# LEARNING WHAT TO REMEMBER: UTILITY-GUIDED MEMORY FOR LONG-HORIZON TEXT-TO-SQL

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Memory systems have proven beneficial in a variety of long-horizon settings. As text-to-SQL research increasingly extends to multi-turn interactions, these systems must retain user-specific knowledge across turns, creating a need for effective memory management. Text-to-SQL presents both opportunities and challenges for memory: each completed round yields a structured record comprising a request, SQL program, execution outcome, and possible correction, while interactions tend to revisit a bounded set of schema-grounded concepts. However, data reflecting real user workloads in long-horizon, multi-turn settings remains limited. Leveraging structured interactions, we represent completed rounds as standardized memory entries and formulate bounded text-to-SQL memory management as an eviction problem. We further exploit schema concentration through P-UTL, a lightweight, text-to-SQL-native policy that predicts the utility of stored information using execution-based supervision. To address the lack of workload simulators, we introduce a controlled trajectory generator that varies dormancy, recurrence, information density, and memory pressure across extended SParC- and CoSQL-derived streams. Experiments across workload configurations and baselines demonstrate the advantages of our method and show that predictive utility is particularly valuable when useful information competes for bounded memory.

## 1 INTRODUCTION

Text-to-SQL translates natural-language questions into executable SQL programs, providing a direct interface between users and relational databases. Recent advances in language models have improved performance on complex schemas and unseen domains (Yu et al., 2018), as well as on analytical requests (Yu et al., 2019a;b). Consequently, increasing attention has focused on extending text-to-SQL to realistic user workloads (Yin et al., 2026), which frequently involve multiple turns and longer interaction horizons. These settings require effective context management.

Meanwhile, long-horizon memory agents have received increasing research attention, with prior work showing promising results on generic long-context benchmarks (Yu et al., 2026). Recent studies have also evaluated memory through incremental multi-turn interactions (Hu et al., 2026) and in long-running agents (Zhao et al., 2026). In this context, long-horizon text-to-SQL is both feasible and valuable: users may establish private definitions, analytical conventions, or reporting criteria that remain relevant after many unrelated requests. The consequences of forgetting such information can be catastrophic—SQL queries might be generated incorrectly.

Long-horizon, multi-turn text-to-SQL presents both opportunities and challenges. It offers two properties a task-native memory system can exploit. First, its interactions are *structured*: each completed round can be represented by the same combination of a natural-language request, SQL program, execution result, and possible correction. Second, long-running analytics is often *schema-concentrated*: a trajectory operates over one or a few schemas and repeatedly encounters a constrained set of entities, conventions, and business concepts. This setting differs from generic interactions, in which the volume and content of incoming information are less constrained. However, interactions in existing text-to-SQL benchmarks are too short to support long-horizon evaluation. Recent work continues to adapt conversational text-to-SQL datasets containing only three to five rounds (Guo et al., 2026). Moreover, existing benchmarks do not provide practical simulations of how users introduce and later reuse analytical definitions and conventions.



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.

In this work, we seek to utilize these properties and address the evaluation challenges. Specifically, retaining or dropping elements in the interaction structure in a limited-size memory pool can be formulated as an online eviction problem. Schema concentration makes memory utility more predictable, while executable SQL outcomes provide task-grounded supervision for learning. Prior work on modeling user behaviour further suggests that parametric distributions support the simulation of extended user trajectories (Breslau et al., 1999; Arlitt & Williamson, 1996).

We therefore introduce P-UTL, a lightweight policy that predicts memory utility from counterfactual SQL execution outcomes and uses it to guide online eviction. To evaluate the policy in multi-turn setups, we introduce a controlled trajectory generator that embeds verbatim and convention callbacks in extended SParC and CoSQL streams (Yu et al., 2019b;a). It varies callback timing, density, load, and memory capacity, enabling reproducible stress tests across diverse recurrence regimes, which collectively represent a variety of practical long-running text-to-SQL workloads.

Our contributions are:

1. We formulate structured, bounded text-to-SQL memory management as online eviction and propose P-UTL, a lightweight policy that learns memory utility from the counterfactual effect of a memory on future SQL execution.
2. We introduce a controlled trajectory-generation framework for execution-grounded, long-horizon text-to-SQL that supports verbatim and convention callbacks and configurable dormancy, density, callback, and capacity regimes.
3. We conduct an extensive empirical study across these controlled regimes using representative heuristic, LLM-based, and open-weight memory systems as baselines, demonstrating the advantages of our utility-based method.

## 2 Related Work

### 2.1 Memory agents

Research on memory typically seeks to enable agents to operate beyond the context available in a single interaction. Existing systems consolidate experience into reflections or user memories (Park et al., 2023; Zhong et al., 2023), organize context into hierarchical or linked stores (Packer et al., 2024; Hu et al., 2024; Xu et al., 2025), or learn to update and compress memory jointly with reasoning (Zhang et al., 2026; Yu et al., 2026; Zhou et al., 2025). Long-term-memory benchmarks accordingly evaluate retrieval, temporal reasoning, updating, and robustness over sustained interaction (Wu et al., 2025; Maharana et al., 2024; Hu et al., 2026; Zhao et al., 2026). However, they generally measure language-level recall or overall agent success rather than the executable consequence of retaining an individual memory.

Memory has also begun to receive task-specific treatment in text-to-SQL. AgentSM stores structured execution traces as semantic programs that can guide future reasoning, while EvoSQL maintains contextualized candidate memories during iterative critic–generator refinement (Biswal et al., 2026; Zhou et al., 2026). MTSQL-R1 combines database feedback with persistent dialogue memory to train iterative verification and correction, whereas EnterpriseMem-Bench studies the effects of working memory, episodic retrieval, and semantic augmentation (Guo et al., 2026; Tummalapenta & Addanki, 2026). Semantic indexing has also been explored in deployed text-to-SQL memory (Yin et al., 2026). Whereas these systems emphasize trace reuse, within-task refinement, or architectural comparison, we formulate sustained text-to-SQL memory as bounded admission and eviction and use actual execution outcomes to supervise the utility for later rounds.

### 2.2 Multi-turn text-to-SQL

Spider established large-scale, cross-domain evaluation over unseen databases (Yu et al., 2018); BIRD broadened evaluation to large-scale, database-grounded interfaces (Li et al., 2023); and Spider 2.0 introduced enterprise schemas, multiple SQL dialects, metadata, and project-level artifacts (Lei et al., 2025). BIRD-INTERACT further evaluates dynamic interaction, including clarification, environment exploration, and recovery from execution errors (Huo et al., 2026). These benchmarks

increase schema, reasoning, and workflow complexity, but such complexity does not necessarily test whether bounded memory preserves user information across a long stream of unrelated queries.

SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) provide foundational multi-turn settings, but their context remains comparatively local. SParC averages roughly three questions per sequence, CoSQL approximately ten utterances per dialogue, and EnterpriseMem-Bench approximately 4.7 turns per session (Tummalapenta & Addanki, 2026). They therefore provide limited support for studying information that remains dormant across hundreds of queries, competes for bounded capacity, and later becomes execution-critical, while offering little control over long-horizon recurrence.

Since real long-horizon text-to-SQL usage patterns remain largely unknown, we draw on prior work using configurable distributions to evaluate database and session workloads (Ding et al., 2021; Vögele et al., 2018; Han et al., 2023), stochastic demand (Wolff, 1982), and temporally concentrated activity (Denning, 1968; Leland et al., 1994; Crovella & Bestavros, 1997; Arlitt & Williamson, 1996; Breslau et al., 1999). We use different distributions, such as exponential, poisson, etc., as dormancy families to controll recurrence regimes and vary concentration and tail behavior.

# 3 METHODOLOGY

## 3.1 PROBLEM SETUP: TEXT-TO-SQL AS MEMORY EVICTION

**Long-horizon interaction.** We consider a text-to-SQL agent that serves one user over a trajectory of $T$ rounds, $\tau = (r_1, \ldots, r_T)$, against a relational database $D$. At round $t$, the user issues a natural-language request $q_t$ and may subsequently provide a natural-language correction $c_t$. Conditioned on the database schema, the current request, and retrieved history, the SQL model produces a final query $y_t$. We write a completed round as

$$r_t = (q_t, c_t, y_t, e_t), \tag{1}$$

where $e_t = \mathcal{E}(y_t, D)$ is the execution result and $c_t = \emptyset$ when no correction is given. Unlike independent text-to-SQL, the intended semantics of $q_t$ may depend on a definition, preference, or correction introduced many rounds earlier.

**Memory environment.** The unitized structure of text-to-SQL turns each completed round into a standardized candidate memory

$$m_t = g(q_t, c_t, y_t, e_t), \tag{2}$$

where $g$ is a shared memory-writing procedure. Each entry contains a textual record and a memory-type tag, assigned by the shared writer from the completed round's execution outcome. Candidate memories do not have to contain equal semantic information, but they expose the same interface to the memory policy. The agent maintains a persistent pool $\mathcal{M}_t$ containing at most $B$ entries. Before answering $q_t$, a fixed retriever $R_K$ selects the $K$ entries most similar to the current request,

$$\mathcal{R}_t = R_K(q_t, \mathcal{M}_t), \tag{3}$$

and supplies them to the SQL model together with the schema. The newly written $m_t$ becomes available only after the round is complete and therefore cannot help answer the request from which it was constructed.

Writing $m_t$ produces the candidate pool $\mathcal{C}_t = \mathcal{M}_t \cup \{m_t\}$. If $|\mathcal{C}_t| \leq B$, every entry can be retained. Otherwise, a memory policy $\pi$ must choose a subset of exactly $B$ entries:

$$\mathcal{M}_{t+1} = \pi(\mathcal{C}_t), \qquad |\mathcal{M}_{t+1}| = B. \tag{4}$$

This operation defines admission and eviction. The policy may reject $m_t$ immediately, admit it by evicting an older entry, or preserve the current pool unchanged. Once discarded, an entry is unavailable to future retrieval. Within our fixed-slot comparison, methods share the SQL model, writer $g$, retriever $R_K$, and capacity $B$. They differ in the policy $\pi$. Thus, standardized candidate units and bounded capacity turn the primary long-horizon text-to-SQL memory comparison into an online eviction problem. The left panel of Figure 1 summarizes this interaction and the resulting admission–eviction decision.

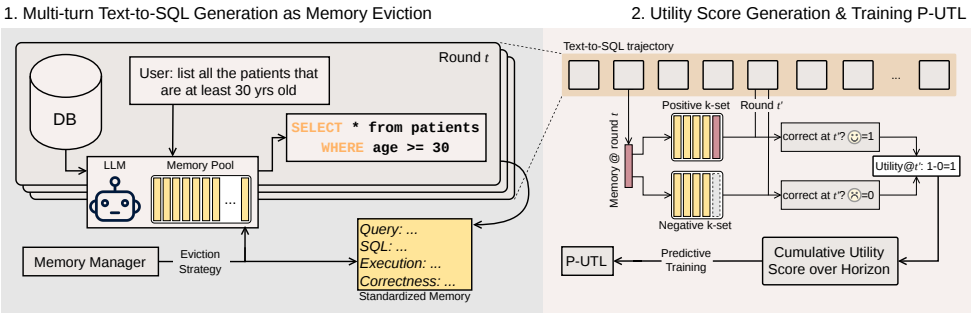


Figure 1: Overview of the proposed framework. The left panel formulates multi-turn text-to-SQL memory as bounded admission and eviction over standardized completed rounds. The right panel shows how counterfactual executions produce cumulative utility supervision for training P-UTL.

**Objective.** Let $y_t^*$ be the reference SQL and let $\mathcal{E}(y, D)$ denote the result of executing $y$ on $D$. Round-level execution accuracy is defined as

$$A(y_t, y_t^*; D) = \mathbb{I}[\mathcal{E}(y_t, D) = \mathcal{E}(y_t^*, D)]\,. \tag{5}$$

The global objective of $\pi$ is to maximize execution accuracy over future rounds, not merely to reconstruct past text:

$$\max_{\pi}\ \mathbb{E}_{\tau}\left[\frac{1}{T}\sum_{t=1}^{T} A(y_t, y_t^*; D)\right]. \tag{6}$$

Direct optimization of Equation 6 would require modeling future requests and the joint consequences of sequential memory decisions. We instead learn a supervised ranking surrogate: the marginal change in execution accuracy caused by removing a memory from a retrieved prompt during offline replay with an unbounded store. This surrogate does not estimate the full downstream effect of an eviction decision. Section 3.2 defines the predictor and retention rule, and Section 3.3 describes the controlled workloads used to evaluate them.

### 3.2 Predicting Memory Utility

Long-running text-to-SQL interactions are often schema-concentrated: requests revisit a constrained collection of entities, conventions, and business concepts. Our hypothesis is that this recurrence makes the utility of a completed round learnable from its content and write-time context. Our policy, P-UTL, estimates that utility and uses it to update the bounded pool. The right panel of Figure 1 shows an overview of the counterfactual supervision and predictive training described below.

**Marginal counterfactual execution utility.** Consider a memory $m$ written at round $t$ and a future request $q_k$ for which $m$ is retrieved. We measure its counterfactual utility using two outputs, $y_k^{+m}$ and $y_k^{-m}$. The former is the SQL produced with a retrieved set of $K$ memories that includes $m$, such that $\mathcal{R}_{t'}^{+} = \{m, \ldots\}$ and $|\mathcal{R}_{t'}^{+}| = K$. The latter is produced using the same set with $m$ removed, such that $\mathcal{R}_{t'}^{-} = \mathcal{R}_{t'}^{+} \setminus \{m\}$. We define the local execution utility of $m$ as

$$\Delta(m, k) = A(y_k^{+m}, y_k^*; D) - A(y_k^{-m}, y_k^*; D) \in \{-1, 0, 1\}. \tag{7}$$

A positive value means that access to $m$ changes an incorrect execution into a correct one. A negative value means that its presence is harmful. Zero means that it does not change execution correctness. This intervention measures the marginal utility of $m$ in the observed prompt. It does not model the item that a different pool might retrieve in its place or higher-order interactions among memories.

The future utility of $m$ over horizon $H$ is the sum of its local effects on future requests for which it is retrieved:

$$U_H(m, t) = \sum_{k=t+1}^{\min(t+H,T)} \mathbb{I}[m \in R_K(q_k, \mathcal{M}_k)]\,\Delta(m, k). \tag{8}$$

If $m$ is never retrieved in this horizon, its observed utility is zero. We convert the accumulated value into the three-way target $z(m) = \text{sign}(U_H(m,t))$, denoting a harmful, neutral, or helpful memory. The sign is a coarse surrogate: it preserves the direction of the observed effect but discards its magnitude and recurrence frequency. Labels are constructed offline from training trajectories using future reference queries and counterfactual SQL executions. Neither future queries nor labels are exposed at deployment. Appendix A.2 specifies the replay policy and labeling procedure, and the design choices above are examined through ablation studies in Appendix C.3.

**Utility prediction and online eviction.** Since $U_H(m,t)$ depends on future requests, it cannot be computed when $m$ is written. We instead learn a predictor $p_\theta(z \mid x_m)$ from information $x_m$ observable at write time. Its scalar utility score is defined as

$$S_\theta(m) = p_\theta(z = +1 \mid x_m) - p_\theta(z = -1 \mid x_m). \tag{9}$$

The score increases when a memory is more likely to improve a future execution and decreases when it is likely to be harmful. The formulation is independent of predictor architecture. Section 4.1 describes how we configure the predictor in our experiments.

At runtime, $x_m$ is a static write-time feature vector. At every overflow boundary, P-UTL scores the entries in $\mathcal{C}_t$ using these fixed features and retains the $B$ highest-utility candidates. A new entry is rejected if its utility does not place it in the pool, whereas an older entry can survive multiple boundaries while its score remains among the top $B$.

### 3.3 Controlled long-horizon trajectory generation

Existing conversational text-to-SQL data provide realistic local dependencies but comparatively short interactions. To evaluate eviction over a genuinely long horizon and under different recurrence patterns, we construct persistent trajectories from SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a), which share the Spider database collection (Yu et al., 2018). We normalize each source interaction into natural-language requests with executable reference SQL and stitch interactions over the same database into a persistent analyst stream. Source interactions remain locally contiguous, while transitions between them provide unrelated background activity and new candidate memories.

The generator inserts initialization–callback units into this stream. Each unit introduces an anchor, leaves it dormant for a sampled background-stream gap $d \sim \mathcal{D}$, and later makes successful SQL execution depend on that. A **verbatim** callback reissues an earlier query with the same reference program, i.e., $q_t = q_{t+d}$. A **convention** callback first assigns a user-facing concept to an executable predicate, exercises that definition in adjacent warm-up queries, and later uses the name in a different query without restating the predicate. It therefore tests semantic reuse rather than string repetition.

The unitized stream lets the generator control candidate arrival and delayed reuse separately. We sample $d$ from exponential, Poisson, or normal families, vary the number of initialized conventions $C$, use definition spacing $s$ to control how densely critical candidates arrive, and vary the memory budget $B$. These distributions define controlled recurrence regimes rather than exact models fitted to observed text-to-SQL users. All policies within a condition receive identical requests, reference programs, callback locations, and realized gaps. Appendix A provides the exact construction, validity checks, gap accounting, and callback examples. Appendix B specifies the workload configurations.

## 4 Experiments

### 4.1 Experimental setup

**Datasets and metrics.** We construct long-horizon workloads from SParC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) over the Spider databases (Yu et al., 2018), following Section 3.3. The resulting corpus contains 38 database–analyst trajectories over 20 databases. We split the databases into 10 training, 4 validation, and 6 test databases. At testing, for each distributional configuration, we generate 15 long trajectories for every parameter setting. Appendix B.1 reports full statistics.

Our primary quality metric is the accuracy. Predicted and reference SQL queries are executed on the corresponding database, and their canonicalized results are compared while preserving order when the reference contains `ORDER BY`. We compute 95% confidence intervals using 20,000 bootstrap

resamples at the trajectory level. Each resample draws complete analyst trajectories with replacement, retaining callbacks from the same trajectory to account for within-trajectory dependence.

**Baselines.** We compare our **p**redictor–**uti**l**i**ty policy, P-UTL, with three classes of memory systems. P-UTL learns an execution-supervised utility score and retains the $B$ candidates with the highest predicted utility. The heuristic baseline H-LRU (**H**euristic–**LRU**) evicts the item that was retrieved least recently. The API-based methods include E-LLM (**E**viction–**LLM**), which asks an LLM to select exactly $B$ items; E-Sum (**E**viction–**sum**marization), which continually compresses memory into a token-bounded textual state; and ACE, an adaptation of Agentic Context Engineering (Zhang et al., 2026). We also evaluate MemAgent-7B (Yu et al., 2026) and MEM1 (Zhou et al., 2025). A stateless agent is included in selected experiments as a no-memory control. Because E-Sum, MemAgent-7B, and MEM1 maintain textual states rather than discrete slots, we compare them at the task level but omit them from fixed-slot analyses.

**Implementation details.** All methods use the pinned `gpt-4o-mini` model as the SQL answerer for cost efficiency. Using the same backbone across methods ensures a controlled comparison. E-LLM, E-Sum, and ACE use the same model for their primary management calls unless otherwise noted. Holding both the answerer and API-manager backbones fixed further isolates the memory policy. MemAgent-7B uses `RL-MemoryAgent-7B`, while MEM1 uses `Qwen2.5-7B-RL-RAG-Q2-EM-Release`.

For P-UTL, we instantiate the utility predictor as a lightweight MLP. The reported models use static write-time features derived from memory content, memory type, schema references, recent-query context, and usage state; they exclude privileged generator metadata, such as convention IDs and scheduled gaps. At inference time, all fixed-slot methods use top-$K = 5$ retrieval and $B = 16$ unless otherwise noted. SQL trajectory calls use a temperature of 0.2, whereas manager decisions and offline counterfactual calls use a temperature of 0. Table 5 reports only online memory-decision latency and API cost. The labeling procedure is detailed in Appendix A.2; additional hyperparameters, prompts, and implementation details are provided in Appendix B.

**Research questions.** We attempt to address several research questions. First, does persistent memory improve repeated and ordinary text-to-SQL queries without introducing a performance trade-off (Scenario 1, Section 4.2)? Second, how do memory systems behave across controlled recurrence regimes with different dormancy characteristics (Scenario 2, Section 4.3)? Third, how do memory systems behave when controlled recurrence regimes suggest harder scenarios(Scenario 3, Section 4.4)? We also conduct ablations on training workloads and the effect of backbone capability on API-based methods (Section 4.5), as well as on utility-score design choices (Appendix C.3).

### 4.2 Scenario 1: Is memory useful?

Scenario 1 asks whether persistent memory improves text-to-SQL before introducing the epistemic ambiguity of a user-defined convention. We therefore use verbatim callbacks: each trajectory repeats earlier self-contained questions after a delay sampled uniformly from 15–50 turns. Since each callback preserves the anchor request and reference SQL, it measures repetition under a controlled target rather than uncertainty about a coined term. We conduct three comparisons: (1) verbatim callbacks against a stateless answerer without memory; (2) non-verbatim queries against stateless, to determine whether gains on callbacks come at the expense of ordinary turns; and (3) callback queries against their initial anchor queries, to determine whether the gain is non-trivial.

Table 1 presents results that support these claims. Relative to the stateless baseline, nearly all memory-based methods improve on both verbatim and non-verbatim queries. These results indicate that memory improves text-to-SQL performance in the repetitive setting and that this improvement does not arise from sacrificing performance on turns that do not depend on memory. The comparison between verbatim callbacks and their anchors also indicates a generally positive effect: models tend to answer a query more accurately when it is repeated. Although this setting primarily evaluates the general value of memory systems, our method consistently ranks among the strongest approaches.

Table 1: Scenario 1 execution accuracy (%). Small tags report differences against the comparison named in each column: stateless execution for verbatim and non-verbatim queries, and the shared initial anchor-query accuracy for callbacks.

| Method | Verbatim vs. Stateless | Non-verbatim vs. Stateless | Verbatim vs. Anchor |
|---|---|---|---|
| Reference | 80.0 | 44.1 | 77.3 |
| P-UTL | 88.0 +8.0 | 53.4 +9.3 | 88.0 +10.7 |
| H-LRU | 81.3 +1.3 | 53.7 +9.6 | 81.3 +4.0 |
| E-LLM | 82.7 +2.7 | 52.4 +8.3 | 82.7 +5.3 |
| E-Sum | 84.0 +4.0 | 50.5 +6.4 | 84.0 +6.7 |
| ACE | 81.3 +1.3 | 51.6 +7.5 | 81.3 +4.0 |
| MemAgent-7B | 80.0 +0.0 | 52.6 +8.5 | 80.0 +2.7 |
| MEM1 | 76.0 −4.0 | 54.3 +10.2 | 76.0 −1.3 |

Table 2: Execution accuracy on Scenario 2 convention callbacks at $B = 16$ across dormancy-gap distributions. All values are percentages.

| Distribution | P-UTL | H-LRU | E-LLM | E-Sum | ACE | MemAgent-7B | MEM1 |
|---|---|---|---|---|---|---|---|
| Exponential scale 4 | **98.3** | 91.7 | 96.7 | 63.3 | 90.0 | 25.0 | 36.7 |
| Exponential scale 8 | 90.0 | 83.3 | **96.7** | 76.7 | 88.3 | 20.0 | 25.0 |
| Exponential scale 16 | **96.7** | 70.0 | 95.0 | 73.3 | 86.7 | 23.3 | 28.3 |
| Normal mean 15 | 91.7 | 63.3 | **98.3** | 65.0 | 83.3 | 35.0 | 35.0 |
| Normal mean 25 | **98.3** | 40.0 | 90.0 | 71.7 | 83.3 | 26.7 | 33.3 |
| Normal mean 35 | 88.3 | 43.3 | **90.0** | 70.0 | 81.7 | 36.7 | 33.3 |
| Poisson $\lambda = 20$ | **100.0** | 53.3 | 90.0 | 71.7 | 78.3 | 25.0 | 15.0 |
| Poisson $\lambda = 30$ | **100.0** | 48.3 | 88.3 | 70.0 | 80.0 | 33.3 | 28.3 |
| Poisson $\lambda = 40$ | **100.0** | 41.4 | 89.7 | 74.1 | 60.3 | 36.2 | 29.3 |

### 4.3 Scenario 2: Memory across recurrence regimes

Scenario 2 addresses the absence of an empirical distribution describing when users return to earlier definitions. Rather than assuming a single interaction pattern, we use convention callbacks and sample dormancy from three controlled families representing different recurrence patterns. Exponential schedules place more callbacks near their definitions while retaining a long tail; normal schedules concentrate returns around a characteristic horizon; and Poisson schedules produce discrete gaps whose spread increases with the mean. Multiple parameters within each family span short to long dormancy while preserving the same executable dependency. We additionally vary the memory budget and analyze $B/d$, which measures capacity relative to the realized intervening horizon.

Table 2 reveals the performance of memory policies. H-LRU is a strong baseline when callbacks occur near their definitions, but its performance deteriorates substantially as the characteristic gap increases, demonstrating that recency is useful but insufficient over long horizons. The open-weight agents transfer poorly; one plausible explanation is that they are fine-tuned for other long-horizon tasks and that execution-grounded SQL retention is extrinsic to their training settings. Across all callbacks, P-UTL achieves 95.9% accuracy, compared with 92.8% for E-LLM, while every other baseline performs substantially worse than P-UTL. Figure 2 provides a complementary capacity-based view: at approximately $B/d = 0.25$, P-UTL maintains callback accuracy above 0.9. Predictive utility therefore remains highly competitive when memory can store only a small portion of the intervening horizon, with E-LLM representing the only close alternative in this regime. We additionally evaluate memory-pool sizes of $B = 8$ and $B = 32$; the results appear in Appendix C.1.

### 4.4 Scenario 3: Memory under capacity and density pressure

Scenario 3 converts the unitized memory stream into direct eviction pressure. We increase the number of initialized conventions until it approaches or exceeds the primary budget of $B = 16$, and contrast standard spacing with dense streams in which critical definitions arrive rapidly. We retain the longest-horizon Normal-35 and Poisson-40 regimes because E-LLM leads on the former in Scenario 2, whereas P-UTL leads on the latter. We compare these two policies because the easier study identifies E-LLM as the only competitive general baseline. The matched sparse and dense trajectories use identical callbacks. Appendix A.1 describes how density is controlled.

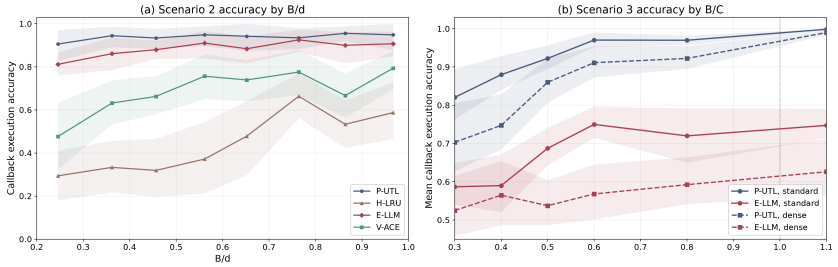


Figure 2: Convention-callback execution accuracy as a function of memory budget relative to (a) dormancy gap $d$ and (b) convention load $C$, pooled across $B \in \{8, 16, 32\}$. Shading gives trajectory-clustered confidence intervals at 95% confidence in both panels.

Table 3: Scenario 3 convention-callback execution accuracy at $B = 16$. We vary convention load $c$ and compare standard callback spacing with dense spacing; see Appendix A.1 for details. All values are percentages.

**(a) Normal, $\mu = 35$**

| $c$ | P-UTL | | E-LLM | |
|---|---|---|---|---|
| | *Standard* | *Dense* | *Standard* | *Dense* |
| 15 | **100.0** | **100.0** | 73.1 | 62.5 |
| 20 | **96.5** | **91.5** | 71.5 | 61.5 |
| 25 | **96.8** | **91.8** | 74.1 | 56.8 |
| 30 | **92.5** | **89.2** | 72.1 | 56.2 |

**(b) Poisson, $\lambda = 40$**

| $c$ | P-UTL | | E-LLM | |
|---|---|---|---|---|
| | *Standard* | *Dense* | *Standard* | *Dense* |
| 15 | **100.0** | **99.4** | 78.8 | 65.6 |
| 20 | **97.5** | **93.0** | 72.5 | 57.0 |
| 25 | **97.3** | **90.5** | 75.9 | 56.8 |
| 30 | **94.6** | **85.0** | 70.0 | 55.4 |

Table 3 presents the results of these experiments. On average, P-UTL achieves 94.2% accuracy, compared with 65.9% for E-LLM, and leads by a large margin in all 16 setups. Dense arrival reduces E-LLM accuracy by 10.0–19.1 percentage points across paired settings, whereas P-UTL changes less substantially. The $B/C$ panel of Figure 2 shows the same trend as capacity decreases to convention load. Additional results for $B = 8$ and $B = 32$ are provided in Appendix C.1.

### 4.5 Ablation studies

We present ablation studies on P-UTL training workloads and memory backbone capability below. Ablations of utility-score design choices and other explorations are provided in Appendix C.3.

**P-UTL training workloads.** The primary experiments train one P-UTL predictor for each workload configuration, which could make its advantage schedule-specific. In particular, the utility predictor might learn a particular distribution rather than generalizable user recurrence patterns. We test this possibility by comparing P-UTL with P-UTL-Full, a variant trained on all trajectories in the training splits of Scenarios 2 and 3. Evaluation databases remain held out, so success requires a single predictor to operate across workload mixtures rather than memorize test schemas.

Table 4 presents the results. They show small, bidirectional changes rather than systematic degradation: pooling causes a modest aggregate reduction in Scenario 2 but slightly improves the Scenario 3 micro-average. We conclude that, within the evaluated families, utility remains learnable when training expands from a single configuration to mixed recurrence, load, and density settings.

**Memory-manager backbone.** We use `GPT-4o-mini` as the memory-manager backbone for all API-based methods. One possible explanation for E-LLM's underperformance is that its manager lacks sufficient reasoning capacity or visibility into the interaction. To test whether a stronger LLM improves performance, we first replace its 4o-mini manager with `gpt-5.6-sol`, and then expose that stronger manager to the complete available history. The SQL answerer, trajectories, and memory budget remain fixed. The history is provided to the manager rather than directly to the SQL answerer.

Table 5 confirms that a stronger backbone and fuller manager context improve LLM-based eviction. A `GPT-5.6` memory manager performs substantially better than a `GPT-4o-mini` manager, with further overall improvement when the full conversational context is provided. These improvements,

Table 4: Generalization of P-UTL across trajectory configurations. Execution accuracy is reported in percentages. small tags report Full minus original, with green, red, and gray denoting positive, negative, and near-zero differences.

(a) Scenario 2. E, N, and P denote exponential, normal, and Poisson distributions, respectively.

| Method | E4 | E8 | E16 | N15 | N25 | N35 | P20 | P30 | P40 | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| P-UTL | 98.3 | 90.0 | 96.7 | 91.7 | 98.3 | 88.3 | 100.0 | 100.0 | 100.0 | 95.9 |
| P-UTL-FULL | 95.0 −3.3 | 95.0 +5.0 | 95.0 −1.7 | 93.3 +1.7 | 95.0 −3.3 | 95.0 +6.7 | 95.0 −5.0 | 95.0 −5.0 | 94.8 −5.2 | 94.8 −1.1 |

(b) Scenario 3 across capacity and definition-density configurations.

| Setup | Normal, $\mu = 35$ | | Poisson, $\lambda = 40$ | |
|---|---|---|---|---|
| | P-UTL | P-UTL-FULL | P-UTL | P-UTL-FULL |
| $c = 15$, *standard* | 100.0 | 100.0 +0.0 | 100.0 | 100.0 +0.0 |
| $c = 20$, *standard* | 96.5 | 94.5 −2.0 | 97.5 | 96.0 −1.5 |
| $c = 25$, *standard* | 96.8 | 95.5 −1.4 | 97.3 | 98.6 +1.4 |
| $c = 30$, *standard* | 92.5 | 92.9 +0.4 | 94.6 | 93.8 −0.8 |
| $c = 15$, *dense* | 100.0 | 100.0 +0.0 | 99.4 | 100.0 +0.6 |
| $c = 20$, *dense* | 91.5 | 94.5 +3.0 | 93.0 | 94.5 +1.5 |
| $c = 25$, *dense* | 91.8 | 90.0 −1.8 | 90.5 | 90.0 −0.5 |
| $c = 30$, *dense* | 89.2 | 88.8 −0.4 | 85.0 | 88.8 +3.8 |
| **Average** | 94.33 | 94.02 −0.30 | 94.15 | 94.76 +0.61 |

Table 5: Memory-manager backbone comparison on Normal $\mu = 35$ trajectories.

| Criterion | P-UTL | E-LLM 4o-mini | E-LLM GPT-5.6 | E-LLM-full GPT-5.6 |
|---|---|---|---|---|
| 15 callbacks | **99.0** | 65.7 | 91.4 | 90.5 |
| 20 callbacks | **95.0** | 60.0 | 91.4 | 90.7 |
| 25 callbacks | **93.1** | 62.3 | 86.3 | 90.3 |
| Average | **95.2** | 62.4 | 89.3 | 90.5 |
| Mean processing time | 2.07 ms | 0.77 s | 3.16 s | 5.82 s |
| Average cost per trajectory | $0.00 | $0.51 | $2.00 | $7.60 |

however, introduce two drawbacks. First, both API latency and cost increase substantially with a stronger model and additional context; the cost of processing one trajectory can increase by up to an order of magnitude relative to the original `GPT-4o-mini` version. Second, even with the stronger backbone, E-LLM does not outperform our method. Meanwhile, our method uses a lightweight local predictor, so its mean processing time and inference-time cost are comparatively negligible. These results demonstrate the value of a predictive layer in long-horizon text-to-SQL memory systems.

## 5 LIMITATIONS

Our generator is designed for controlled attribution rather than an exhaustive simulation of analyst intent. Real interactions may paraphrase, revise, compose, or reuse conventions unpredictably. The exponential, normal, and Poisson schedules therefore control recurrence timing and memory pressure; they are not distributions fitted to observed text-to-SQL users. We encourage future work to develop more diverse simulations of user behavior and to extend evaluation to broader collections of schemas. In addition, although effective, the utility predictor requires training on trajectory splits, therefore trade-offs between training and api-based LLM-calls still exist on low-resource scenarios.

## 6 CONCLUSION

Text-to-SQL produces structured memory units, recurring schema-grounded concepts, and executable consequences, making it a natural domain for learned memory utility. We formulate bounded, long-horizon text-to-SQL memory management as online eviction, introduce P-UTL as an execution-supervised utility policy, and provide a controlled generator for delayed, dense, and capacity-constrained recurrence. Across simulations based on different distributions, P-UTL outperforms the baselines and is particularly robust when useful information competes for limited capacity, while maintaining low online decision overhead. Our results demonstrate how executable agent domains can support memory policies optimized for downstream consequences and provide reproducible tools for studying what long-running agents should remember.

## AI USE STATEMENT

In this work, we used generative AI tools only to revise author-written manuscripts for clarity, grammar, and academic formality. All AI-assisted revisions were reviewed and approved by the authors. We take responsibility for the final content of this work, including text, claims, tables, and other artifacts produced with the aid of generative AI.

## REFERENCES

Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: the search for invariants. *SIGMETRICS Perform. Eval. Rev.*, 24(1):126–137, May 1996. ISSN 0163-5999. doi: 10.1145/233008.233034. URL https://doi.org/10.1145/233008.233034.

Asim Biswal, Chuan Lei, Xiao Qin, Aodong Li, Balakrishnan Narayanaswamy, and Tim Kraska. Agentsm: Semantic memory for agentic text-to-sql, 2026. URL https://arxiv.org/abs/2601.15709.

L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 1, pp. 126–134 vol.1, 1999. doi: 10.1109/INFCOM.1999.749260.

M.E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997. doi: 10.1109/90.650143.

Peter J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, May 1968. ISSN 0001-0782. doi: 10.1145/363095.363141. URL https://doi.org/10.1145/363095.363141.

Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek Narasayya. Dsb: a decision support benchmark for workload-driven and traditional database systems. *Proc. VLDB Endow.*, 14(13): 3376–3388, September 2021. ISSN 2150-8097. doi: 10.14778/3484224.3484234. URL https://doi.org/10.14778/3484224.3484234.

Taicheng Guo, Hai Wang, ChaoChun Liu, Mohsen Golalikhani, Xin Chen, Xiangliang Zhang, and Chandan K. Reddy. Mtsql-r1: Towards long-horizon multi-turn text-to-sql via agentic training, 2026. URL https://arxiv.org/abs/2510.12831.

Yongqi Han, Qingfeng Du, Jincheng Xu, Shengjie Zhao, Zhekang Chen, Li Cao, Kanglin Yin, and Dan Pei. Lws: A framework for log-based workload simulation in session-based sut, 2023. URL https://arxiv.org/abs/2301.08851.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model, 2024. URL https://arxiv.org/abs/2408.09559.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions, 2026. URL https://arxiv.org/abs/2507.05257.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. Bird-interact: Re-imagining text-to-sql evaluation for large language models via lens of dynamic interactions, 2026. URL https://arxiv.org/abs/2510.05318.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows, 2025. URL https://arxiv.org/abs/2411.07763.

W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994. doi: 10.1109/90.282603.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls, 2023. URL https://arxiv.org/abs/2305.03111.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents, 2024. URL https://arxiv.org/abs/2402.17753.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. 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, 2023. URL https://arxiv.org/abs/2304.03442.

Ravi Kumar Tummalapenta and Suman Addanki. Memory architectures for multi-turn text-to-sql: A benchmark and empirical study, 2026. URL https://arxiv.org/abs/2605.26394.

Christian Vögele, André Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. Wessbas: extraction of probabilistic workload specifications for load testing and performance prediction–a model-driven approach for session-based application systems. *Softw. Syst. Model.*, 17(2): 443–477, May 2018. ISSN 1619-1366. doi: 10.1007/s10270-016-0566-5. URL https://doi.org/10.1007/s10270-016-0566-5.

Ronald W. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1982. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/170165.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Ming Yin, Danmei Xu, Zhewei Yao, Yuxiong He, Harshn Pathak, and Asim Shankar. ArcticMem: Persistent memory behind Snowflake CoCo. Snowflake Engineering Blog, July 2026. URL https://www.snowflake.com/en/blog/engineering/arcticmem-persistent-memory-ai-agents/. Published July 29, 2026. Accessed September 20, 2026.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2026. URL https://arxiv.org/abs/2507.02259.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong,

Richard Socher, Walter Lasecki, and Dragomir Radev. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1962–1979, Hong Kong, China, November 2019a. Association for Computational Linguistics. doi: 10.18653/v1/D19-1204. URL https://aclanthology.org/D19-1204/.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. SParC: Cross-domain semantic parsing in context. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4511–4523, Florence, Italy, July 2019b. Association for Computational Linguistics. doi: 10.18653/v1/P19-1443. URL https://aclanthology.org/P19-1443/.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026. URL https://arxiv.org/abs/2510.04618.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. Ama-bench: Evaluating long-horizon memory for agentic applications, 2026. URL https://arxiv.org/abs/2602.22769.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

Jiawei Zhou, Jianwei Wang, Chenyu Zhou, Chaojian Shi, Ming Dong, and Kai Wang. Evosql: Memory-augmented critic-generator co-evolution for text-to-sql, 2026. URL https://arxiv.org/abs/2607.20489.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

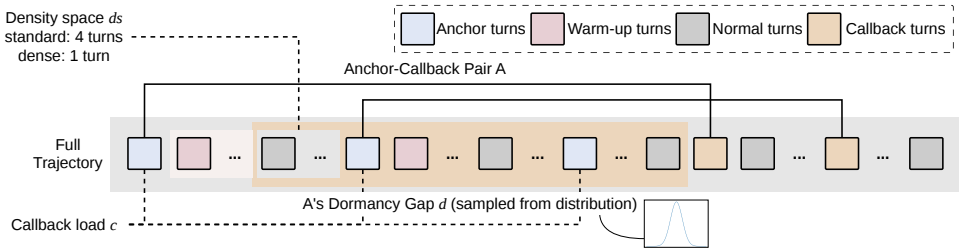


Figure 3: Controls used to construct a long-horizon trajectory. Initialization–callback units are interleaved with ordinary turns, while dormancy $d$, convention load $C$, and definition spacing $s$ control delayed reuse and memory pressure. The logged generator names $c$ and `ds` shown in the figure correspond to $C$ and $s$, respectively.

# A Additional Methodology

In this section, we describe additional definitions and provide illustrations for all concepts and terminologies we used to form our trajectory.

## A.1 Trajectories

**Trajectory construction.** We convert each SParC and CoSQL interaction into an ordered sequence of natural-language requests paired with executable reference SQL. Interactions against the same Spider database are then assigned to a persistent analyst and stitched into one stream. We preserve the order and adjacency of turns inside each source interaction; moving to another interaction produces ordinary background activity on the same database. Queries whose reference programs do not execute are excluded during construction. Each emitted round records its source identifier, database snapshot, gold program, execution result, and callback metadata.

**Verbatim callbacks.** Verbatim callbacks are the anchor-callback pairs that are identical to each other. A verbatim callback selects the first request of a source interaction, require at least four tokens, and reject anchors whose text is duplicated in a local window. The callback embeds the anchor request unchanged in a fixed wrapper and inherits its reference SQL and canonical execution result. The following is an actual unit from the released `activity_1::analyst_0` trajectory.

Listing 1: A frozen verbatim initialization–callback pair.

```
[turn 6: anchor]
USER: How many professors are there?
REFERENCE SQL:
SELECT COUNT ( * ) FROM Faculty WHERE Rank = "Professor"

[memory written after the corrected round]
Q: How many professors are there?
Accepted SQL: SELECT COUNT(*) FROM Faculty WHERE Rank LIKE 'professor'
Head result: ((27,),)

[turn 22: callback; recorded dormancy d = 16]
USER: Earlier in our conversation we asked: "How many professors are there?". Please recompute
      that same query now and return the result.
REFERENCE SQL:
SELECT COUNT ( * ) FROM Faculty WHERE Rank = "Professor"
```

The wrapper makes the requested repetition explicit without supplying the SQL. Calling the unit *verbatim* refers to the preserved anchor request and identical executable target; it does not mean that the surrounding recall instruction is absent. The candidate memory becomes available only after turn 6 is completed and can therefore help turn 22 but not its own source round.

**Convention callbacks.** Convention callback are anchor-callback pairs that ask queries around some custom concepts and definitions related to the schema. The concepts are private so the model

is not able to answer the question if it does not preserve related memories. A convention unit is synthesized from an eligible numeric database column. The generator computes a readable threshold between the empirical mean and maximum, checks that the resulting predicate selects a nonempty proper subset, and gives it a user-facing name. A definition round and three adjacent warm-ups establish the convention using count, maximum, average, and distinct-count operations. The callback changes the requested aggregation to a sum and mentions only the coined term. Listing 2 reproduces the requests and reference programs from the frozen `activity_1::analyst_0` trajectory.

Listing 2: A frozen convention unit; intervening background rounds are omitted.

```
[turn 11: definition]
USER: For the rest of our conversation, a 'large phone' is a row in Faculty where Phone >
    6000. Use this definition. To start: how many large phone entries are there?
REFERENCE SQL: SELECT COUNT(*) FROM "Faculty" WHERE "Phone" > 6000
[turn 12: warm-up]
USER: What's the maximum Phone we've seen among the large phone?
REFERENCE SQL: SELECT MAX("Phone") FROM "Faculty" WHERE "Phone" > 6000
[turn 13: warm-up]
USER: What's the average Phone among the large phone?
REFERENCE SQL: SELECT AVG("Phone") FROM "Faculty" WHERE "Phone" > 6000
[turn 14: warm-up]
USER: How many distinct values of Phone appear in the large phone?
REFERENCE SQL: SELECT COUNT(DISTINCT "Phone") FROM "Faculty" WHERE "Phone" > 6000

... SParC/CoSQL activity and other convention blocks ...

[turn 45: callback; recorded dormancy d = 22]
USER: Going back to our earlier definition: what is the SUM of Phone across all large phone?
REFERENCE SQL: SELECT SUM("Phone") FROM "Faculty" WHERE "Phone" > 6000
```

The callback does not expose the threshold 6000, and its target operation does not duplicate any earlier operation. A system must therefore preserve and apply the latent predicate rather than retrieve a matching SQL string. We execute every generated reference query during construction and retain the unit only when its definition, warm-ups, and callback are valid. Metadata marks definition, warm-up, and callback rounds with a shared convention identifier, the realized dormancy, and the final location, enabling callback-only scoring and paired analysis. These fields are used by the generator and evaluator only; P-UTL receives none of the convention identifier, role, callback flag, or scheduled dormancy.

**Dormancy.** Dormancy controls the distance between the anchor-callback pairs. Synthetic recall units are inserted only at interaction boundaries. Figure 3 illustrates the anchor–callback interval over which dormancy is measured. For a verbatim unit, the recorded dormancy $d$ is the elapsed background-stream turn distance between the selected anchor and its callback. For a convention unit, $d$ is scheduled from the final warm-up on the source/background coordinate. All convention units are planned before they are jointly spliced into the rendered trajectory; consequently, blocks belonging to other conventions can occur inside the displayed interval.

**Definition density.** Definition density controls the density of anchor generation. The higher density, the higher expected number of anchors and critical concepts under each anchor-callback pair, and thus the higher challenge posed to the memory management. Definition density is controlled independently of convention load and dormancy. The left side of Figure 3 illustrates convention load and the standard-versus-dense spacing intervention. Let $s$ denote the spacing between definition blocks, measured in source-interaction insertion slots. For convention $i \in \{0, \ldots, C-1\}$, the generator schedules its four-round block (one definition followed by three warm-ups) after source-interaction slot $q_i = \min\{(i+1)s, |\mathcal{I}| - 1\}$, where $\mathcal{I}$ is the ordered background stream. Standard and dense settings use different values of $s$ (Appendix B). The value of $s$ is not a distance in the final rendered trajectory: each definition expands into four rounds, and callbacks and other definition blocks are subsequently interleaved.

For each convention, the generator samples dormancy $d$ from the configured distribution and places the callback at the earliest unoccupied source slot whose end reaches the target distance from the convention's final warm-up. Consequently, decreasing $s$ increases the number of active definitions that overlap during the same dormancy interval without changing the convention load $C$, memory budget $B$, or recurrence family.

Table 6: Input features of the reported P-UTL utility predictor.

| Block | Dimension | Source at write time |
|---|---|---|
| Memory content | 768 | `all-mpnet-base-v2` embedding |
| Memory type | 4 | Episodic/corrective writer category |
| Schema reference | $2(\|\Sigma_0\| + 64)$ | Mentioned and available schema names |
| Trajectory state | 4 | Age, elapsed time, boundary, recent similarity |
| Recent queries | 768 | Mean embedding of the preceding 10 requests |
| Usage state | 4 | Retrieval/correctness/recency/utility history |

### A.2 Utility Scores

**Counterfactual utility construction.** Labels are generated by replaying every completed memory write into an unbounded store; no learned or bounded eviction policy removes entries during this offline pass. For each memory and each future question inside horizon $H$, an ablation is issued only if the unbounded-store retriever originally placed that memory among its top $K = 5$ results. The observed first SQL attempt is $y^{+m}$. To obtain $y^{-m}$, we remove $m$ and leave the other $K-1$ retrieved memories unchanged, so the next-ranked candidate does not fill the vacant position. A memory that is never retrieved within the horizon therefore has observed utility zero.

This procedure estimates marginal prompt utility under a fixed observed retrieval state. It is not a replay of the pool that would result had $m$ been evicted earlier, and it does not estimate interactions among jointly retrieved memories. The accumulated integer utility is converted to its harmful, neutral, or helpful sign for the reported classifier.

**Utility predictor.** The reported predictor uses only the write-time snapshot of each memory. Its feature blocks are listed in Table 6. All are derived from the ordinary transcript, schema, and memory writer. Synthetic convention identifiers, initialization/warm-up/callback roles, scheduled dormancy, callback locations, future queries, and future labels are excluded.

At a write snapshot, age and elapsed time are zero and usage history is empty; these blocks are retained for a common feature interface but are not updated by the reported static policy. At each overflow, the static predictor scores the candidate memories using Equation 9 and retains the top $B$ entries. Architecture and training settings are given in Appendix B.

### A.3 Scope of the Generator

The generator is intended to isolate delayed memory consequences, not to reproduce the complete distribution of analyst intentions. Verbatim units use an explicit repetition wrapper, while the present convention units use named numeric predicates, three adjacent warm-ups, and one principal later callback. This regularity makes the executable dependency and its dormancy observable, which is useful for controlled comparison, but excludes ambiguous introductions, paraphrased or repeated callbacks, categorical conventions, join-based business rules, and definitions that are later revised or superseded. Likewise, the probability families control recurrence timing and tail behavior rather than fitting an empirical population of text-to-SQL users. All conclusions in the main paper are scoped to these controlled SParC- and CoSQL-derived regimes.

## B Additional Experimental Setups

This section details additional information on constructing the trajectory and building the baselines.

### B.1 Datasets and Workloads

**Datasets and evaluation cohorts.** Within each generated configuration, the database split is fixed before training: 10 databases are used for training, four for validation, and six for testing, yielding 15 held-out test trajectories. Scenario 1's ordinary-query evaluation uses a deterministic five-trajectory subset (626 queries), whereas its verbatim evaluation contains 75 callbacks per trial on the frozen 15-trajectory cohort. The verbatim stateless reference averages two trials; its difference from P-UTL is therefore reported descriptively rather than with a single-trial paired interval. Scenario 2

Table 7: Trajectory-generator configurations. Exponential labels E4, E8, and E16 denote exponential scales 4, 8, and 16 (rates $1/4$, $1/8$, and $1/16$). The normal standard deviation is 7.5.

| Scenario | Recall unit | Gap family | $C$ | Spacing / range |
|---|---|---|---|---|
| 1 | Verbatim | Uniform | 5 | $d \in [15, 50]$ |
| 2 | Convention | E4/8/16; N15/25/35; P20/30/40 | 5 | $s = 4$, $d \in [0, 50]$ |
| 3 | Convention | N35; P40 | 15/20/25/30 | $s \in \{4, 1\}$, $d \in [5, 65]$ |

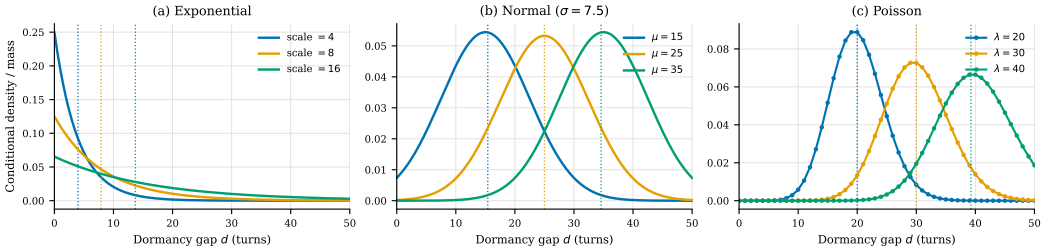


Figure 4: Dormancy distributions used in Scenario 2 after rejection sampling to $d \in [0, 50]$. The panels show (a) exponential scales 4, 8, and 16; (b) normal means 15, 25, and 35 with standard deviation 7.5; and (c) Poisson rates 20, 30, and 40. Dotted lines mark the conditional means. Scenario 3 reuses Normal-35 and Poisson-40 with $d \in [5, 65]$.

contains 60 callbacks per distribution except Poisson-40, which contains 58, for 538 callbacks overall. Scenario 3 contains 160–240 callbacks per cell and 3,280 per method overall. Table 7 and 8 contain full details of data scale of our work.

**Workload configurations.** Table 7 summarizes the workload parameters. Its count column denotes verbatim recall units in Scenario 1 and initialized conventions in Scenarios 2–3. Normal and exponential draws are rounded to integers, Poisson draws use a discrete sampler, and all draws are rejection-sampled to the configured range. The exponential scales 4, 8, and 16 correspond to rates $1/4$, $1/8$, and $1/16$. Figure 4 visualizes the three distribution families and all Scenario 2 parameter settings. Scenario 3 lengthens the source stream to accommodate the larger convention loads. Every method within a cell replays the same frozen trajectories. Standard and dense definition arrival use $s = 4$ and $s = 1$, respectively. For each Scenario 3 family and convention load, these conditions share the seed, source stream, convention identities, callback questions and SQL targets, dormancy distribution, and evaluation cohort; only spacing changes.

**Correction protocol.** In our workload we provide corrections to incorrect text-to-SQL answers. Corrections are generated by our execution-grounded simulator rather than inherited from SParC or CoSQL. When an attempted SQL program does not match the canonical result of the reference program, the simulator compares the two programs, produces terse error-specific natural-language feedback, validates the corresponding correction by execution, and supplies the failed attempt and feedback to the answerer for its next attempt, with at most three attempts per round. If the proposed correction cannot be validated, the simulator supplies the reference SQL as a fallback; rounds answered correctly on the first attempt receive no correction.

The correction protocol only applies to the non-callback turns for memory generation. Callback rounds do not involve corrections. The accuracy metrics for those rounds are computed by the one-shot correctness of the text-to-SQL query.

**Trajectory scale.** Table 8 reports mean trajectory length, user-query tokens, and total tokens introduced per trajectory, together with estimated cumulative context reading. The reading columns sum tokens across all turns rather than describing the context length of one call; the accounting assumptions are specified below.

Table 8: Mean per-trajectory scale and estimated context-reading cost. Full-context and memory-reading values are cumulative token counts across all turns. Each number is a per-trajectory average.

| Scenario | Mean turns | Mean query tokens | Mean all tokens | Full-context reading | 16-slot memory reading |
|---|---|---|---|---|---|
| Scenario 1 | 193.9 | 2,542 | 22,454 | 2.19M | 192k |
| Scenario 2 | 211.7 | 2,817 | 23,479 | 2.50M | 202k |
| Scenario 3 | 654.7 | 8,863 | 68,476 | 22.45M | 616k |

### B.2 BASELINES

**Shared evaluation protocol.** Unless an ablation explicitly changes the manager, the SQL answerer and all API-based memory managers use the pinned `gpt-4o-mini-2024-07-18` snapshot. Trajectory SQL calls use temperature 0.2 and best-effort seed 20260901; manager calls use temperature 0 and seed 20260902. Offline counterfactual SQL calls use temperature 0. Because API seeding is best-effort, the recorded model snapshots and system fingerprints are retained with the run telemetry. Fixed-slot methods retrieve the five most similar memories. The memory produced from round $t$ is written only after that round is scored, preventing access to the answer being evaluated. Retrieval-result shortcuts are disabled.

**Scoring and uncertainty.** We compare predicted and reference programs by executing both on the same Spider database snapshot and canonicalizing their results. Row order is preserved when the reference contains `ORDER BY` and ignored otherwise. Scenario 1 reports both callback and ordinary-gap queries; Scenarios 2 and 3 score only convention callbacks. We report 95% percentile intervals from 20,000 bootstrap resamples of whole trajectories, retaining correlated callbacks from the same analyst in each cluster. The frozen trajectory manifest and callback fingerprints are checked before aggregation.

**Predictor training.** The experimental model is a two-hidden-layer MLP with widths 128 and 32, ReLU activations, and dropout 0.3. Feature blocks are normalized separately. Counterfactual labels follow Appendix A.2, using $H = 100$ in Scenarios 1–2 and $H = 200$ in Scenario 3.

We train for at most 50 epochs with batch size 32, class-weighted cross-entropy, AdamW (learning rate $10^{-3}$ and weight decay $10^{-2}$), and gradient-norm clipping at 1.0. Training stops after eight validation epochs without an improvement in macro-F1 and retains the best checkpoint. Runs are deterministic with seed `20260618`. Training used an NVIDIA A6000; online retention requires only independent MLP scores and a top-$B$ selection.

**Memory budgets.** The primary fixed-slot budget is $B = 16$; auxiliary sweeps use $B \in \{8, 32\}$. E-Sum uses a calibrated 857-token budget at the primary setting and 429 or 1,714 tokens in the corresponding smaller and larger settings. These token budgets are not slot constraints. MemAgent-7B and MEM1 are omitted from capacity sweeps because their recurrent states do not implement fixed-cardinality pools.

**Heuristic baseline.** H-LRU evicts the memory retrieved least recently and also supplies the fallback selections for API-based fixed-slot managers.

**E-LLM and E-Sum.** At every overflow, E-LLM receives the numbered candidate pool, with each memory's type and first 400 content characters, and returns exactly $B$ one-based indices as JSON. Invalid, duplicate, or missing selections are removed and H-LRU fills any remaining positions. E-LLM-full uses the same selection interface but additionally receives up to 400 preceding user turns; the SQL answerer still receives only the memories selected by the manager. E-Sum instead appends new memory text to one state and, when it exceeds its token budget, prompts the manager to preserve distinct definitions, thresholds, schema references, and question–SQL patterns. An over-budget or failed compression is deterministically trimmed to the configured budget.

**ACE adaptation.** Our ACE-inspired curator uses the fixed-slot admission/eviction interface. At overflow, it receives numbered candidates containing each memory's type and first 400 content characters and returns exactly $B$ indices (Listing 3).

The response budget is 300 tokens. We parse the response as JSON, falling back to the first embedded array. Invalid and duplicate indices are discarded, and H-LRU fills missing slots, including after API failure.

**MemAgent-7B.** We use the released `BytedTsinghua-SIA/RL-MemoryAgent-7B` checkpoint and adapt its section-reading interface to database conversations. The problem is fixed across updates; the previous textual state and new turn use the native tags shown in Listing 4.

**MEM1.** We use `Mem-Lab/Qwen2.5-7B-RL-RAG-Q2-EM-Release`. MEM1 updates a running consolidated state, from which we extract the text enclosed by `<think>` tags (Listing 5).

For both models, the current-turn string contains the user request, the first SQL attempt, and convention-definition metadata when present. It is truncated to 2,000 characters; the previous state is truncated to 1,500 characters. Generation is greedy with a 512-token maximum, using bfloat16 on CUDA and each checkpoint's chat template. An empty generation preserves the previous state; MEM1 falls back to the raw generation when no complete `<think>` span is present.

At an assessed round, the current consolidated state is passed to the shared SQL answerer as one retrieved-memory block, and the memory model is updated only after SQL execution is scored. State is reset at the beginning of every trajectory. For these textual-state models, the reported $B = 16$ is a nominal run label; they are excluded from the $B/d$, $B/C$, and capacity comparisons.

**Computational accounting.** Context-reading estimates sum the tokens read across all turns of a trajectory. At turn $n$, the full-context estimate includes all tokens introduced from turns 1 through $n$; the bounded-memory estimate assumes a store of at most $B = 16$ slots. This is an estimate of cumulative reading, not the context length of an individual call or measured API usage. The online protocol separately retrieves $K = 5$ entries for SQL answering.

The latency measurements in Table 5 cover one online memory decision: an MLP top-$B$ update or a manager API call. SQL answering is excluded. Online manager API costs are reported per trajectory; offline labeling and predictor training are accounted for separately where telemetry is available.

**Prompt templates.** Listings 3–5 reproduce the manager templates. Braced fields are substituted with the corresponding memory or turn text.

Listing 3: ACE curator adapter prompt.

```
SYSTEM:
You are a knowledge base curator for a text-to-SQL agent. Your task: select which memories to
     keep so the agent has the most useful knowledge for FUTURE sessions. Respond ONLY with a
     JSON array of the 1-based indices you choose to KEEP; do not include any other text.

USER:
Memories currently held: {n}

1. [{type_1}] {memory_1}
...
n. [{type_n}] {memory_n}

Keep exactly {B} of these. Respond with a JSON array of the 1-based indices to KEEP.
```

Listing 4: MemAgent-7B update prompt.

```
SYSTEM:
You will be shown a problem, a previous memory, and a new section of a database conversation.
     Update the memory to retain information that helps answer the problem, while preserving
     all relevant details from the previous memory.

USER:
<problem> Track SQL conventions, business terminology definitions, and factual patterns from
     this database conversation to help answer future SQL questions. </problem>
<memory> {previous_memory} </memory>
<section> {current_turn} </section>
Updated memory:
```

Table 9: Convention-callback execution accuracy for additional episodic-preserving heuristic controls at $B = 16$. All values are percentages.

(a) Normal-35, standard

| $C$ | P-UTL | E-LLM | EPI-FIFO | EPI-Random |
|---|---|---|---|---|
| 15 | **100.0** | 73.1 | 21.3 | 50.0 |
| 20 | **96.5** | 71.5 | 27.0 | 55.0 |
| 25 | **96.8** | 74.1 | 22.0 | 52.4 |

(b) Normal-35, dense

| $C$ | P-UTL | E-LLM | EPI-FIFO | EPI-Random |
|---|---|---|---|---|
| 15 | **100.0** | 62.5 | 24.0 | 36.7 |
| 20 | **91.5** | 61.5 | 24.0 | 32.5 |
| 25 | **91.8** | 56.8 | 26.4 | 34.0 |

(c) Poisson-40, standard

| $C$ | P-UTL | E-LLM | EPI-FIFO | EPI-Random |
|---|---|---|---|---|
| 15 | **100.0** | 78.8 | 25.0 | 57.5 |
| 20 | **97.5** | 72.5 | 29.0 | 49.5 |
| 25 | **97.3** | 75.9 | 31.8 | 57.7 |

(d) Poisson-40, dense

| $C$ | P-UTL | E-LLM | EPI-FIFO | EPI-Random |
|---|---|---|---|---|
| 15 | **99.4** | 65.6 | 23.3 | 36.7 |
| 20 | **93.0** | 57.0 | 19.5 | 29.5 |
| 25 | **90.5** | 56.8 | 28.0 | 35.0 |

Table 10: Scenario 2 convention-callback execution accuracy at $B = 8$. E-Sum uses a 429-token memory budget. All values are percentages.

| Distribution | P-UTL | H-LRU | E-LLM | E-Sum | ACE |
|---|---|---|---|---|---|
| Exponential scale 4 | **91.7** | 73.3 | **91.7** | 48.3 | 76.7 |
| Exponential scale 8 | 85.0 | 50.0 | **93.3** | 48.3 | 68.3 |
| Exponential scale 16 | **95.0** | 48.3 | 88.3 | 53.3 | 63.3 |
| Normal mean 15 | 88.3 | 33.3 | **93.3** | 58.3 | 58.3 |
| Normal mean 25 | **86.7** | 26.7 | 81.7 | 38.3 | 63.3 |
| Normal mean 35 | **80.0** | 30.0 | 76.7 | 45.0 | 45.0 |
| Poisson $\lambda = 20$ | **95.0** | 28.3 | 85.0 | 40.0 | 53.3 |
| Poisson $\lambda = 30$ | **95.0** | 25.0 | 78.3 | 50.0 | 43.3 |
| Poisson $\lambda = 40$ | **100.0** | 34.5 | 81.0 | 41.4 | 56.9 |
| **Micro-average** | **90.7** | 38.8 | 85.5 | 47.0 | 58.7 |

Listing 5: MEM1 consolidation prompt.

```
SYSTEM:
You are a memory consolidation assistant for a text-to-SQL agent. At each step you will see
     the current conversation turn and a previous memory summary. Update the summary to retain
      SQL conventions, business rules, and factual patterns that will help answer future SQL
     questions. Output the updated summary inside <think>...</think> tags.

USER:
Previous memory summary:
{previous_memory}

New conversation turn:
{current_turn}

Update the memory summary. Output inside <think>...</think> tags.
```

# C Additional Results

This section reports additional experimental results, ablation studies and discoveries we have made.

## C.1 Additional Experimental Results

**Additional heuristic methods.** We are interested in how the text-to-SQL performs with other heuristic methods. Empirically, we observe that all convention anchor turns are tagged as episodic by the memory tagger. Therefore, beyond H-LRU, we evaluate two episodic-preserving policies to check whether we can preserve convensions by only heuristic methods. EPI-FIFO gives episodic memories priority over other memory types and, when episodic entries exceed capacity, evicts them in insertion order; any remaining slots are filled using LRU over non-episodic memories. EPI-Random applies the same type priority but selects among episodic memories using a deterministic pool-hashed random ordering. These controls test whether prioritizing all episodic memories is sufficient without estimating their downstream utility.

Table 11: Scenario 2 convention-callback execution accuracy at $B = 32$. E-Sum uses a 1,714-token memory budget. All values are percentages.

| Distribution | P-UTL | H-LRU | E-LLM | E-Sum | ACE |
|---|---|---|---|---|---|
| Exponential scale 4 | **100.0** | 96.7 | 96.7 | 91.7 | 86.7 |
| Exponential scale 8 | **96.7** | **96.7** | 91.7 | 78.3 | 88.3 |
| Exponential scale 16 | **98.3** | 88.3 | 95.0 | 86.7 | 81.7 |
| Normal mean 15 | **100.0** | 90.0 | 93.3 | 86.7 | 85.0 |
| Normal mean 25 | **100.0** | 85.0 | 95.0 | 75.0 | 81.7 |
| Normal mean 35 | **98.3** | 78.3 | 91.7 | 75.0 | 75.0 |
| Poisson $\lambda = 20$ | **100.0** | 93.3 | 98.3 | 78.3 | 90.0 |
| Poisson $\lambda = 30$ | **98.3** | 73.3 | 90.0 | 78.3 | 83.3 |
| Poisson $\lambda = 40$ | **100.0** | 69.0 | 91.4 | 82.8 | 70.7 |
| **Micro-average** | **99.1** | 85.7 | 93.7 | 81.4 | 82.5 |

Table 12: Scenario 3 convention-callback execution accuracy at $B = 8$. We vary convention load $c$ and compare standard ($ds = 4$) with dense ($ds = 1$) definition arrival. All values are percentages.

**(a) Normal, $\mu = 35$**

| $c$ | P-UTL | | E-LLM | |
|---|---|---|---|---|
| | $ds = 4$ | $ds = 1$ | $ds = 4$ | $ds = 1$ |
| 15 | **90.6** | **86.2** | 63.7 | 54.4 |
| 20 | **87.5** | **71.5** | 56.0 | 59.5 |
| 25 | **85.0** | **70.0** | 53.2 | 53.6 |
| 30 | **80.8** | **69.2** | 55.4 | 47.5 |

**(b) Poisson, $\lambda = 40$**

| $c$ | P-UTL | | E-LLM | |
|---|---|---|---|---|
| | $ds = 4$ | $ds = 1$ | $ds = 4$ | $ds = 1$ |
| 15 | **90.0** | **82.5** | 66.9 | 46.9 |
| 20 | **88.5** | **78.0** | 62.0 | 53.5 |
| 25 | **81.8** | **72.7** | 65.0 | 53.2 |
| 30 | **80.8** | **69.6** | 61.3 | 55.8 |

As shown by Table 9, EPI-Random consistently improves on EPI-FIFO, showing that insertion order is particularly poorly aligned with delayed callback utility, especially in complex trajectories. The conclusion is similar to what we drawed from H-LRU. Nevertheless, both episodic-preserving heuristics remain substantially below E-LLM and P-UTL across every displayed configuration. The result indicates that an episodic label alone is too coarse: successful bounded retention requires distinguishing which episodic entries are likely to matter later, rather than preserving the category indiscriminately.

**Results on different memory pool size.** We report auxiliary budgets $B = 8$ and $B = 32$ on the same frozen trajectories as the primary $B = 16$ experiments. Textual-state budgets and exclusions follow Appendix B. In the capacity tables, boldface denotes the highest observed accuracy within each matched condition, including ties.

Tables 10 and 11 report the $B = 8$ and $B = 32$ results, respectively, on **scenario 2**. The exponential values 4, 8, and 16 denote scales, as specified in Table 7. Increasing capacity benefits every fixed-slot policy in aggregate, but it does not erase the differences among retention rules. At $B = 8$, where admission is most selective, P-UTL retains a micro-average advantage over E-LLM and a much larger advantage over H-LRU, E-Sum, and ACE. At $B = 32$ most methods improve substantially, yet P-UTL remains the strongest aggregate policy. Together with the primary $B = 16$ table, these results show that the conclusion is not tied to a single chosen capacity.

Tables 12 and 13 report the $B = 8$ and $B = 32$ results on **scenario 3**, with separate panels for Normal-35 and Poisson-40. Their notation $c$ and $ds$ corresponds to convention load $C$ and definition spacing $s$; $ds = 4$ is standard arrival and $ds = 1$ is dense arrival. The harder capacity-and-density regime preserves the ordering observed at $B = 16$. P-UTL leads E-LLM in every displayed $B = 8$ and $B = 32$ cell, including dense streams and convention loads that meet or exceed the primary budget. Capacity changes the absolute difficulty, but additional slots alone do not substitute for predicting which definitions will have downstream execution value. The occasional non-monotonic change in an individual cell is consistent with stochastic SQL generation and retrieval interactions; our claim concerns the repeated cross-condition advantage rather than strict monotonicity for every trajectory.

### C.2 Ablations on Non-Verbatim/Convention Queries

The convention-callback experiments test whether a memory policy preserves information that later becomes execution-critical. However, a policy could improve those callbacks by retaining conven-

Table 13: Scenario 3 convention-callback execution accuracy at $B = 32$. We vary convention load $c$ and compare standard ($ds = 4$) with dense ($ds = 1$) definition arrival. All values are percentages.

**(a) Normal, $\mu = 35$**

| $c$ | P-UTL | | E-LLM | |
|---|---|---|---|---|
| | $ds = 4$ | $ds = 1$ | $ds = 4$ | $ds = 1$ |
| 15 | **100.0** | **100.0** | 81.9 | 65.0 |
| 20 | **100.0** | **100.0** | 79.5 | 68.0 |
| 25 | **100.0** | **100.0** | 76.8 | 61.8 |
| 30 | **99.6** | **97.9** | 75.8 | 62.1 |

**(b) Poisson, $\lambda = 40$**

| $c$ | P-UTL | | E-LLM | |
|---|---|---|---|---|
| | $ds = 4$ | $ds = 1$ | $ds = 4$ | $ds = 1$ |
| 15 | **95.0** | **80.0** | 80.6 | 72.5 |
| 20 | **91.0** | **80.5** | 80.5 | 71.5 |
| 25 | **100.0** | **100.0** | 73.2 | 62.7 |
| 30 | **100.0** | **99.2** | 72.1 | 61.3 |

Table 14: Scenario 2 execution accuracy on ordinary dormancy-gap queries. Accuracy values are percentages. Small green tags beside method scores report percentage-point improvements over the stateless reference; the final column reports P-UTL minus E-LLM in percentage points.

| Distribution | Stateless | P-UTL | E-LLM | P-UTL − E-LLM |
|---|---|---|---|---|
| Exponential-4 | 43.36 | 48.51 +5.15 | 48.51 +5.15 | +0.00 |
| Exponential-8 | 43.23 | 48.42 +5.19 | 51.02 +7.79 | −2.60 |
| Exponential-16 | 44.63 | 52.38 +7.76 | 50.88 +6.26 | +1.50 |
| Normal-15 | 43.93 | 52.49 +8.56 | 51.72 +7.79 | +0.77 |
| Normal-25 | 44.87 | 52.30 +7.43 | 51.67 +6.80 | +0.63 |
| Normal-35 | 45.62 | 51.96 +6.34 | 52.99 +7.37 | −1.03 |
| Poisson-20 | 45.78 | 54.11 +8.32 | 52.87 +7.09 | +1.24 |
| Poisson-30 | 45.62 | 53.11 +7.49 | 50.29 +4.67 | +2.82 |
| Poisson-40 | 45.32 | 54.24 +8.92 | 53.35 +8.03 | +0.89 |
| Average | 44.71 | 51.95 +7.24 | 51.48 +6.77 | +0.47 |

tion memories at the expense of information useful to ordinary, non-callback queries. We test this potential trade-off by evaluating stateless, P-UTL, and E-LLM on the ordinary queries that occur within the dormancy gaps of every Scenario 2 and Scenario 3 configuration. Tables 14 and 15 report execution accuracy, each memory policy's difference from the stateless reference, and the direct difference between P-UTL and E-LLM.

Both memory-enabled methods outperform the stateless control on ordinary queries in every configuration, including the longer and denser trajectories in Scenario 3. Thus, their callback gains do not require sacrificing performance on the non-memory-critical portions of these complex trajectories. Beyond this shared benefit, P-UTL outperforms E-LLM by 0.47 percentage points on average in Scenario 2 and by 1.29 points in Scenario 3. The callback advantage of P-UTL therefore does not arise from a trade-off against ordinary-query accuracy: its predictive policy is also stronger on average over the intervening queries.

### C.3 Ablations on Utility-designs

**Sign versus magnitude-aware utility.** Sign prediction assigns the same positive target, 1, to memories that help once or repeatedly. Across 470,480 memory records from all configurations, 12.0% of nonzero-utility records have $|U_H| > 1$. These records support a magnitude-aware comparison within the existing workloads. Therefore, we train a Huber regressor on raw $U_H$, matching the classifier's features, split, architecture, and optimization budget. We retain the top $B = 16$ memories and evaluate the same frozen callbacks with the pinned 4o-mini answerer. Table 16 reports per-scenario callback counts and the combined micro-average over all callbacks; differences are computed before rounding.

Table 16 shows 11 additional correct callbacks in Scenario 2 and a near tie in Scenario 3. Huber improves several Exponential and Normal settings but reduces Poisson-40 accuracy from 100.0% to 91.4%; its mean test-set rank correlation with raw utility is approximately zero in Scenario 2 and 0.284 in Scenario 3. Magnitude prediction thus offers no consistent advantage across conditions. We retain sign prediction as the simpler, more stable objective for separating helpful memories from neutral or harmful ones, without claiming exact optimization of cumulative utility.

**Marginal versus replacement-aware counterfactuals.** Marginal labels remove one memory while holding the other retrieved items fixed ($K - 1$). We test how supervision changes when

Table 15: Scenario 3 execution accuracy on ordinary dormancy-gap queries. Accuracy values are percentages. Small green tags beside method scores report percentage-point improvements over the stateless reference; the final column reports P-UTL minus E-LLM in percentage points.

| Setup | Stateless | P-UTL | E-LLM | P-UTL − E-LLM |
|---|---|---|---|---|
| N35, $C = 15$, standard | 48.04 | 55.24 +7.21 | 54.21 +6.17 | +1.03 |
| N35, $C = 20$, standard | 47.30 | 54.62 +7.32 | 53.87 +6.57 | +0.75 |
| N35, $C = 25$, standard | 46.64 | 54.19 +7.55 | 53.19 +6.55 | +1.00 |
| N35, $C = 30$, standard | 45.95 | 53.31 +7.36 | 51.15 +5.20 | +2.16 |
| P40, $C = 15$, standard | 45.83 | 53.79 +7.96 | 53.04 +7.21 | +0.75 |
| P40, $C = 20$, standard | 45.83 | 53.15 +7.32 | 52.69 +6.86 | +0.46 |
| P40, $C = 25$, standard | 45.17 | 52.84 +7.67 | 51.71 +6.54 | +1.13 |
| P40, $C = 30$, standard | 44.91 | 52.80 +7.89 | 51.59 +6.68 | +1.21 |
| N35, $C = 15$, dense | 45.88 | 54.73 +8.85 | 53.12 +7.24 | +1.61 |
| N35, $C = 20$, dense | 45.73 | 54.45 +8.72 | 53.02 +7.30 | +1.43 |
| N35, $C = 25$, dense | 45.06 | 53.49 +8.43 | 51.28 +6.22 | +2.21 |
| N35, $C = 30$, dense | 45.05 | 53.90 +8.85 | 51.62 +6.57 | +2.28 |
| P40, $C = 15$, dense | 46.42 | 53.11 +6.69 | 53.67 +7.25 | −0.56 |
| P40, $C = 20$, dense | 47.04 | 55.46 +8.42 | 53.54 +6.51 | +1.92 |
| P40, $C = 25$, dense | 47.07 | 54.70 +7.63 | 52.53 +5.46 | +2.17 |
| P40, $C = 30$, dense | 46.43 | 53.65 +7.23 | 52.50 +6.07 | +1.15 |
| Average | 46.15 | 53.96 +7.82 | 52.67 +6.53 | +1.29 |

Table 16: Utility-target ablation at $B = 16$. Accuracy and differences are percentages and percentage points, respectively.

| Evaluation | Sign target | Huber $U_H$ target | Huber − sign |
|---|---|---|---|
| Scenario 2 | 95.91 | 97.96 | +2.04 |
| Scenario 3 | 94.24 | 94.30 | +0.06 |
| Average | 94.47 | 94.81 | +0.34 |

Table 17: Counterfactual-label transitions on the 500-pair stratified diagnostic sample. Rows use marginal $K - 1$ ablation and columns use replacement-aware $K$ ablation.

| label | −1 | 0 | +1 | Total |
|---|---|---|---|---|
| −1 | 94 | 52 | 0 | 146 |
| 0 | 10 | 183 | 2 | 195 |
| +1 | 0 | 49 | 110 | 159 |
| Total | 104 | 284 | 112 | 500 |

the next-ranked memory instead fills the vacancy ($K$). We sample 500 retrieved memory–query pairs from the pooled training splits, stratified nearly equally by scenario, memory type, and original utility sign; across configurations, the sample spans 16 distinct databases. For each pair, we compare the marginal $K - 1$ label with a replacement-aware $K = 5$ label that excludes the target, filters out future writes, reruns retrieval, and admits the next candidate. Both counterfactuals use the pinned 4o-mini SQL answerer at temperature zero and are repeated three times; Table 17 reports the majority label per pair.

The two constructions preserve polarity: no label swaps directly between $-1$ and $+1$, 101 nonzero labels become neutral, and only 12 neutral labels become nonzero. Replacement primarily reduces nonzero supervision without reversing polarity in this sample. The impact of this move is merely diluting the training set (as more nonzero labels become neutral) rather than producing a completely different training geometry. Therefore, we do not consider our approach as a significant limitation.

### C.4 Computational results

**Cumulative context reading.** In Scenario 3 we provide an ablation with full context visiability. This approach is highly infeasible as it is 20x more expensive compared with a memory-dependent method. In this section, we provide an estimation of token costs needed for both categories. For each trajectory, we count the tokens read under a full context reading, and a bounded-memory reading.

The final two columns of Table 8 report cumulative context-reading estimates under the assumptions in Appendix B. The full-context estimates increase from 2.19M tokens in Scenario 1 to 22.45M

in Scenario 3, compared with 192k and 616k tokens for bounded-memory reading, with a near-quadratic increase with the number of turns.

**Online and offline costs.** Table 5 reports online decision latency and manager API costs. P-UTL makes no online manager API call, but its counterfactual labels require additional offline SQL generations and its predictor is trained on an A6000. While our training is still considerably cheaper, we have outlined the pretraining-api trade-off in our limitation section.