification probes (Appendix B). The set ΔB_t may be empty when no update worth pursuing is identified.

Validator. The validator assesses each mutation–evidence pair against the prior beliefs:

$$d_{t,i} = V(m_{t,i}, e_{t,i}, B_{t-1}), \tag{2}$$

where $d_{t,i}$ is a SUPPORT, REFUTE, or INCONCLUSIVE verdict.

The validator does not generate proposition content. Its verdict governs the update

$$B_t = \mathcal{U}\left(B_{t-1}, \{(m_{t,i}, d_{t,i}, e_{t,i})\}_{i=1}^{K_t}\right). \tag{3}$$

Here $\mathcal{U}$ applies guarded additions or revisions and records their evidence. An inconclusive verdict creates a HYPOTHESIZED belief for an eligible addition but preserves the current status for revalidation. Replacement and invalidation rules are specified in Appendix C.

Solver. After this pre-task update, the solver uses the request, prior experience, and B_t to interact until termination. Its completed trajectory is appended to the history for subsequent tasks.

Algorithm 1 EBM: Evolving Belief Memory

Require: Task-request stream $\mathcal{X} = (x_1, \ldots, x_T)$; proposer P, validator V, solver S

Initialize belief memory $B_0 \leftarrow \emptyset$
Initialize experience history $H_0 \leftarrow ()$
for $t = 1, \ldots, T$ **do**
    Receive task request x_t
    // Belief update: once per task
    $\Delta B_t \leftarrow P(x_t, H_{t-1}, B_{t-1})$
    Validate ΔB_t and update B_t according to Eqs. (2)–(3)
    // Task execution: potentially multiple interactions
    Initialize task trace $h_t \leftarrow ()$ and step index $k \leftarrow 1$
    while x_t is not terminated **do**
        $a_{t,k} \leftarrow S(x_t, H_{t-1}, h_t, B_t)$
        Execute $a_{t,k}$ and observe $o_{t,k}$
        $h_t \leftarrow \textsc{Append}(h_t, (a_{t,k}, o_{t,k}))$; $k \leftarrow k + 1$
    end while
    Observe task outcome r_t
    $H_t \leftarrow \textsc{Append}(H_{t-1}, (x_t, h_t, r_t))$
end for
return B_T

4 Experiments

We evaluate whether persisting an evolving belief state helps language agents accumulate reusable knowledge across tasks in environments with shared latent structure. Our experiments address four questions:

- **RQ1 — Does persistent belief state improve continual task performance?**
- **RQ2 — Is the accumulated belief state useful without raw interaction history?**
- **RQ3 — Does the belief state accumulate learning progress across tasks?**
- **RQ4 — What roles do persistent beliefs and belief validation play in EBM?**

4.1 Experimental Setup

Benchmarks and baselines. We evaluate on adapted versions of two continual-learning environments from CL-Bench: Database and Poker (Asawa et al., 2026). The former requires learning database schema and semantics, while the latter requires learning behavioral patterns. We compare against ICL, ICL + ReAct (Yao et al., 2022), Dynamic Cheatsheet (DC) (Suzgun et al., 2026), ACE (Zhang et al., 2026a), Mem0 (Chhikara et al., 2025), and Tycho (adapted) (Lehmann et al., 2026), together with oracle variants where applicable. In our Tycho runs, the models maintained only a textual insights notepad and did not instantiate executable world models; this comparison therefore evaluates that adapted behavior.

Evaluation. We evaluate across multiple backbone models: Qwen3.8-27B (Qwen Team, 2026a;b), Claude-Sonnet-4.6 (Anthropic, 2026b), and Claude-Opus-4.8 (Anthropic, 2026a). Unless otherwise stated, we report the mean and standard deviation over five independent runs. Similar to CL-Bench (Asawa et al., 2026), our primary outcome is task reward. For Database, the per-instance reward is $1-q/B$ for a correct answer and 0 otherwise, where q is the number of exploratory queries and B is the query budget; for Poker, reward is the profit or loss from each hand, measured in big blinds. We additionally report the continual-learning gain, computed per instance as the difference between the stateful system's reward and that of its stateless counterpart on the same instance, and aggregated over the task stream. We use reward for cross-method comparisons; gain is relative to each system's stateless counterpart. For Database, we also report the average number of environment queries. Across all LLM calls, we use reasoning effort `medium` and `max_tokens`=40,000. ICL retains up to 40 previous instances on Database and 30 on Poker; EBM retains a raw-

Table 1: Main results on the Database and Poker environments. Cumulative rewards and gains are averaged over five runs and reported as mean ± standard deviation. For Database, we additionally report the average number of queries per task. Higher reward and gain are better, while fewer queries are better. Best mean results within each backbone are bold and second-best results are underlined; these markings do not imply statistical significance. Tycho (adapted) used a textual notepad rather than an executable world model. Database EBM includes the STALE extension.

		Database			Poker	
Model	**Method**	**Reward ↑**	**Gain ↑**	**Queries ↓**	**Reward (BB) ↑**	**Gain (BB) ↑**
Qwen3.8	ICL	16.79 ± 1.71	7.79 ± 1.71	2.420 ± 0.250	78.40 ± 98.41	-5.00 ± 98.41
	ICL (w ReAct)	20.83 ± 6.43	10.16 ± 6.43	2.295 ± 0.104	60.20 ± 48.96	-156.00 ± 48.96
	ACE	13.69 ± 2.27	1.63 ± 2.27	6.280 ± 0.465	76.34 ± 65.35	23.34 ± 65.35
	DC	21.76 ± 1.34	10.83 ± 1.34	2.880 ± 0.349	12.40 ± 25.63	-103.00 ± 25.63
	Mem0	17.17 ± 2.24	6.24 ± 2.24	4.390 ± 0.374	55.98 ± 22.32	-49.02 ± 22.32
	Tycho (adapted)	19.53 ± 0.52	9.20 ± 0.52	5.955 ± 0.261	$\mathbf{290.90 \pm 349.85}$	$\mathbf{88.20 \pm 349.85}$
	EBM (Ours)	$\mathbf{25.16 \pm 3.08}$	$\mathbf{12.16 \pm 3.08}$	$\mathbf{2.090 \pm 0.155}$	141.06 ± 101.08	41.06 ± 101.08
Sonnet4.6	ICL	17.89 ± 1.19	7.36 ± 1.19	2.395 ± 0.144	172.82 ± 26.40	-32.88 ± 26.40
	ICL (w ReAct)	19.13 ± 2.63	8.40 ± 2.63	2.275 ± 0.121	193.28 ± 29.81	-15.62 ± 29.81
	ACE	19.69 ± 2.17	5.63 ± 2.17	2.660 ± 0.302	206.56 ± 45.81	23.06 ± 45.81
	DC	21.11 ± 3.13	11.11 ± 3.13	2.780 ± 0.176	110.38 ± 32.63	-87.42 ± 32.63
	Mem0	12.16 ± 0.84	0.23 ± 0.84	7.335 ± 0.160	203.32 ± 38.08	-35.68 ± 38.08
	Tycho (adapted)	16.48 ± 2.21	5.35 ± 2.21	4.695 ± 0.608	223.32 ± 23.09	$\mathbf{52.82 \pm 23.09}$
	EBM (Ours)	$\mathbf{23.41 \pm 3.33}$	$\mathbf{13.48 \pm 3.33}$	$\mathbf{2.135 \pm 0.207}$	$\mathbf{238.16 \pm 76.26}$	0.46 ± 76.26
Opus4.8	ICL	22.55 ± 4.01	9.75 ± 4.01	2.310 ± 0.129	89.90 ± 14.75	-3.80 ± 14.75
	ICL (w ReAct)	27.13 ± 1.51	11.27 ± 1.51	$\mathbf{2.115 \pm 0.128}$	85.84 ± 17.70	16.24 ± 17.70
	DC	23.12 ± 2.78	8.99 ± 2.78	2.495 ± 0.108	56.52 ± 36.59	-37.88 ± 36.59
	Tycho (adapted)	25.45 ± 1.50	11.45 ± 1.50	2.380 ± 0.101	143.52 ± 38.99	45.42 ± 38.99
	EBM (Ours)	$\mathbf{28.61 \pm 1.52}$	$\mathbf{13.61 \pm 1.52}$	2.265 ± 0.078	$\mathbf{159.12 \pm 49.51}$	$\mathbf{65.12 \pm 49.51}$

experience window of 20 instances in addition to its persistent belief state. Benchmark adaptations, baselines, metric definitions, and shared prompt templates are provided in Appendix A.

4.2 RESULTS

RQ1: Does persistent belief state improve continual task performance? Table 1 shows that, on Database, EBM achieves the highest mean reward and gain across all three backbone models. It also has the lowest reported query count with Qwen3.8 and Sonnet4.6 and the second-lowest with Opus4.8. These results show that EBM combines consistently strong task performance with relatively few environment queries. Poker presents a more variable setting. EBM achieves the highest mean reward with Sonnet4.6 and Opus4.8, whereas Tycho (adapted) achieves the highest mean with Qwen3.8 but also exhibits the largest standard deviation. The large run-to-run variation makes differences in sample means less conclusive. Moreover, gain is computed relative to a memoryless baseline that was run only once. The reported gain variance therefore captures variation in the evaluated method but not uncertainty in the baseline itself. In a high-variance environment such as Poker, absolute gain values may consequently be sensitive to the particular baseline run and should be interpreted descriptively rather than as precise estimates of continual-learning improvement.

Overall, EBM shows its most consistent reward advantage on Database and remains competitive on Poker. Query count should be interpreted as environment-interaction efficiency rather than total inference efficiency: EBM uses more tokens than several baselines, as reported in Appendix A.4. A compute-matched comparison would therefore be needed to more fully separate the effect of the memory representation from additional inference-time computation.

RQ2: Is the accumulated belief state useful without raw interaction history? We replay each saved state through a fresh memoryless solver as described in Figure 3. EBM yields mean reward improvements of $+0.318$ in Database and $+0.670$ in Poker, showing that its retained state can benefit a solver without the underlying cross-instance interaction history.

Tycho (adapted) and DC obtain Database gains close to EBM. In Poker, DC and ACE reduce mean replay reward, while Tycho (adapted) and EBM have positive means with substantial variability.

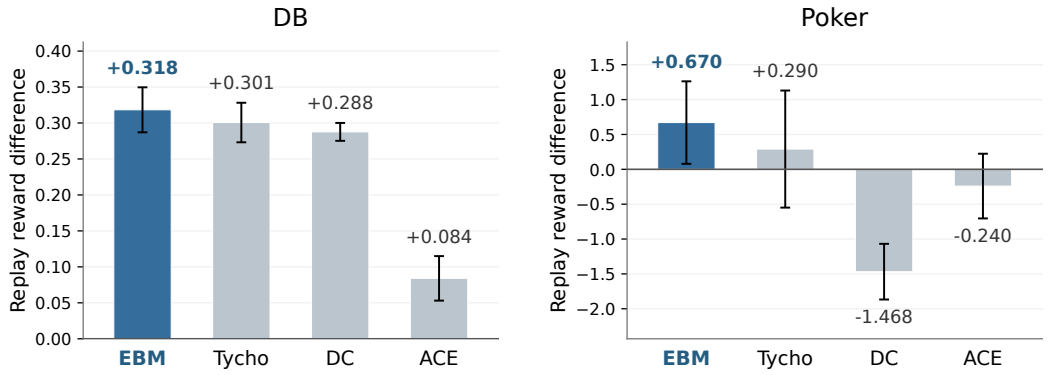

Figure 3: **Persistent-state replay with a memoryless solver on Qwen3.8-27B.** At each instance i in the task sequence, we take the persistent-state checkpoint learned by each method from preceding instances and inject it into the context of a fresh memoryless solver. The solver receives no raw cross-instance interaction history. Bars show the mean reward difference relative to the same memoryless solver run on the same instances without the checkpoint, averaged over five runs; error bars denote the standard error of the mean (SE).

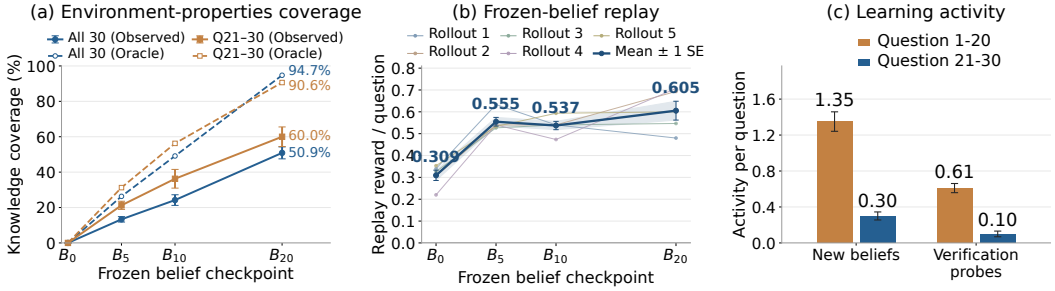

Figure 4: **Belief-state accumulation and reuse across a controlled Database stream on Qwen3.8-27B.** (a) Observed coverage of annotated environment properties across frozen-belief checkpoints, shown together with oracle coverage for all 30 questions and the reuse-focused Q21–30; the oracle provides a ground-truth reference for the progression of coverage. (b) Replay reward on Q21–30 from frozen belief states, using a fresh memoryless solver with no raw cross-task interaction history and no further belief updates, to measure forward utility relative to the blank checkpoint B_0. (c) New-belief and verification-probe activity per question during normal online execution, including raw interaction history, for Q1–20 and Q21–30. Results are averaged over five independent runs; B_i denotes the checkpoint at instance i, error bars show ± 1 SE, and thin lines in (b) show individual runs.

These comparisons assess each method's retained state, without isolating the contribution of belief-status labels.

RQ3: Does the belief state accumulate learning progress across tasks? We construct a controlled Database stream in which the first 20 instances front-load most properties needed by the final 10 reuse-focused instances: 54 of 57 annotated properties are exposed by Q20. Across frozen-belief checkpoints, EBM's observed coverage increases but remains below the ground-truth oracle, indicating partial acquisition of the annotated properties (Figure 4a).

To test whether this accumulated knowledge has forward utility, we replay Q21–30 from frozen checkpoints with a fresh memoryless solver, removing raw cross-task interaction history and further belief updates. Mean reward at each later checkpoint exceeds that of the blank state B_0, though it does not improve monotonically (Figure 4b). During normal online execution with raw history, new-belief and verification activity decrease from Q1–20 to Q21–30, consistent with reuse under the controlled ordering (Figure 4c).

RQ4: What roles do persistent beliefs and belief validation play in EBM? We compare component configurations in Figure 5a. Removing either the belief state or validator lowers mean cumulative reward, with a larger drop when validation is removed. Performance without the validator is close to that of the proposer alone. The full configuration performs best among these variants, consistent with benefits from combining persistence and validation.

Figure 7 (Appendix A.4) shows a positive marginal contribution from beliefs when raw history is available, with a larger contribution as history is reduced. This pattern is consistent with partially overlapping information in beliefs and raw history; it does not distinguish belief-state tracking from other forms of history compression.

In the GPT-based factual correctness evaluation (Figure 5b), supported beliefs are predominantly correct, hypothesized beliefs are more mixed, and all contradicted beliefs are false. For each belief, a GPT-6-Astra (OpenAI, 2026) judge (Appendix A.6) assesses whether its content is factually correct with respect to the task-provided ground-truth properties. This evaluation does not require a one-to-one correspondence between beliefs and ground-truth properties: a single ground-truth property may be represented across multiple beliefs, while each belief is judged individually based on the correctness of its stated content. Validator status remains a model-dependent assessment of the available evidence and should not be interpreted as a calibrated probability of factual correctness. Validator status remains a model-dependent assessment of the available evidence, not a guarantee of factual correctness or a calibrated probability.

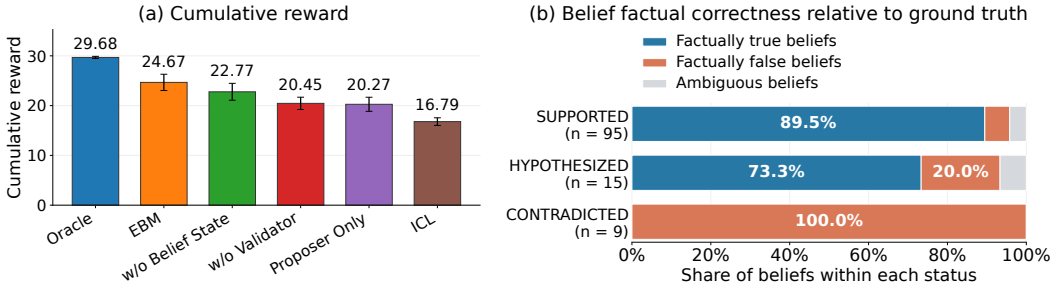

Figure 5: **Component ablation and factual correctness of beliefs in EBM on Qwen3.8-27B. (a)** Cumulative reward under different EBM component ablations, together with the oracle and ICL baselines. All configurations, including the full EBM configuration, were rerun specifically for this ablation study. Results are averaged over five independent runs, with error bars denoting ± 1 SE. **(b)** Belief factual correctness relative to task-provided ground truth, as judged by GPT-6-Astra and grouped by validator status. Beliefs were sampled from five independent runs. Bars show the fraction of beliefs in each factual-correctness category; sample counts are shown on the left.

4.3 Case Study

Table 2 illustrates model-assigned belief revisions based on joint counts of two verification fields. The count establishes set inclusion; it does not establish which field matches the task's intended meaning of "verified." The case demonstrates revision behavior, while the preferred semantic interpretation remains a model inference. Additional case study can be found in Appendix A.5.

5 Conclusion

We introduced **Evolving Belief Memory (EBM)** for non-parametric adaptation across tasks in environments with shared latent structure. EBM couples task-directed proposition proposal and verification with a persistent belief state containing environment-level propositions, current status, and associated evidence. This design allows unresolved propositions to be revisited as later tasks provide new evidence.

EBM achieves the highest mean Database reward among the compared non-oracle methods across the evaluated backbone models and competitive results on Poker, where variability is substantially larger. Replay experiments show that the learned state can help a solver without raw cross-task interaction history. Within the controlled Database stream, frozen belief states support later tasks

Table 2: **Belief revision in EBM.** Two examples of persistent beliefs revised through changes in status and/or proposition as new evidence is incorporated. Belief 1 undergoes only a status change, with the contradicted proposition discarded, whereas Belief 2 undergoes both a status change and a proposition update. The observed counts establish the set relation; the final semantic interpretation remains a model inference.

	Earlier belief	Status change	Updated belief
Belief 1	`vrf=1` is interpreted as "verified purchase"; `verified_status` is treated as a separate dimension and ignored.	SUPPORTED → CONTRADICTED	–
Belief 2	`vrf=1` and `verified_status='verified'` may denote different sets; based on an earlier failed query, the latter is hypothesized to be the correct filter for "verified" reviews or purchases.	HYPOTHESIZED → SUPPORTED	`verified_status='verified'` is a strict subset of `vrf=1`, and recent query results are consistent with the former, rather than `vrf=1`, as the intended filter for "verified" items.

Observed evidence. All 246,267 rows with `verified_status='verified'` have `vrf=1`, while 105,478 additional rows have `vrf=1` and `verified_status IS NULL`. Thus, `vrf=1` is a strict superset of `verified_status='verified'`.

and acquisition activity decreases during the reuse-focused phase. Component ablations further support the utility of persistent belief state and validation.

6 LIMITATIONS

Our evaluation focuses on Database and Poker with three backbone models. Extending EBM to broader task families and longer interaction streams would help characterize its generality.

Belief validation relies on LLM judgments and available evidence, so the reliability of updates depends on the quality of observations and the underlying model's reasoning.

Maintaining an explicit belief state alongside raw history and running proposal and validation calls can also increase context and inference costs. More selective retrieval, evidence consolidation, and adaptive update frequency are promising directions for improving efficiency as task streams and belief collections grow.

AI USE STATEMENT

In this work, we used generative AI tools to assist with literature search, critical reading, drafting and revising the manuscript. This assistance included checking consistency between claims, tables, and prompts, and clarifying comparisons with related work. The authors are responsible for verifying the resulting text and literature descriptions against the cited sources.

In addition, for the controlled Database stream experiment, we used GPT-6-Astra to reorder 30 tasks based on the environment properties associated with each task. These environment properties were manually identified and specified by the authors; the model was used only to produce the resulting task ordering from this author-provided information. We also used GPT-6-Astra as an evaluation judge for the factual-correctness analysis of persistent beliefs described in Section 4.2 and Appendix A.6.

AI-assisted manuscript revision did not generate or replace the reported experimental measurements. The authors take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 describes the algorithm, and Appendix C summarizes its shared status-transition rules. Appendix A documents the benchmark adaptations, reported settings, metrics, and shared prompt

templates. We additionally provide details of the task-specific evidence acquisition mechanisms in Appendix B.

REFERENCES

Anthropic. Claude Opus 4.8 System Card, May 2026a. URL https://www.anthropic.com/system-cards.

Anthropic. Claude Sonnet 4.6 System Card, February 2026b. URL https://www.anthropic.com/system-cards.

ARC Prize Foundation. Arc-agi-3: A new challenge for frontier agentic intelligence. *arXiv preprint arXiv:2603.24621*, 2026.

Parth Asawa, Christopher M Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E Gonzalez. Continual learning bench: Evaluating frontier ai systems in real-world stateful environments. *arXiv preprint arXiv:2606.05661*, 2026.

Ryan Brown. Retrodict. https://github.com/ryanbbrown/Retrodict, 2026. GitHub repository.

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.

David Courtis, Wenhao Li, and Scott Sanner. Opine-world: Programmatic world modeling with ontology-error-prioritized interactive exploration for arc-agi-3, 2026. URL https://arxiv.org/abs/2607.01531.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2022.

Tianyi Guan, Yiding Wang, Haotong Yang, Siyuan Cao, Shirui Liu, Yi Hu, Jiaqi Li, and Muhan Zhang. Continualskillbench: Can llm agents truly evolve their capabilities? *arXiv preprint arXiv:2608.03874*, 2026.

Anne Harrington, Nayan Saxena, Michael Murphy, Anastasia Borovykh, Zeyu Yun, Sridhar Kamath, Ara Eindra Kyi, Trevor Darrell, Jitendra Malik, and Yutong Bai. When does continual learning require learning. *arXiv preprint arXiv:2607.07847*, 2026.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. Swe-bench-cl: Continual learning for coding agents. *arXiv preprint arXiv:2507.00014*, 2025.

Borui Kang, Jinrui Gu, Junhan Lv, Wenbin Li, Lei Wang, and Yang Gao. Harness continual learning: Continual adaptation beyond model parameters. *arXiv preprint arXiv:2608.19013*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Jens Lehmann, Andrei Aioanei, and Sahar Vahdati. Tycho: Active abstraction with programmatic world models for arc-agi-3, 2026. URL https://arxiv.org/abs/2607.28287.

Junfeng Liao, Qizhou Wang, Jianing Zhu, Bo Du, Rui Yan, and Xiuying Chen. Belief memory: Agent memory under partial observability. *arXiv preprint arXiv:2605.05583*, 2026.

Lingjun Mao, Jiawei Ren, Kun Zhou, Jixuan Chen, Ziqiao Ma, and Lianhui Qin. Deliverybench: Can agents earn profit in real world? *arXiv preprint arXiv:2512.19234*, 2025.

OpenAI. Gpt-6 astra: A new generation of intelligence. https://openai.com/index/gpt-6-astra/, September 2026. Accessed: 2026-09-26.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026a. URL https://qwen.ai/blog?id=qwen3.8.

Qwen Team. Qwen3.8-27b. https://huggingface.co/Qwen/Qwen3.8-27B, 2026b. Hugging Face model repository.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. Agentcl: Toward rigorous evaluation of continual learning in language agents. *arXiv preprint arXiv:2606.02461*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

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. *arXiv preprint arXiv:2305.16291*, 2023.

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

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Zheng Wu, Xingyu Lou, Xinbei Ma, Yansi Li, Weiwen Liu, Weinan Zhang, Jun Wang, and Zhuosheng Zhang. Agent-dice: Disentangling knowledge updates via geometric consensus for agent continual learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 18245–18262, 2026.

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

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026a.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, et al. Skillflow: Benchmarking lifelong skill discovery and evolution for autonomous agents. *arXiv preprint arXiv:2604.17308*, 2026b.

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.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo FR Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

A EXPERIMENTS

A.1 BENCHMARKS

We use two environments from CL-Bench (Asawa et al., 2026), DATABASE and POKER. Our preliminary inspection identified several issues in their original implementations that could confound evaluation. We adapt their interfaces and evaluation as described below.

Database. The DATABASE environment requires agents to answer questions by interacting with a persistent database. Successful adaptation requires recovering reusable properties of the environment, such as table schemas, join relationships, field semantics, units, encodings, and other data conventions. CL-Bench provides a default 40-instance schedule with database drift after instance 20, as well as a 30-instance no-drift schedule. We use the default schedule in most experiments and the no-drift setting only for controlled stream ablations.

The original database contains three product datasets: Office Products, Electronics, and Musical Instruments. However, some tables also include a `main_cat` field with values such as "All Electronics," making it ambiguous in some natural language questions whether category names refer to the dataset or to this field. This ambiguity affects 35 of the 40 questions. In the traces released by the original authors, 30.5% of the observed errors appear directly attributable to this issue; in five Qwen3.5-27B ICL traces, the corresponding fraction is 92%.

To remove this ambiguity, we append the following clarification to each instance: "In these questions, the terms 'office products,' 'electronics,' and 'musical instruments' refer to the corresponding datasets as a whole."

Poker. The POKER environment evaluates adaptation to recurring behavioral patterns in sequential decision making. The agent plays heads-up Texas Hold'em against deterministic opponents, each following a fixed and potentially exploitable policy that can be inferred from betting behavior and showdown information.

Our inspection identified several implementation issues in the original environment. The displayed action history is unreliable: blinds can be misreported as raises, actions are not associated with betting streets, bet amounts are often omitted, and parts of the history are inferred from the current numerical state rather than explicitly recorded.

We fix these issues by providing an explicit and accurate hand history, complete action-relevant state information, and corrected evaluation metrics. These changes affect only the interface and evaluation; the opponents' policies are left unchanged.

For our experiments, each run contains 60 instances, ordered as follows: instances 1–20 use Tom, 21–40 use Alex, and 41–60 use Adam.

A.2 BASELINES

We compare against several continual-learning baselines, which broadly fall into two categories: raw-history methods and distillation-based methods.

ICL maintains the interaction history across instances, including environment observations and agent actions, and conditions the model on this accumulated history for subsequent decisions.

ICL + ReAct (Yao et al., 2022) uses ReAct-style reasoning at each interaction step and persistently stores observations, reasoning traces, and actions within the configured history window.

Dynamic Cheatsheet (DC) (Suzgun et al., 2026) maintains a persistent, self-curated cheatsheet that accumulates reusable strategies and lessons from previous interactions and is updated throughout evaluation.

ACE (Zhang et al., 2026a) maintains an evolving structured playbook through a Generator–Reflector–Curator pipeline, which extracts reusable insights from trajectories and applies localized updates to the stored context.

Mem0 (Chhikara et al., 2025) maintains a long-term memory store by extracting salient information from interactions and consolidating it via ADD, UPDATE, DELETE, and NOOP operations. Relevant memories are then retrieved for subsequent queries.

Tycho (adapted) (Lehmann et al., 2026) is based on a method designed to use a coding agent to construct and refine executable programmatic world models from interaction history. In our experiments, none of the evaluated models instantiated such models. They only maintained the textual *insights notepad* that accumulated observations and strategies. We label these results as adapted Tycho because they evaluate this observed notepad behavior and do not establish the performance of its intended executable-model mechanism.

We additionally evaluate memoryless variants in replay experiments to assess the utility of the supplied persistent state without raw cross-instance history. For DATABASE, we also construct oracle variants using the ground-truth environment knowledge provided by the original dataset.

A.3 EVALUATION METRICS

We adopt the evaluation metrics from Asawa et al. (2026). For a schedule of N instances, let r_t^{sf} and r_t^{sl} denote the reward on instance t under the stateful and stateless settings, respectively. The cumulative stateful reward is $R^{\mathrm{sf}} = \sum_{t=1}^{N} r_t^{\mathrm{sf}}$. We define the per-instance gain as $g_t = r_t^{\mathrm{sf}} - r_t^{\mathrm{sl}}$, and the cumulative gain as $G = \sum_{t=1}^{N} g_t$. Gain is relative to each method's stateless counterpart; reward provides the direct comparison of task performance across methods.

Database. Let q_t denote the number of exploratory SQL queries used on question t, with query budget $Q = 15$. The per-question reward is $r_t^{\mathrm{DB}} = 1 - q_t/Q$ if the final answer is correct, and 0 otherwise. Thus, the reward captures both answer correctness and query efficiency, with $r_{\max} = 1$. The corresponding gain is $g_t^{\mathrm{DB}} = r_{t,\mathrm{sf}}^{\mathrm{DB}} - r_{t,\mathrm{sl}}^{\mathrm{DB}}$.

Poker. Let p_t denote the chip profit on hand t and B the big-blind size. The per-hand reward is the profit measured in big blinds, $r_t^{\mathrm{Poker}} = p_t/B$, and the average reward is $\bar{r}^{\mathrm{Poker}} = \frac{1}{N}\sum_{t=1}^{N} p_t/B$. The corresponding gain is $g_t^{\mathrm{Poker}} = r_{t,\mathrm{sf}}^{\mathrm{Poker}} - r_{t,\mathrm{sl}}^{\mathrm{Poker}}$.

A.4 MORE RESULTS

A.4.1 MAIN RESULTS

We report detailed results for the DATABASE and POKER tasks in Table 3 and Table 4, respectively.

The token totals show differences in inference usage. For example, on Qwen3.8 Poker, EBM uses 13.018M input and 0.628M output tokens, totaling 13.646M, compared with 3.712M input and 0.088M output tokens for ICL + ReAct, totaling 3.800M. This is approximately 3.59 times as many tokens. Lower reported environment-query counts therefore should not be interpreted as lower total computational cost or as evidence that the memory representation alone accounts for the reward differences.

A.4.2 STALENESS MECHANISM FOR EBM

The core loop of EBM is general across tasks, while we introduce a task-specific adaptation for DATABASE. In the default setting, after instance 20 is completed, the system notifies all agents that the database has changed. EBM explicitly incorporates this notification by marking all existing beliefs as STALE, indicating that they may no longer hold under the changed environment and should therefore be used with caution. Importantly, this notification is part of the environment and is visible to all methods; EBM does not receive any additional task-specific input beyond what is available to the other agents.

We conduct an ablation study to evaluate the impact of this staleness mechanism. As shown in Table 3, removing the STALE status lowers mean reward by different amounts across backbones. EBM without the mechanism retains the highest mean reward among the baselines on DATABASE, while its gain ranking varies by backbone. These results show that EBM remains competitive without this task-specific adaptation; they do not establish equivalence between the two variants.

Table 3: Detailed results on the Database environment, with cumulative rewards and gains. Results are averaged over five runs and reported as mean ± standard deviation. Higher is better for reward, gain, and accuracy, while lower is better for queries and token usage. Best mean results within each backbone are bold; this does not indicate statistical significance. Tycho (adapted) used only its textual notepad.

Model	Method	Reward ↑	Gain ↑	Acc. (%) ↑	Avg. Queries ↓	Tokens (M) ↓	
						Input	Output
Qwen3.8	ICL	16.79 ± 1.71	7.79 ± 1.71	50.0 ± 4.7	2.42 ± 0.25	4.165 ± 0.400	0.041 ± 0.008
	ReAct	20.83 ± 6.43	10.16 ± 6.43	61.0 ± 18.8	2.30 ± 0.10	4.509 ± 0.251	$\mathbf{0.035 \pm 0.004}$
	ACE	13.69 ± 2.27	1.63 ± 2.27	59.5 ± 8.4	6.28 ± 0.47	2.193 ± 0.315	0.189 ± 0.030
	DC	21.76 ± 1.34	10.83 ± 1.34	66.5 ± 2.9	2.88 ± 0.35	$\mathbf{1.054 \pm 0.111}$	0.269 ± 0.015
	Mem0	17.17 ± 2.24	6.24 ± 2.24	60.0 ± 6.4	4.39 ± 0.37	3.405 ± 0.280	0.599 ± 0.157
	Tycho (adapted)	19.53 ± 0.52	9.20 ± 0.52	$\mathbf{78.5 \pm 2.2}$	5.96 ± 0.26	4.844 ± 0.237	0.305 ± 0.027
	EBM (w/o stale)	23.09 ± 2.86	11.29 ± 2.86	66.5 ± 7.2	2.10 ± 0.12	4.863 ± 0.192	0.328 ± 0.015
	EBM (w stale)	$\mathbf{25.16 \pm 3.08}$	$\mathbf{12.16 \pm 3.08}$	73.0 ± 8.2	$\mathbf{2.09 \pm 0.16}$	5.139 ± 0.227	0.333 ± 0.034
Sonnet4.6	ICL	17.89 ± 1.19	7.36 ± 1.19	53.0 ± 4.1	2.40 ± 0.14	4.338 ± 0.341	$\mathbf{0.027 \pm 0.004}$
	ReAct	19.13 ± 2.63	8.40 ± 2.63	56.0 ± 8.0	2.28 ± 0.12	4.176 ± 0.371	$\mathbf{0.027 \pm 0.010}$
	ACE	19.69 ± 2.17	5.63 ± 2.17	59.0 ± 5.2	2.66 ± 0.30	1.687 ± 0.121	0.089 ± 0.006
	DC	21.11 ± 3.13	11.11 ± 3.13	64.0 ± 9.9	2.78 ± 0.18	$\mathbf{1.283 \pm 0.100}$	0.255 ± 0.012
	Mem0	12.16 ± 0.84	0.23 ± 0.84	62.5 ± 2.5	7.34 ± 0.16	4.932 ± 0.110	0.049 ± 0.002
	Tycho (adapted)	16.48 ± 2.21	5.35 ± 2.21	61.0 ± 6.5	4.70 ± 0.61	2.144 ± 0.297	0.124 ± 0.020
	EBM (w/o stale)	23.28 ± 2.70	12.28 ± 2.70	67.0 ± 7.8	2.25 ± 0.21	6.154 ± 0.603	0.150 ± 0.035
	EBM (w stale)	$\mathbf{23.41 \pm 3.33}$	$\mathbf{13.48 \pm 3.33}$	$\mathbf{67.5 \pm 8.5}$	$\mathbf{2.14 \pm 0.21}$	5.840 ± 0.400	0.160 ± 0.020
Opus4.8	ICL	22.55 ± 4.01	9.75 ± 4.01	66.0 ± 11.4	2.31 ± 0.13	5.391 ± 0.560	$\mathbf{0.028 \pm 0.012}$
	ReAct	27.13 ± 1.51	11.27 ± 1.51	78.5 ± 3.8	$\mathbf{2.12 \pm 0.13}$	5.244 ± 0.569	0.035 ± 0.017
	DC	23.12 ± 2.78	8.99 ± 2.78	68.5 ± 7.6	2.50 ± 0.11	$\mathbf{1.597 \pm 0.066}$	0.322 ± 0.008
	Tycho (adapted)	25.45 ± 1.50	11.45 ± 1.50	75.5 ± 4.8	2.38 ± 0.10	1.940 ± 0.112	0.073 ± 0.005
	EBM (w/o stale)	27.71 ± 1.58	11.17 ± 1.58	80.0 ± 4.7	2.23 ± 0.12	7.173 ± 0.668	0.159 ± 0.026
	EBM (w stale)	$\mathbf{28.61 \pm 1.52}$	$\mathbf{13.61 \pm 1.52}$	$\mathbf{82.5 \pm 4.0}$	2.27 ± 0.08	7.088 ± 0.707	0.161 ± 0.016

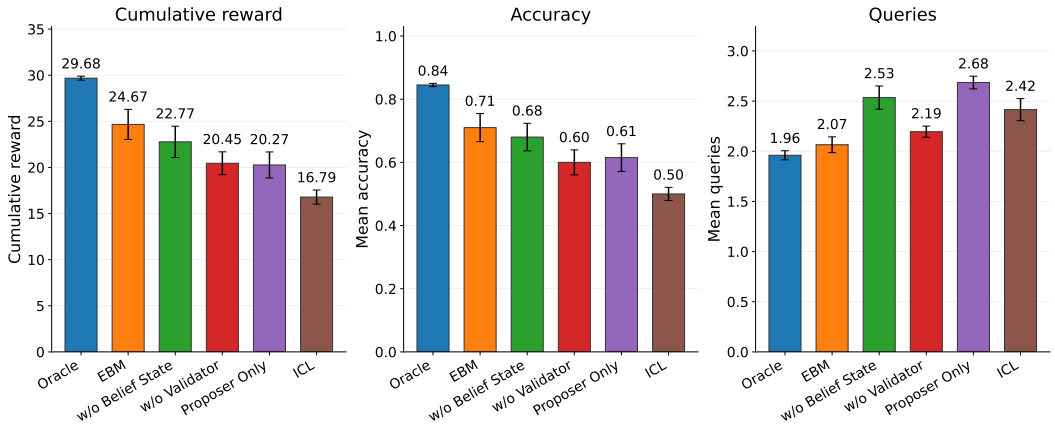

Figure 6: Component ablation of EBM on DATABASE with Qwen3.8-27B. We report cumulative reward, mean accuracy, and mean query count. Results are averaged over five independent runs, with error bars denoting ±1 SE. *w/o Belief State* removes the persistent belief state from both the proposer and solver; *Proposer Only* additionally removes the validator; *w/o Validator* retains the persistent belief state but removes validation. The oracle is included as a reference for the remaining headroom.

A.4.3 COMPONENT ABLATION

Figure 6 provides the complete component-ablation metrics corresponding to Figure 5a.

Figure 7 provides the raw-history-window analysis referenced in the main text.

Table 4: Detailed results on the Poker environment. Cumulative rewards and gains are measured in big blinds (BB). Results are averaged over five runs and reported as mean ± standard deviation. Higher reward and gain are better, while lower token usage is better. Best mean results within each backbone are bold; this does not indicate statistical significance. Tycho (adapted) used only its textual notepad.

Model	Method	Reward (BB) ↑	Gain (BB) ↑	Input (M) ↓	Output (M) ↓
Qwen3.8	ICL	78.40 ± 98.41	-5.00 ± 98.41	3.197 ± 0.449	$\mathbf{0.079 \pm 0.015}$
	ReAct	60.20 ± 48.96	-156.00 ± 48.96	3.712 ± 0.643	0.088 ± 0.028
	ACE	76.34 ± 65.35	23.34 ± 65.35	0.816 ± 0.198	0.177 ± 0.046
	DC	12.40 ± 25.63	-103.00 ± 25.63	$\mathbf{0.754 \pm 0.108}$	0.396 ± 0.075
	Mem0	55.98 ± 22.32	-49.02 ± 22.32	2.405 ± 0.539	0.379 ± 0.093
	Tycho (adapted)	$\mathbf{290.90 \pm 349.85}$	$\mathbf{88.20 \pm 349.85}$	6.906 ± 1.890	0.796 ± 0.279
	EBM	141.06 ± 101.08	41.06 ± 101.08	13.018 ± 2.076	0.628 ± 0.150
Sonnet4.6	ICL	172.82 ± 26.40	-32.88 ± 26.40	3.382 ± 0.254	0.023 ± 0.003
	ReAct	193.28 ± 29.81	-15.62 ± 29.81	2.916 ± 0.259	$\mathbf{0.012 \pm 0.001}$
	ACE	206.56 ± 45.81	23.06 ± 45.81	1.420 ± 0.211	0.102 ± 0.013
	DC	110.38 ± 32.63	-87.42 ± 32.63	$\mathbf{1.019 \pm 0.101}$	0.240 ± 0.019
	Mem0	203.32 ± 38.08	-35.68 ± 38.08	1.977 ± 0.061	0.026 ± 0.001
	Tycho (adapted)	223.32 ± 23.09	$\mathbf{52.82 \pm 23.09}$	1.423 ± 0.107	0.078 ± 0.015
	EBM	$\mathbf{238.16 \pm 76.26}$	0.46 ± 76.26	12.730 ± 0.360	0.110 ± 0.021
Opus4.8	ICL	89.90 ± 14.75	-3.80 ± 14.75	7.117 ± 0.512	$\mathbf{0.039 \pm 0.002}$
	ReAct	85.84 ± 17.70	16.24 ± 17.70	7.580 ± 0.977	$\mathbf{0.039 \pm 0.004}$
	DC	56.52 ± 36.59	-37.88 ± 36.59	$\mathbf{1.490 \pm 0.162}$	0.316 ± 0.027
	Tycho (adapted)	143.52 ± 38.99	45.42 ± 38.99	3.051 ± 0.155	0.087 ± 0.004
	EBM	$\mathbf{159.12 \pm 49.51}$	$\mathbf{65.12 \pm 49.51}$	19.276 ± 1.165	0.184 ± 0.008

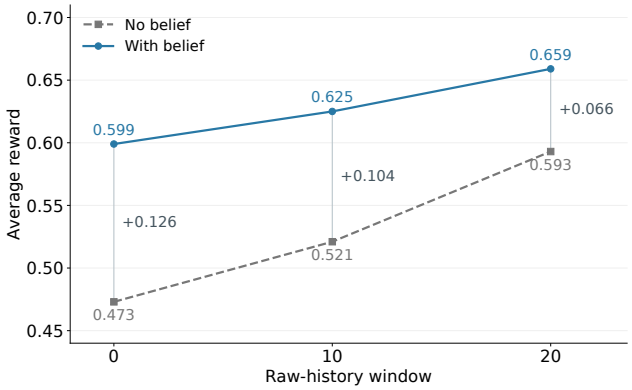

Figure 7: Effect of raw-history window size on the utility of persistent beliefs. Reward is evaluated with and without persistent beliefs while varying the number of recent task instances directly available to the model. We use only the first 20 no-drift DATABASE instances, so the larger gain at shorter history windows cannot be explained by removing stale pre-drift information. The experiment measures the marginal utility of the persistent representation, but does not distinguish belief-state tracking from other forms of useful history compression.

A.5 ADDITIONAL EXAMPLE OF BELIEF REVISION

Table 5 shows a second form of belief revision, where broader evidence weakens an initially overgeneralized relation. The core relation remains supported but is narrowed in scope as additional evidence reveals systematic exceptions.

Table 5: **Belief refinement in EBM.** An example of a persistent belief revised after broader interaction reveals more evidence. The full-table counts establish that `prc_usd` and `prc/100` are not universally equivalent.

Earlier belief	Status change	Updated belief
`prc_usd` equals `prc/100`; the two fields are treated as equivalent representations of list price.	SUPPORTED → SUPPORTED	`prc_usd` equals `prc/100` for most rows, but the two fields are not universally equivalent.

Observed evidence. Among 6,570 rows with positive `prc`, 6,346 satisfy `prc_usd = prc/100`, while 224 either violate the equality or have `prc_usd IS NULL`.

A.6 JUDGE PROMPT FOR FACTUAL CORRECTNESS

We use GPT-6-Astra as the judge to assess the factual correctness of each persistent belief. The evaluated beliefs are sampled from five independent runs.

The following templates present the prompts used in RQ4 of the main text to evaluate each belief against the corresponding ground truth. To avoid biasing the judgment, we provide the judge only with the belief statement and do not reveal the status assigned by the validator.

GPT-6-Astra Judge

```
You are a factual judge evaluating beliefs about a database
    environment.

You will receive:
1. One belief, including its statement.
2. Ground-truth properties of the database, each with a unique
    property ID.
3. The applicable database version.

Judge whether the belief's content is correct relative to the
    supplied ground-truth properties.

Return:
- "true": GT supports all substantive assertions in the belief.
- "false": GT directly contradicts at least one substantive
    assertion.
- "ambiguous": Neither applies, including insufficient GT coverage
    or unclear scope.

Respect table, column, entity group, and database phase. Missing
    information is not evidence of falsity.

Input:
Belief: {belief}
Ground-truth properties: {gt_properties}

Return JSON only:
{"label": "true|false|ambiguous", "reason": "Brief justification."}
```

A.7 SHARED PROMPT TEMPLATES

The following templates provide the shared instructions for each role. Task-specific interface descriptions and output schemas are omitted from these templates.

Proposer

```
You maintain reusable, evidence-grounded claims about an environment
    across an ordered sequence of task instances. Your role is to
    improve claim memory and understanding of the environment, not
    to solve the current instance.

You are given the current task and visible trajectory, existing
    claims, and historical experiences with eligible evidence
    records.

PROPOSAL PROCEDURE

1. Read all existing claims before proposing changes. Identify
    semantic equivalence even when wording or keys differ. Never add
    a duplicate. Prefer revising an existing claim when updating the
    same fact.

2. Retrieve past complete experiences relevant to the current task.
    Match evidence to the entities, conditions, and scope involved.

3. For past failures, trace backward to the earliest questionable
    assumption. Distinguish errors in the environment model from
    execution mistakes, random outcomes, overly broad scope, biased
    observations, and environmental change. Final correctness or
    reward is a cue for investigation, not direct evidence for an
    environment claim.

4. Prioritize a relevant failure's root cause, a missing
    prerequisite for the current decision, an outdated or
    conflicting claim, and then other reusable knowledge.

5. Make each claim atomic, independently falsifiable, and scoped as
    narrowly as its evidence warrants. Separate facts that can
    change independently. Express uncertainty or probabilistic
    tendencies when appropriate; finite observations do not justify
    "always" or "never."

6. Ground each proposed mutation in exactly one eligible evidence
    record. Multiple mutations may cite the same record. If active
    verification is permitted, a mutation may instead propose a
    diagnostic probe that distinguishes plausible alternatives,
    subject to the interface rules and available budget.

7. Store reusable environment knowledge. Do not store benchmark
    answers, one-off solutions, restatements of the current task, or
    instructions for the solver's strategy. Do not invent hidden
    facts or identifiers.

MEMORY OPERATIONS

ADD: introduce a new claim with a scope, stable semantic key, and
    atomic statement.

REVISE/revalidate: reassess an existing claim using new evidence.

REVISE/replace: correct an existing claim within the same semantic
    slot.
```

Proposer (continued)

```
REVISE/invalidate: identify direct evidence against an existing
    claim without asserting a replacement.

NOOP: make no change when no relevant, reusable mutation is
    justified.

CLAIM STATUSES

HYPOTHESIS: uncertain and requires further evidence.
SUPPORTED: backed by usable evidence within its stated scope.
CONTRADICTED: directly refuted by evidence.

<task interface>

<output format>
```

Validator

```
You validate a batch of proposed atomic claims about an environment.
    Judge what the evidence establishes, not whether a claim is
    useful for solving the current instance or recommends a good
    action.

You are given proposed mutations, existing claims, and the evidence
    record cited by each mutation. Final correctness feedback or
    reward provides context but cannot replace the cited evidence.

VALIDATION PROCEDURE

1. Judge every mutation independently, including mutations that cite
    the same evidence record. Check that the evidence is eligible
    and pertains to the claim's entities, conditions, and scope.

2. Assign one evidence verdict: SUPPORT when the evidence supports
    the complete scoped statement; REFUTE when the evidence directly
    refutes the statement; INCONCLUSIVE when neither conclusion is
    justified.

3. Match the strength of the conclusion to the observations.
    Distinguish deterministic assertions from probabilistic
    tendencies. An observation inconsistent with a tendency does not
    automatically refute it. Account for counterexamples, selection
    bias, hidden information, and outcome variance.

4. Check existing claims for semantic duplicates and genuine
    conflicts. Similar wording, overlapping scope, or
    general-versus-specific claims do not automatically imply a
    conflict. Do not favor a claim merely because it is newer.
    Report a conflict only when the evidence supporting the proposed
    mutation also directly refutes the existing claim.

STATUS AND REVISION RULES

ADD: SUPPORT results in SUPPORTED; REFUTE results in CONTRADICTED;
    INCONCLUSIVE results in HYPOTHESIS.

REVISE/revalidate: SUPPORT results in SUPPORTED; REFUTE results in
    CONTRADICTED; INCONCLUSIVE preserves the target's current status.
```

Validator (continued)

```
REVISE/replace: SUPPORT results in SUPPORTED; INCONCLUSIVE results
    in HYPOTHESIS. A refuted replacement must not be applied;
    preserve the target.

REVISE/invalidate: apply only when the evidence directly refutes the
    target, setting its status to CONTRADICTED.

A semantic duplicate must not be applied. Identify the existing
    duplicate claim.

<task interface>

<output format>
```

Solver

```
Solve the current task using the visible state, task instructions,
    available actions, and supplied claims. Optimize the task's
    stated objective while respecting action constraints and budgets.

CLAIM STATUSES

SUPPORTED: Can be used when relevant, but current observations and
    task rules take priority.
HYPOTHESIS: Treat as uncertain. Verify when materially relevant and
    verification is allowed/useful.
CONTRADICTED: Validated as false.

Resolve material uncertainty with permitted actions when useful and
    within budget.


<task interface>

<output format>
```

B TASK-SPECIFIC EVIDENCE ACQUISITION

In Eq. 1, P includes the acquisition of evidence for each candidate mutation. A returned pair $(m_{t,i}, e_{t,i})$ contains retrieved or observed evidence, not merely a request for a probe. The acquisition mechanism depends on the environment's interaction constraints.

DATABASE. The database environment supports both history-based evidence retrieval and active verification. For a proposed mutation $m_{t,i}$, the proposer may retrieve relevant evidence from previously observed interactions,

$$e_{t,i} = \textsc{RetrieveEvidence}(H_{t-1}, m_{t,i}),$$

when the accumulated history already contains an eligible observation for evaluating the proposal.

Alternatively, when additional verification is useful, the proposer may issue a permitted verification query targeted at the proposed mutation:

$$a^{\text{ver}}_{t,i} = \textsc{ConstructVerification}(x_t, m_{t,i}), \qquad e_{t,i} = \textsc{ExecuteAndObserve}(a^{\text{ver}}_{t,i}).$$

Importantly, in the database experiments, such verification calls consume the same query budget as ordinary task-solving queries and therefore directly affect the final reward. Active verification thus does not provide additional free opportunities to explore the database; rather, the proposer must

trade off the potential value of obtaining targeted evidence against the cost of spending a query that could otherwise be used for answering the task. The resulting mutation–evidence pair is then passed to the validator.

POKER. The poker environment does not permit additional information-gathering actions solely for belief verification. Consequently, evidence for each proposed mutation must be retrieved from interactions that have already occurred:

$$e_{t,i} = \textsc{RetrieveEvidence}(H_{t-1}, m_{t,i}).$$

In both settings, the proposer may return no candidate updates, i.e., $\Delta B_t = \emptyset$. The acquisition equations describe candidates for which non-empty eligible evidence can be obtained under the task interface; this evidence may still be inconclusive. They do not specify fallback behavior when acquisition fails or how verification calls are charged against the task budget. Those details require the task-specific interfaces omitted from the shared prompt templates (Appendix A).

C BELIEF UPDATE RULES

The validator distinguishes an evidence verdict from the stored proposition status. Its verdict is SUPPORT, REFUTE, or INCONCLUSIVE; the resulting status also depends on the proposed operation. Table 6 summarizes the shared validator prompt. Semantic duplicates are not applied. Each mutation is assessed independently, including when multiple mutations cite the same evidence record. Final correctness or reward can motivate an investigation but does not replace evidence for the scoped environment proposition.

The manuscript label HYPOTHESIZED corresponds to the constant `HYPOTHESIS` in the shared prompts.

Table 6: Status transitions specified by the shared validator prompt. "Preserve" means that the target proposition and its current status remain unchanged. A replacement changes the statement within the same semantic slot. The task-specific STALE extension is described separately in Appendix A.

Operation	SUPPORT	REFUTE	INCONCLUSIVE
Add	SUPPORTED	CONTRADICTED	HYPOTHESIZED
Revalidate	SUPPORTED	CONTRADICTED	Preserve
Replace	SUPPORTED	Preserve	HYPOTHESIZED
Invalidate	Preserve	CONTRADICTED	Preserve

For example, an inconclusive revalidation leaves a HYPOTHESIZED belief unresolved and leaves a SUPPORTED proposition supported. The latter transition does not encode a reduction in confidence: the shared state representation uses discrete assessments rather than calibrated probabilities. A proposed replacement with inconclusive evidence instead produces a new HYPOTHESIZED belief; a refuted replacement is rejected and the original target is preserved. These rules permit later evidence to revise prior assessments without requiring each new task to reconstruct all propositions from raw history.