REPRIEVE: WHAT SURVIVES WHEN MEMORY EVICTION BECOMES REVERSIBLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agents increasingly keep evicted memories in a cold tier from which they can be swapped back before deletion, as paging systems do. Standard eviction analysis assumes removal is final; which of its tools survive reversibility? We study this with Reprieve, a minimal live–cold–dead abstraction, and show that the answer is governed by locality: when demotion is entry-local and recovery requests do not depend on tier states, every entry is an independent automaton, and three properties survive at the level of entry trajectories. The unit that repairs a failure is the exile run, the sub-threshold decisions after the entry's last rescue, not a single demotion; used prospectively, run lengths set a protection horizon without harm labels that cuts LoCoMo failures by 41–48% under oracle requests, on par with tuned access-aware cache policies and at least twice what lowering the threshold buys. Threshold monotonicity fails only on re-inserted entries, for any retriever; with exogenous requests one sweep plus replay of that set evaluates every threshold. Tombstones make failed recoveries observable, yielding certificates whose resolution is set by the number of independent conversations. We also map where locality ends: capacity triggers and retrievers that skip expired keys. Across three corpora the query schedule alone changes swap-in demand up to $4.4\times$, and reversibility pays only when cold storage is cheaper than live storage.

1 INTRODUCTION

Long-horizon agents must trade retained experience against memory growth, so practical systems decide which information remains immediately accessible (Packer et al., 2023; Sidik & Rokach, 2026; Du, 2026). Increasingly this decision is reversible: MemGPT pages content between the context window and external storage and recalls it on demand, and tiered stores keep a cold tier (Packer et al., 2023; Sidik & Rokach, 2026), much as virtual memory pages to disk (Denning, 1968). A common abstraction is the *threshold gate*: each memory entry is assigned a retention score (e.g., recency, frequency, or learned importance, Sun et al., 2025; Reddy, 2026; Chen & Cheng, 2026), and entries below a threshold λ or capacity constraint are evicted. Most existing analyses of such policies implicitly assume that eviction is irreversible: once an entry is removed, it cannot participate in future decisions.

This assumption gives threshold-based eviction two useful structural properties. First, because an entry is removed at most once, a failed query can be attributed to a single eviction decision, and per-decision harm can be aggregated into total failure risk. Second, because the evicted set grows monotonically with the threshold, a single chronological sweep can evaluate all threshold values efficiently. Harm accounting and threshold sweeps rely on them (Mattson et al., 1970).

Reversible eviction changes this: an evicted entry moves to a cold state and may return through a swap-in before eventual deletion, so whether information is lost depends on future requests and the properties above need not transfer.

We introduce *Reprieve*, a minimal live–cold–dead abstraction of reversible eviction; it is not a new memory architecture or retrieval system. Our central observation is that reversibility does not destroy the properties above; it moves them from individual decisions to entry trajectories. When demotion depends only on the entry's own state (*entry-local*) and requests do not depend on any entry's tier state (*exogenous*), each entry evolves as an independent automaton (Figure 1). Three results follow, each with a measured boundary.

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

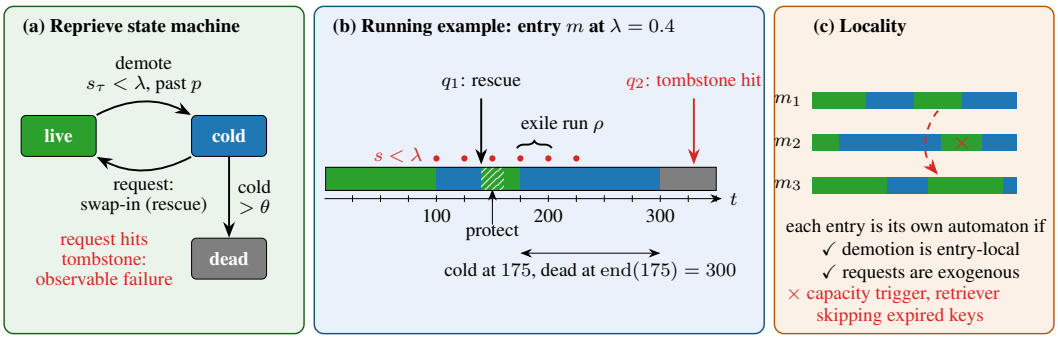


Figure 1: **Reprieve.** (a) The live–cold–dead state machine. (b) The running example of §3–7: m is rescued at 140, demoted again at 175 and dead at 300; suppressing the exile run $\{175, 200\}$ repairs q_2's failure, which the tombstone makes observable. (c) Under entry-local demotion and exogenous requests every entry is an independent automaton.

- **Attribution moves to the exile run.** Suppressing an entry's exile run always repairs its failure, whereas one blamed demotion repairs only 10–54%, as under destructive eviction; run lengths diagnosed on other conversations set a protection horizon that cuts future LoCoMo failures by 41–48%, matching tuned access-aware cache policies without harm labels.
- **Monotonicity survives outside the re-inserted set.** Violations arise only from re-inserted entries, for any retriever; with exogenous requests one sweep plus exact replay of the re-inserted set (under 10% of entries for $\lambda \leq 0.6$) evaluates every threshold. Under a capacity trigger violations escape this set but stay within a computable trigger set.
- **Failures become observable and certifiable.** Tombstones make failed recoveries observable, giving certificates whose resolution is set by independent conversations; 442 content-disjoint LongMemEval histories certify at $\alpha = 0.02$. A retriever that skips expired entries hides failures; ranking all keys restores them.

Reversibility is no free accuracy gain: when a cold entry costs as much as a live one neither semantics dominates, and it pays only when cold storage is cheaper (§8.4).

2 Related Work

Agent memory. Agent architectures separate short-lived context from persistent memory: Generative Agents retrieves from an experience stream (Park et al., 2023), CoALA organizes working, episodic, semantic, and procedural memory (Sumers et al., 2024), MemGPT moves content between tiers, MemTier uses a tiered store, and MemForest a hierarchical temporal index (Packer et al., 2023; Sidik & Rokach, 2026; Chen et al., 2026); MemoryBank forgets along a decay curve, and Mem0 and A-Mem consolidate, update and link memories (Zhong et al., 2024; Chhikara et al., 2025; Xu et al., 2025). They motivate the live/cold distinction; Reprieve analyzes it rather than proposing another system.

Retention, eviction and caching. Retention policies combine recency, frequency, relevance, and learned importance (Sun et al., 2025; Reddy, 2026; Chen & Cheng, 2026); Reprieve holds the score fixed and varies only the removal semantics. Its structures have caching precedents. The sweep is the inclusion property of stack algorithms (Mattson et al., 1970), which non-stack policies break (Belady et al., 1969); the live–cold pair is an exclusive two-level cache with demotion (Jouppi, 1990; Wong & Wilkes, 2002); θ is a cache TTL (Che et al., 2002; Choungmo Fofack et al., 2014); and a tombstone is a ghost entry, which ARC, 2Q and LIRS keep to observe misses on evicted keys (Megiddo & Modha, 2003; Johnson & Shasha, 1994; Jiang & Zhang, 2002). New here are the exile run as the minimal joint cause of a failure, containment of every monotonicity violation in the re-inserted set (or $W \cup T$ under a capacity trigger), and certification per independent conversation.

Retrieval is not eviction. Retrieval selects content for the current query (Lewis et al., 2020); eviction decides what remains retrievable. A miss arises because a needed entry was not requested or because it was deleted; our oracle replay isolates the second, and we identify when a real retriever preserves the analysis.

Risk certification. Learn-then-test (Angelopoulos et al., 2025), conformal risk control (Angelopoulos et al., 2024), and betting confidence sequences (Waudby-Smith & Ramdas, 2024) give finite-sample guarantees for bounded losses; grouped data require the group as the exchangeable unit (Dunn et al., 2023). Reprieve supplies an observable loss and certifies it per conversation. More discussion is in Appendix G.

3 MODEL

Memory stream. A conversation consists of memory entries $m \in \mathcal{M}$ and queries $q \in \mathcal{Q}$. Each entry has ingest time $a(m) \in \{1, \dots, L\}$, and each query occurs at time t_q with gold required-evidence set $E_q \subseteq \mathcal{M}$, where $a(m) < t_q$ for all $m \in E_q$. The system issues recovery requests $A_q \subseteq \mathcal{M}$; in deployment a retriever produces them, and our main replay uses oracle requests $A_q = E_q$ to isolate eviction. Decisions occur at times $\tau_1 < \tau_2 < \cdots$ with period π; at each time step, queries with $t_q = t$ are processed first, followed by ingestion of the entry with $a(m) = t$, and then any scheduled decision at t. Each entry receives a retention score $s_\tau(m)$ at decision time τ.

Assumption 1 (M1: policy-independent scores)**.** *The retention score $s_\tau(m)$ depends only on the stream and decision time τ, not on any tier state; recency ages from $a(m)$.*

Eviction semantics. We consider two semantics under the same retention policy.

Definition 1 (Destructive semantics $\mathcal{D}$)**.** *At decision time τ, a live entry m is permanently removed if $\tau - a(m) > p$ and $s_\tau(m) < \lambda$, where p is the protect window.*

Definition 2 (Reprieve semantics $\mathcal{R}_\theta$)**.** *Each entry has a tier state $z(m) \in \{\text{live}, \text{cold}, \text{dead}\}$, latest-live timestamp $\iota(m)$, and cold timestamp $c(m)$, with $\iota(m) = a(m)$ initially. Its transitions are*

$$\begin{aligned} \text{cold} \to \text{dead} \quad & \text{if } \tau - c(m) > \theta, \\ \text{live} \to \text{cold} \quad & \text{if } \tau - \iota(m) > p,\ s_\tau(m) < \lambda, \quad c(m) \leftarrow \tau, \\ \text{cold} \to \text{live} \quad & \text{when query } q \text{ requests } m \in A_q, \quad \iota(m) \leftarrow t_q. \end{aligned}$$

A request for a dead entry hits its tombstone: a swap-in failure.

Running example. Take $p = 20$, $\pi = 25$, $\theta = 100$, $\lambda = 0.4$ and one entry m ingested at $t = 1$, with score 0.35 at decisions $100, 125, 175$, 0.2 at $150, 200, 225$, and 0.5 at every other decision; queries q_1 and q_2 need m at $t = 140$ and 330. Under $\mathcal{D}$, m is removed at 100 and both queries fail. Under $\mathcal{R}_{100}$, m is demoted at 100, rescued by q_1 (so $\iota = 140$, protected until 160), demoted again at 175 and dead at 300: q_2 fails with demotion history $\{100, 175\}$.

Analysis objects. Under $\mathcal{D}$, permanent removal is an *eviction*; under $\mathcal{R}_\theta$, the live-to-cold, cold-to-live, and repeated live-to-cold transitions are *demotion*, *swap-in*, and *re-demotion*, respectively. We also call a swap-in a *rescue*. Under $\mathcal{R}_\infty$ no entry expires. The *cold lifetime* θ is how long a demoted entry's content stays in the cold tier; afterwards only a tombstone, the entry key, remains, and it is this key that makes a failed swap-in observable. A destructive store that logged deleted keys would be as observable. For threshold λ, define

$$\begin{aligned} B_\lambda(m) &= \{\tau > a(m) + p : s_\tau(m) < \lambda\}, \\ \text{end}(d) &= \min\{\tau_j : \tau_j - d > \theta\}, \\ W_\lambda &= \{m : m \text{ is swapped in at least once}\}. \end{aligned}$$

Under $\mathcal{D}$, m is removed at $\min B_\lambda(m)$. Let $H_\mathcal{D}(\lambda)$ and $H_\mathcal{R}(\lambda)$ denote the harmed pairs (q, m), $m \in E_q$, i.e. those with m removed (under $\mathcal{D}$) or dead (under $\mathcal{R}_\theta$) at t_q. For Reprieve, let $J_\mathcal{R}(\lambda) = \{(q, m) : m \in A_q,\ m \text{ is dead at } t_q\}$ denote observable failed recovery requests. Under oracle replay, $A_q = E_q$ and hence $J_\mathcal{R} = H_\mathcal{R}$; this equality need not hold for a learned retriever.

Locality. Two conditions make each entry evolve on its own.

Definition 3 (Entry-local rule; exogenous requests)**.** *A demotion rule is* entry-local *if decisions follow the fixed clock τ_j, dead is absorbing, a demotion or rescue changes only that entry, a cold entry dies at the first τ_j with $\tau_j - c(m) > \theta$, and whether a live entry is demoted at τ depends only on λ, the stream and its own history (for a never-rescued entry: $\tau - a(m) > p$ and $s^0_\tau(m) < \lambda$). M1 is entry-local with $s^0 = s$. Requests are* exogenous *if whether $m \in A_q$ is a function of the stream alone, not of any tier state: oracle requests are; so is a retriever that ranks every ingested key, which must keep the retrieval features of expired entries.*

Proposition 1 (Locality principle)**.** *Under an entry-local rule and exogenous requests, the tier trajectory of each entry depends only on $a(m)$, its own scores and the times at which it is requested, not on any other entry.*

Proof in Appendix B. Each pair (q, m) is thus a single-entry automaton. Both conditions are needed: a capacity trigger (§5.1) and a retriever that skips expired entries (Appendix C.4) each couple entries.

4 Attribution: The Repair Unit Is the Exile Run

A *witness*, a harmed pair (q, m) with $m \in E_q$ dead at t_q under $\mathcal{R}_\theta$, may have several demotions $d_1 < \cdots < d_k$ before t_q. Which, if any, caused the failure: the first, the last, or all of them? We use *cause* in the but-for counterfactual sense (Halpern & Pearl, 2005), and formalize the question through counterfactual suppression.

Definition 4 (Counterfactual suppression)**.** *For a set S of (entry, decision) pairs, $\mathcal{R}_\theta^{\neg S}$ denotes the run in which the demotion of entry m at time d is suppressed for every $(m, d) \in S$.*

A rule attributing witness (q, m) to S is sufficient *if (q, m) is not a failure in $\mathcal{R}_\theta^{\neg S}$.*

Definition 5 (Exile run)**.** *For a witness pair (q, m) under threshold λ, let r be the time of the last rescue of m before t_q, or $a(m)$ if there is none. Define*

$$\rho_\lambda(q, m) = \{d \in B_\lambda(m) : d > r + p,\ \mathrm{end}(d) < t_q\}.$$

This is the exile run*: the deletable sub-threshold decisions after the last rescue. If a demotion occurs at $d \in \rho_\lambda(q, m)$ and is not rescued, the entry is dead when q arrives. A demotion at d is* rescued *if a request for m occurs in $(d, \mathrm{end}(d)]$ (queries at a time step are processed before the decision).*

Theorem 1 (Counterfactual sufficiency)**.** *Assume M1.*

(a) ***No-rescue case*** *(threshold gate, exogenous requests). If no request other than q's asks for m during $(a(m), t_q)$, then suppressing S prevents the failure of (q, m) if and only if $S \supseteq \rho_\lambda(q, m)$. Therefore, a single demotion is sufficient only when $|\rho_\lambda(q, m)| = 1$.*

(b) ***General case.*** *Suppressing $\rho_\lambda(q, m)$ always prevents the failure, for any request policy and λ-gated rule. With exogenous requests, a set S that misses part of the run prevents it only by moving the last rescue; replaying m alone with S skipped (Proposition 1) decides which.*

Proof in Appendix B.

The exile run is thus the minimal repairing intervention: always sufficient, and necessary without other requests. This is a but-for notion; in Halpern and Pearl's modified definition the actual last demotion is itself a cause. Suppression is not monotone: a superset of the run can fail to repair (Appendix B).

Running example. The exile run of (q_2, m) is $\rho = \{175, 200\}$; 225 is excluded because $\mathrm{end}(225) = 350 > 330$. Blaming the first demotion fails the but-for test: without it m is demoted at 125, rescued at 140, and dies as before. Blaming the last fails too: without it the demotion moves to 200. Suppressing both run members defers it to 225, so m is still cold, and is rescued, at 330.

Corollary 1 (Accounting versus causation)**.** *Charging a failure to one removal is valid accounting, but even under $\mathcal{D}$ the removal is a but-for cause only if $|\{d \in B_\lambda(m) : d < t_q\}| = 1$; interventions need the full run.*

Proposition 2 (Run length and budgeted repair)**.** *Assume M1 and exogenous requests. (i) $|\rho_\lambda(q, m)| \le N(r, t_q) = |\{\tau_j > r + p : \mathrm{end}(\tau_j) < t_q\}| \approx (t_q - r - p - \theta)/\pi - 1$ (for θ a*

multiple of π), with equality if the score stays below λ after $r + p$. (ii) If each failing entry is requested once, repairing witness (q, m) costs exactly $|\rho_\lambda(q, m)|$ suppressions; under a budget the most witnesses are repaired by taking them in increasing $|\rho|$, whereas a single-decision rule repairs only those with $|\rho| = 1$.

5 Monotonicity Holds Outside the Re-inserted Set

Under $\mathcal{D}$ the harmed set grows with λ because $B_\lambda(m) \subseteq B_{\lambda'}(m)$ for $\lambda < \lambda'$, which is what lets one sweep evaluate all thresholds. A swap-in resets the protect window and may let an entry escape a demotion at another threshold. Is monotonicity lost globally, or only for re-inserted entries?

Lemma 1 (Coincidence outside the re-inserted set)**.** *Assume M1 (any request policy). Fix λ, θ and let $m \notin W_\lambda$. Then m is demoted under $\mathcal{R}_\theta$ exactly when $\mathcal{D}$ removes it, at $d = \min B_\lambda(m)$, reaches the dead state at $\mathrm{end}(d)$, and $(q, m) \in H_\mathcal{R}(\lambda)$ iff $(q, m) \in H_\mathcal{D}(\lambda)$ for every query q with $t_q > \mathrm{end}(d)$; no query requests m in $(d, \mathrm{end}(d)]$.*

Proof in Appendix B.

Theorem 2 (Localized monotonicity)**.** *Assume M1 (any request policy). For thresholds $\lambda < \lambda'$, $(q, m) \in H_\mathcal{R}(\lambda) \setminus H_\mathcal{R}(\lambda')$ implies that m is rescued before t_q in the λ' run, so $m \in W_{\lambda'}$. Therefore every monotonicity violation is contained in the re-inserted set,*

$$|H_\mathcal{R}(\lambda) \setminus H_\mathcal{R}(\lambda')| \le |\{(q, m) : m \in E_q,\ m \in W_{\lambda'}\}|,$$

and when $\theta = \infty$, no entry expires and no violation occurs.

Proof in Appendix B. Localization thus survives any retriever, even one that skips expired keys; exogeneity is what the mechanism and exact replay below need.

Proposition 3 (Mechanism of monotonicity violation)**.** *Assume M1 and exogenous requests, and let $(q, m) \in H_\mathcal{R}(\lambda) \setminus H_\mathcal{R}(\lambda')$. Let d^* be the demotion that kills m in the λ run. Then in the λ' run m is rescued at some $r_j < t_q$ with $r_j \le d^* \le r_j + p$: the killing decision falls in a protect window opened by a λ' rescue. Violations are thus rescue-induced protection, not a change in retention-score ordering.*

Proof in Appendix B.

Running example. Move q_2 to $t = 300$. At $\lambda' = 0.4$, m is demoted at 175 and still cold at 300 ($\mathrm{end}(175) = 300$, and queries precede decisions), so q_2 succeeds. At the lower $\lambda = 0.3$ the scores 0.35 are above threshold: m is live at 140, never rescued, demoted at 150 and dead at 275, so q_2 fails. The violation involves $m \in W_{0.4}$, and the killing demotion 150 lies in the window $[140, 160]$ reset by the λ' rescue, as Proposition 3 states.

Corollary 2 (One sweep, replay only on W_λ)**.** *Assume M1 and exogenous requests, and let $d_1(m) = \min B_\lambda(m)$. Then $W_\lambda = \{m : \text{some request for } m \text{ falls in } (d_1, \mathrm{end}(d_1))\}$, so one chronological pass over the prefix-minimum score records decides W_λ for every λ. For $m \notin W_\lambda$, (q, m) is harmed iff $\mathrm{end}(d_1) < t_q$; replaying the single-entry automaton only for $m \in W_\lambda$ (Proposition 1) reproduces $H_\mathcal{R}(\lambda)$ exactly (Fig. 7).*

5.1 Beyond M1: Locality Is the Boundary

Recency scores that restart after a rescue are entry-local but not M1. Rules that read other entries' states couple entries; we study one: a *capacity trigger* with capacity K fires a decision only when more than K entries are live, a firing decision demotes every eligible live entry with $s_\tau(m) < \lambda$, and expiry is still checked at every τ_j.

Theorem 3 (Locality suffices)**.** *Under any entry-local rule, Lemma 1 and Theorem 2 hold with $B_\lambda(m)$ computed from s^0 for any request policy, and Corollary 2 for exogenous requests; Theorem 1 holds whenever demotion after the latest insertion ι depends only on (τ, ι) and the stream, with ρ computed from that post-rescue score. Proposition 3 need not hold, since a rescue may change the entry's own later score.*

Proposition 4 (Coupling breaks the containment)**.** *Under a capacity trigger with any capacity $K \ge 2$, a stream of $K + 1$ entries and one query has $(q, m) \in H_\mathcal{R}(\lambda) \setminus H_\mathcal{R}(\lambda')$ for some $\lambda < \lambda'$ with m never rescued; it also violates monotonicity under $\mathcal{D}$.*

Table 1: Which result holds under which condition (✓ proved; × fails; – not established).

result	M1	entry-local	capacity trigger	retriever skipping expired keys
single-entry automaton (Prop. 1)	✓	✓	×	×
exile run repairs (Thm. 1)	✓	renewal rules	sufficiency	sufficiency
violations in W (Thm. 2)	✓	✓	in $W \cup T$ (Prop. 5)	✓
sweep + replay on W (Cor. 2)	✓	✓	two full replays	×
certificate (Thm. 4)	✓	✓	✓	valid, but records no failure

Proposition 5 (Localisation under a capacity trigger)**.** *Under a capacity trigger with M1 scores and exogenous requests, let* $\mathrm{Fire}(\lambda)$ *be the decisions that fire in the* λ*-run and* $T_{\lambda,\lambda'}$ *the entries whose first demotion in the* λ*-run occurs at a decision in* $\mathrm{Fire}(\lambda) \setminus \mathrm{Fire}(\lambda')$*. Then every* $(q, m) \in H_{\mathcal{R}}(\lambda) \setminus H_{\mathcal{R}}(\lambda')$ *has* $m \in W_\lambda \cup W_{\lambda'} \cup T_{\lambda,\lambda'}$*; all three sets are computable from two full replays.*

Proofs in Appendix B.

On real data violations escape W but never $W \cup T$ (Appendix C.3). MemGPT-like paging and LRU tiering are coupled too, yet keep every observed violation in W (Appendix C.5). Table 1 summarizes the scope.

6 The Query Schedule Is a Structural Confound

Reversible eviction is observable only when queries occur during the memory lifecycle. If all queries come at the end of the stream, a demoted entry can still be swapped in, but never demoted again, so the repeated demotions and resets that distinguish reversible memory disappear.

Proposition 6 (End-of-stream schedules hide reversible dynamics)**.** *Under an entry-local rule, if every query occurs after the final eviction decision, then no entry is demoted more than once, the three attribution rules of §8.1 (blame the first, the last, or all demotions) coincide, and* $H_{\mathcal{R}}(\lambda)$ *is monotone in* λ*.*

Proof in Appendix B.

The schedule is thus part of the evaluation protocol: benchmarks must interleave queries between demotion and expiry (§8.3).

7 Observability and Certificates on the Swap-in Failure Rate

Definition 6 (Observable harm)**.** *A failure is* observable *if it is a deterministic function of the system state and request trace, without access to gold evidence.*

Under Reprieve, a query requesting an expired entry resolves to a tombstone and produces an observable swap-in failure of the requested key; whether the retriever requested all gold evidence remains unobserved.

Running example. When q_2 requests m at 330, the store finds m's tombstone and records a failed recovery without any gold label, so this conversation contributes $F_i = 1/2$ to the certificate below. Under $\mathcal{D}$ the key is gone.

7.1 Conversation-level risk certification

We treat each conversation as the statistical unit because queries within a conversation are dependent: let $X_1, \ldots, X_n$ be i.i.d. conversations, and for fixed (λ, θ) define the conversation-level loss

$$F_i(\lambda) = \frac{|\{q \in \mathcal{Q}_i : (q, m) \in J_{\mathcal{R}}(\lambda) \text{ for some } m \in A_q\}|}{|\mathcal{Q}_i|}. \tag{1}$$

the fraction of queries with a failed recovery request (under oracle requests, the harm rate); we write $F_i(\theta)$ when θ varies.

Theorem 4 (Transfer of certificates). *For any deterministic replay policy, coupled or not, $F_i(\lambda)$ is a function of the conversation X_i, so the losses are i.i.d. across conversations. Hence, with the scorer, request policy and configuration grid fixed independently of the calibration conversations, any distribution-free guarantee for bounded i.i.d. losses (Hoeffding; learn-then-test, Angelopoulos et al., 2025; betting, Waudby-Smith & Ramdas, 2024) applies; methods that also need a monotone loss, such as conformal risk control (Angelopoulos et al., 2024), apply along θ with exogenous requests (Lemma 2) but not along λ.*

Proof in Appendix B.

The certificate does not cover unrequested evidence or answer correctness.

7.2 Certified cold lifetime

Lemma 2 (Monotonicity in cold lifetime). *Under an entry-local rule and exogenous requests, for fixed λ, if $\theta < \theta'$ then the set of swap-in failures under $\mathcal{R}_{\theta'}$ is contained in that under $\mathcal{R}_\theta$, and therefore $F_i(\theta') \le F_i(\theta)$.*

Proof in Appendix B.

We test $\theta_K = \infty, \theta_{K-1}, \dots$ in a fixed sequence and return the last lifetime before the first whose upper confidence bound on $\mathbb{E}[F(\theta_k)]$ exceeds α; this needs no union bound for any policy, and Lemma 2 makes the order natural, using an anytime-valid betting confidence sequence (Waudby-Smith & Ramdas, 2024) (a Hoeffding bound works identically but is looser).

Corollary 3 (Resolution limit). *A valid certificate that is monotone in the observed losses (e.g. Hoeffding or Clopper–Pearson) cannot certify any $\alpha < 1-\delta^{1/n} \approx \ln(1/\delta)/n$ from n independent conversations, however many queries each contains; this is the classical zero-numerator bound (Hanley & Lippman-Hand, 1983) applied to conversations. For $\delta = 0.1$ the limit is 0.32, 0.11 and 0.009 at $n = 6$, 19 and 265.*

8 Empirical Validation

Setup. §8.1–8.3 and the certificates are deterministic replays that use no model API; only the end task of §8.4 calls an LLM reader and judge (Appendix F). *Data:* LoCoMo (Maharana et al., 2024) (10 conversations, 1,531 questions), PerLTQA (Du et al., 2024) (31 streams, 8,214 queries), LongMemEval-S (Wu et al., 2025) (500 histories, 479 with usable evidence), all with per-entry gold evidence. *Requests:* oracle ($A_q = E_q$) unless stated; BM25 top-10 over all keys on LoCoMo and over non-expired keys on PerLTQA (Appendices D.6, D.10). *Scores:* semantic overlap, recency and re-mention frequency, policy-independent; $(\pi, p) = (25, 20)$, or $(5, 4)$ on PerLTQA. *Metrics: compression* is the share of entries ever demoted, *storage* the live-plus-cold count relative to keep-all (policies are matched on it), the *load-bearing fraction* the share of entries some query needs, and *swap-in demand* the share of demoted entries swapped in at least once. *Inference:* conversation clusters, 95% bootstrap percentiles.

8.1 Is the Blamed Demotion a Cause? (Theorem 1)

Finding. *Blaming one demotion rarely explains a failure; the exile run does, and it is the unit that repairs.*

- Suppressing the blamed demotion repairs 10.5–33.3% of LoCoMo and 32–37% of PerLTQA failures (12–54% across scorers and variants); a single removal under $\mathcal{D}$ fares similarly (22–39%), so reversibility mainly moves the run's start to the last rescue (Appendix D.5).
- Exile runs average 4.5–7.7 decisions on LoCoMo and 2.7 on PerLTQA; the share with $|\rho| = 1$ predicts the rates within 3.3 points.
- Prospectively, runs from other conversations set a pin-after-access horizon that cuts LoCoMo failures by 41–48% (62–63% under BM25): as good as the best fixed horizon or a tuned frequency boost, 16–29 points above LRU and 21–32 above a lower λ at equal storage; no gain on PerLTQA (Appendix D.7).

One suppression mostly shifts removal along a long run (Fig. 2c). The prospective rule protects a requested entry for $p + k\pi$ after its latest request, with k the 80th percentile of the extension each diagnosed run of the other conversations needed (Appendix D.6).

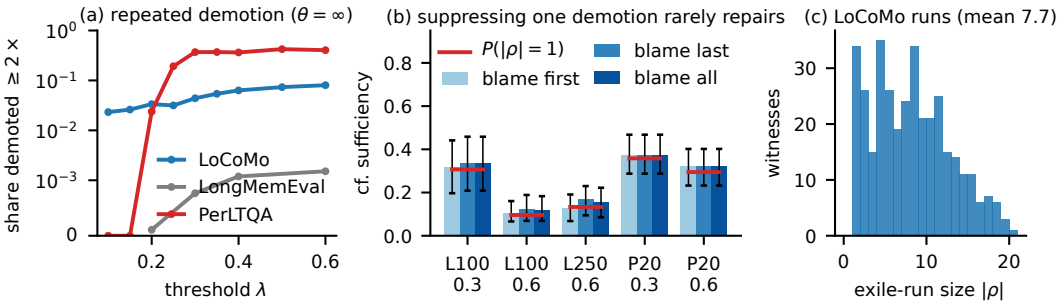


Figure 2: Failure attribution. Reversible eviction creates multi-demotion exile runs. (a) Share of demoted entries demoted at least twice ($\theta = \infty$). (b) Counterfactual sufficiency of three single-decision rules with 95% cluster intervals and the predicted $P(|\rho| = 1)$; L = LoCoMo, P = PerLTQA, labels give θ and λ. (c) Exile-run sizes of LoCoMo witnesses ($\theta = 100$, $\lambda = 0.6$).

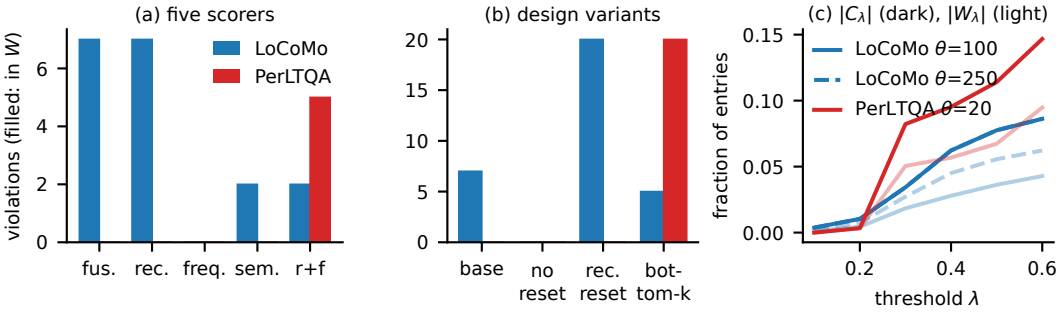


Figure 3: Localized monotonicity violations. (a,b) Monotonicity violations (outline) and those inside W (filled) for five scorers and four design variants; every violation is inside W. (c) Candidate set C_λ ($\supseteq W_\lambda$: evicted under $\mathcal{D}$ before a query needs them) and re-inserted set W_λ as fractions of entries.

8.2 Where Does Monotonicity Fail? (Theorem 2)

Finding. *Every observed violation lies in the re-inserted set, and exact replay is needed only on that set.*

- LoCoMo: 7 violations among the 359 pairs harmed anywhere on the 59-point grid ($\theta = 100$) and 2 among 182 ($\theta = 250$); none at $\theta = \infty$ or on PerLTQA with the default scorer; all in W across five scorers, four variants and BM25 requests.
- Removing the protect-window reset removes every violation ($7 \to 0$), as Proposition 3 predicts.
- Per-entry replay is exact in $18/18$ settings; it is needed only on W_λ, at most 6.2% (LoCoMo) and 9.5% (PerLTQA) of entries for $\lambda \leq 0.6$.

Each violation is the rescue-reset mechanism of the running example in §5 (Appendix C.1).

8.3 How Much Does the Query Schedule Confound? (Proposition 6)

Finding. *The query schedule alone changes swap-in demand several-fold.*

- End-of-stream versus interleaved queries: 3.3× on LoCoMo (0.072 → 0.235 at $\lambda = 0.3$), 4.4× on LongMemEval, 1.2× on PerLTQA.
- At the end of the stream demand reaches the load-bearing fraction on LongMemEval and LoCoMo (0.0036, 0.24) and approaches it on PerLTQA (0.77 vs 0.87); interleaved queries stay lower (0.0016, 0.09; $\lambda = 0.6$).

At the end of the stream every needed demoted entry is still cold when requested, so demand equals the load-bearing fraction and re-demotion disappears (Fig. 4a,b; Appendix D.2). TTL-cache theory fits random schedules only (Appendix D.3).

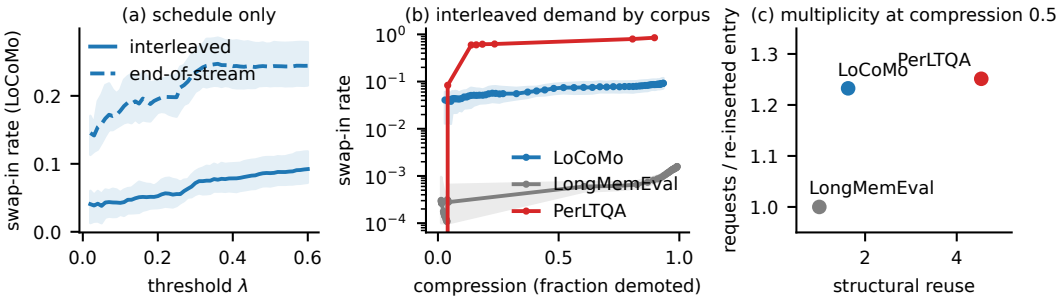


Figure 4: Query timing controls reversible dynamics. (a) The end-of-stream schedule raises swap-in demand. (b) Interleaved demand by corpus, ordered by the load-bearing fraction. (c) Requests per re-inserted entry at compression 0.5.

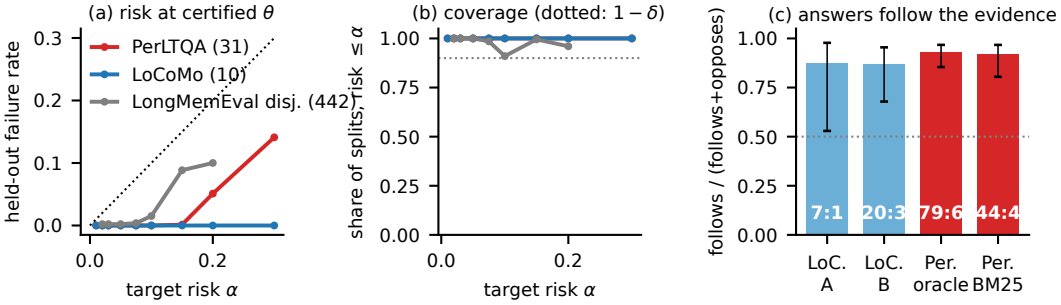


Figure 5: Risk certification and end-task validation. (a,b) Betting certificates control held-out failure risk. (c) When exactly one policy is right, the share where it holds the evidence (Wilson 95%, questions treated as independent): LoCoMo matches A ($\mathcal{D}_{0.23}$, $\mathcal{R}_{50}$ at 0.27) and B ($\mathcal{D}_{0.30}$, $\mathcal{R}_{100}$ at 0.40); PerLTQA with oracle and BM25 requests.

8.4 Do Certificates Hold and Reach Answers? (Theorem 4)

Finding. *Certificates are valid, with resolution set by independent units; answers follow the evidence (Fig. 5); reversibility pays only when cold storage is cheaper.*

- Finite θ is certified at $\alpha = 0.2$ on PerLTQA (31 streams), never on LoCoMo (10), and at $\alpha = 0.02$ on 442 content-disjoint LongMemEval histories (held-out risk 0.002); betting bound, fixed-sequence selection, $\delta = 0.1$ (Appendices E, E.3).
- When exactly one of $\mathcal{D}$, $\mathcal{R}_\theta$ is right, it holds the evidence in $7/8$, $20/23$ (LoCoMo), $79/85$ and $44/48$ (PerLTQA) cases.
- On 100 random LoCoMo questions $\mathcal{R}_\theta$ minus $\mathcal{D}$ accuracy is $+4.0$ and $+1.0$ points, both intervals containing zero.
- At equal pricing neither semantics dominates the harm–storage frontier; with cold entries at ≤ 0.75 of a live entry's cost Reprieve never loses on LoCoMo or PerLTQA, and gains 9–11 points at 0.25 (Appendix D.4).

9 Limitations

Harm is a proxy for answer error, checked with one reader and one judge. Scores are heuristic or simply learned; most replays request gold evidence (BM25 on two corpora). Beyond entry-local demotion we prove only $W \cup T$ localization. Storage counts entries; LoCoMo has ten conversations.

10 Conclusion

Reversibility moves eviction analysis from decisions to entry trajectories. Under entry-local demotion and exogenous requests, reversible memory should be repaired by exile runs, swept once, indexed with tombstones, and certified per independent conversation.

Reproducibility Statement

All replay experiments (Sections 4–8, Appendices C–E) are deterministic, need no model API and no GPU (except the optional dense-embedding scorer), and run on one CPU core from cached retention scores; seeds, grids, clocks and schedules are listed in Appendix A. Proofs of all stated results are in Appendix B. For the answer-level checks (Appendices F and F.1) we release the frozen question sets and membership rules, frozen necessary points, raw reader answers, raw and repaired judge outputs with every correction and adjudication logged, run manifests with model identifiers and decoding settings, and cost records. The supplementary material contains the code, the replay outputs from which every number and figure is regenerated, and the dataset loaders; LoCoMo, LongMemEval-S and PerLTQA are public.

LLM Usage Disclosure

Large language models are part of the experimental method: `gemini-2.5-flash` is the reader and `gpt-5-mini` the rubric generator and judge in the answer-level checks (Appendices F and F.1). AI coding assistants were used to write analysis scripts, to audit the frozen rubrics and a sample of judgments and draft the source-only rubric corrections, and to apply the logged, format-only repairs of judge outputs and one logged adjudication; every such decision is logged and released. Generative AI tools were also used for literature retrieval and for language editing. The authors reviewed and verified all AI-assisted outputs and take full responsibility for the content of the paper.

References

Anastasios N. Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=33XGfHLtZg.

Anastasios N. Angelopoulos, Stephen Bates, Emmanuel J. Candès, Michael I. Jordan, and Lihua Lei. Learn then test: Calibrating predictive algorithms to achieve risk control. *The Annals of Applied Statistics*, 19(2):1641–1662, 2025. doi: 10.1214/24-AOAS1998.

L. A. Belady, R. A. Nelson, and G. S. Shedler. An anomaly in space-time characteristics of certain programs running in a paging machine. *Communications of the ACM*, 12(6):349–353, 1969. doi: 10.1145/363011.363155.

Sam Bowyer, Laurence Aitchison, and Desi R. Ivanova. Position: Don't use the CLT in LLM evals with fewer than a few hundred datapoints. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 81143–81184, 2025. URL https://proceedings.mlr.press/v267/bowyer25a.html.

Hao Che, Ye Tung, and Zhijun Wang. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, 2002. doi: 10.1109/JSAC.2002.801752.

Han Chen, Zining Zhang, Wenqi Pei, Bingsheng He, Ming Wu, Jason Zeng, Michael Heinrich, Wei Wu, and Hongbao Zhang. MemForest: An efficient agent memory system with hierarchical temporal indexing. *arXiv preprint arXiv:2605.23986*, 2026. doi: 10.48550/arXiv.2605.23986.

Zhibao Chen and Qian Cheng. Learning what to remember: A cognitively grounded multi-factor value model for agentic memory. *arXiv preprint arXiv:2606.12945*, 2026. doi: 10.48550/arXiv.2606.12945.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI)*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160.

Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks*, 65:212–231, 2014. doi: 10.1016/j.comnet.2014.03.006.

Peter J. Denning. The working set model for program behavior. *Communications of the ACM*, 11 (5):323–333, 1968. doi: 10.1145/363095.363141.

Pengfei Du. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026. doi: 10.48550/arXiv.2603.07670.

Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. PerLTQA: A personal long-term memory dataset for memory classification, retrieval, and fusion in question answering. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pp. 152–164, 2024. doi: 10.18653/v1/2024.sighan-1.18.

Robin Dunn, Larry Wasserman, and Aaditya Ramdas. Distribution-free prediction sets for two-layer hierarchical models. *Journal of the American Statistical Association*, 118(544):2491–2502, 2023. doi: 10.1080/01621459.2022.2060112.

Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *The British Journal for the Philosophy of Science*, 56(4):843–887, 2005. doi: 10.1093/bjps/axi147.

James A. Hanley and Abby Lippman-Hand. If nothing goes wrong, is everything all right? Interpreting zero numerators. *JAMA*, 249(13):1743–1745, 1983. doi: 10.1001/jama.1983.03330370053031.

Song Jiang and Xiaodong Zhang. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 31–42, 2002. doi: 10.1145/511334.511340.

Theodore Johnson and Dennis Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pp. 439–450, 1994. URL https://www.vldb.org/conf/1994/P439.PDF.

Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA)*, pp. 364–373, 1990. doi: 10.1109/ISCA.1990.134547.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 9459–9474, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pp. 13851–13870, 2024. doi: 10.18653/v1/2024.acl-long.747.

Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970. doi: 10.1147/sj.92.0078.

Nimrod Megiddo and Dharmendra S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, pp. 115–130, 2003. URL https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024. doi: 10.48550/arXiv.2411.00640.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. doi: 10.48550/arXiv.2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763.

Pranath Reddy. Selective memory retention for long-horizon LLM agents. *arXiv preprint arXiv:2606.29178*, 2026. doi: 10.48550/arXiv.2606.29178.

Bronislav Sidik and Lior Rokach. MEMTIER: Tiered memory architecture and retrieval bottleneck analysis for long-running autonomous AI agents. *arXiv preprint arXiv:2605.03675*, 2026. doi: 10.48550/arXiv.2605.03675.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=1i6ZCvflQJ.

Changzhi Sun, Xiangyu Chen, Jixiang Luo, Dell Zhang, and Xuelong Li. Beyond heuristics: A decision-theoretic framework for agent memory management. *arXiv preprint arXiv:2512.21567*, 2025. doi: 10.48550/arXiv.2512.21567.

Kuan Wang. MemDelta: Controlled baselines and hidden confounds in agent memory evaluation. *arXiv preprint arXiv:2606.29914*, 2026. doi: 10.48550/arXiv.2606.29914.

Ian Waudby-Smith and Aaditya Ramdas. Estimating means of bounded random variables by betting. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 86(1):1–27, 2024. doi: 10.1093/jrsssb/qkad009.

Theodore M. Wong and John Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the USENIX Annual Technical Conference*, pp. 161–175, 2002. URL https://www.usenix.org/legacy/event/usenix02/tech/general/wong.html.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=pZiyCaVuti.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

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

APPENDIX OVERVIEW

The proofs show that the exact guarantees rest on locality (entry-local demotion with exogenous requests); the experiments show that the predicted breakdowns occur exactly where locality fails, and that the structural findings survive other scorers, variants and a real retriever. Only Appendix F uses a model API.

Appendix	Content	Supports
A	notation, running example in full, corpora, decision clock, statistics	§3, §8
B	proofs of all results, counterexamples ($p < \pi$, capacity $K \geq 2$)	§3–7
C	variants, factorisation boundary, capacity trigger, retriever coupling, paging	Thm. 3, Prop. 4
D	scorers, synthetic plane, cold lifetime and TTL theory, frontier and differential pricing, repair (incl. destructive attribution), prospective rule, access-aware baselines, learned scorer, robustness, retrieval	§8.1–8.4
E	storage at certified lifetime, synthetic sizes, content-disjoint LongMemEval	§7, §8.4
F	end-task protocol (reader, rubric judge, sampling, cost), PerLTQA checks	§8.4
G	additional related work	§2

A CONVENTIONS, WORKED EXAMPLE AND REPRODUCIBILITY

Table 2: Symbols.

λ	threshold	$B_\lambda(m)$	eligible sub-threshold decisions of m
p	protect window	$\mathrm{end}(d)$	expiry of a demotion at d: first τ with $\tau - d > \theta$
θ	cold lifetime (time content stays cold)	$\rho_\lambda(q, m)$	exile run: deletable decisions after the last rescue
π	decision period	W_λ	re-inserted set: entries swapped in at least once
$a(m)$	ingest time	C_λ	candidates: evicted under $\mathcal{D}$ before a query needs them
$\iota(m)$	latest insertion (reset by a rescue)	$H_\mathcal{D}, H_\mathcal{R}$	harmed pairs (q, m) under $\mathcal{D}$ and $\mathcal{R}_\theta$
$c(m)$	cold-since time	$F_i(\lambda)$	per-conversation observable request-failure rate (§7)
$s_\tau(m)$, $s^0_\tau(m)$	retention score; its never-rescued value	$\mathcal{R}_\theta^{\neg S}$	run with the demotions in S suppressed
$\mathrm{Fire}(\lambda)$	decisions that fire in the λ-run (capacity trigger)	$T_{\lambda,\lambda'}$	entries first demoted at a decision in $\mathrm{Fire}(\lambda) \setminus \mathrm{Fire}(\lambda')$

Conventions. Table 2 lists the symbols. We set $\min \emptyset = \infty$ and $\mathrm{end}(\infty) = \infty$, take $p \geq 0$, and let decisions continue on the clock after the last ingest until the last query. A request for a live entry changes nothing, except under the renewal rules of Appendix D.6. Suppression sets and exile runs are sets of decisions and may contain decisions at which m is already cold; $\min \rho_\lambda(q, m)$ is the actual last demotion. The latest-live timestamp $\iota(m)$ and cold timestamp $c(m)$ of §3 are the latest insertion and cold-since times of the table.

Running example. The boxes of §3–7 use $\lambda = 0.4$, $p = 20$, $\pi = 25$, $\theta = 100$ and one entry m with $a(m) = 1$, scores below λ at decisions $100, 125, 150, 175, 200, 225$ and 0.5 elsewhere, and queries needing m at 140 and 330 (Fig. 1b); every boxed outcome was checked with the replay engine.

Corpora (Table 6). We need entry-level gold evidence to attribute anything. LoCoMo (Maharana et al., 2024) and LongMemEval-S (Wu et al., 2025) provide it; PerLTQA (Du et al., 2024) links each QA to a memory record. The validated LoCoMo loader retains $1{,}531$ questions over 10 native conversations, skips 455 questions without a usable mapping, and drops 9 invalid evidence references. PerLTQA provides 32 QA views, but one character has no matching memory base, leaving 31 replay streams ($8{,}214$ queries). For LongMemEval, the loader groups 500 question histories into

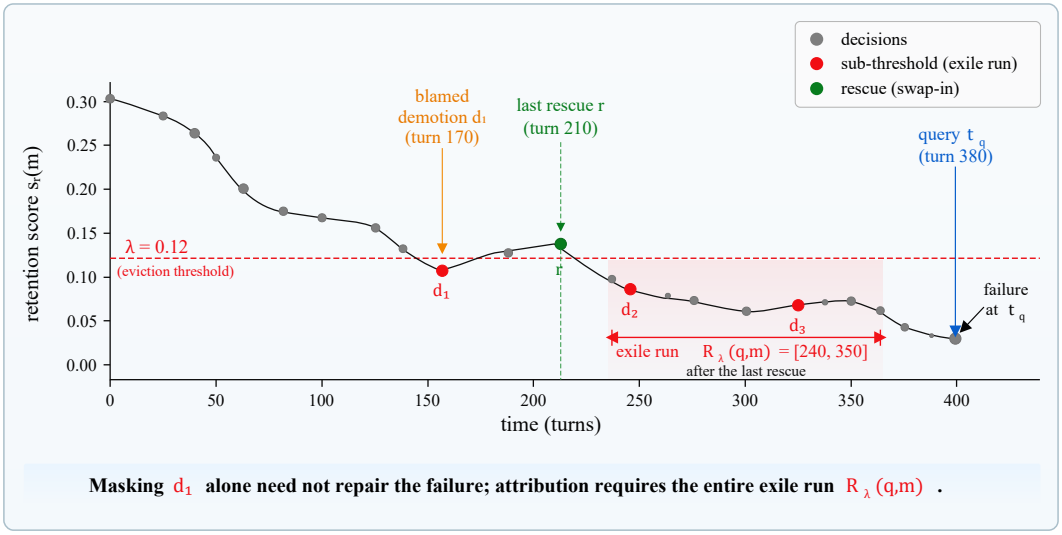


Figure 6: Why a single demotion is not a cause (an illustrative trajectory, not the running example). The blamed demotion d_1 precedes the last rescue r, so suppressing it cannot repair the failure; after r the score stays below λ, and the sub-threshold decisions whose expiry precedes the query form the exile run $\rho_\lambda(q, m)$, which is what repair must suppress.

50 pseudo-clusters of ten; 4,233 source session IDs occur in more than one pseudo-cluster, so these are not established independent conversations.

Query schedules. LoCoMo and LongMemEval pose their questions after the full history; we construct an *interleaved* schedule from their session annotations (a query fires at the first turn after its last evidence session; queries whose evidence completes in the final session fire at $L + 1$). It is the earliest time each question becomes answerable, the extreme most favourable to observing re-insertion, as *end* is the opposite extreme; both are constructed. PerLTQA is a static memory base with QA and has none; we impose two: *end* (every query at $L + 1$) for cross-corpus placement, where comparability matters, and *uniform* (each query at a uniform random time after its evidence clears the protect window, fixed seed) for state-machine experiments, where re-insertion must be observable (Prop. 6). Cross-corpus statements hold the schedule fixed.

Decision clock. $\pi = 25$, $p = 20$ for LoCoMo and LongMemEval; PerLTQA streams are ≈ 67 records long, so $\pi = 5$, $p = 4$; its swap-in rates move by less than 0.03 between the two clocks for $\lambda \geq 0.29$ and by up to 0.16 for $\lambda \leq 0.27$.

Compression and storage matching. Thresholds are not comparable across corpora (the same λ demotes 48% of LoCoMo and 14% of PerLTQA at 0.3); cross-corpus scorer readings use matched compression, the fraction of entries demoted. Matched-storage experiments instead pair operating points by the normalized cumulative live-plus-cold entry count. They never use harm or accuracy to select λ or θ.

Statistics. The unit is the conversation; intervals are 95% cluster-bootstrap percentile intervals with 2,000 resamples (seed 17; seed 3 for the candidate-set intervals of Appendix C.2) for the replay experiments and 10,000 for the matched-storage and end-task experiments; certification uses 200 random 60/40 splits (seed 7 real, 11 synthetic) and $\delta = 0.1$.

B Deferred proofs

Proof of Proposition 1. By Definition 3, every transition of m (demotion at a fixed decision time, expiry at the first τ_j with $\tau_j - c(m) > \theta$, rescue by a request) reads only λ, θ, the stream, m's own

n	Clopper–Pearson	betting	Hoeffding
10	0.206	0.259	0.339
19	0.114	0.146	0.246
31	0.072	0.092	0.193
50	0.045	0.058	0.152
100	0.023	0.030	0.107
200	0.011	0.015	0.076
500	0.005	0.006	0.048
1000	0.002	0.003	0.034
conversations needed for an all-zero calibration set			
certify $\alpha = 0.02$	$n \geq 114$	$n \geq 149$	
certify $\alpha = 0.05$	$n \geq 45$	$n \geq 59$	
certify $\alpha = 0.1$	$n \geq 22$	$n \geq 29$	
certify $\alpha = 0.2$	$n \geq 11$	$n \geq 14$	

Table 3: Certificate resolution (Corollary 3, $\delta = 0.1$). Above, the tightest valid upper bound on a loss that is zero on all n calibration conversations; below, the conversations each bound needs to certify α.

state and the times at which m is requested; with exogenous requests these times are fixed by the stream. Induction over events gives the claim. □

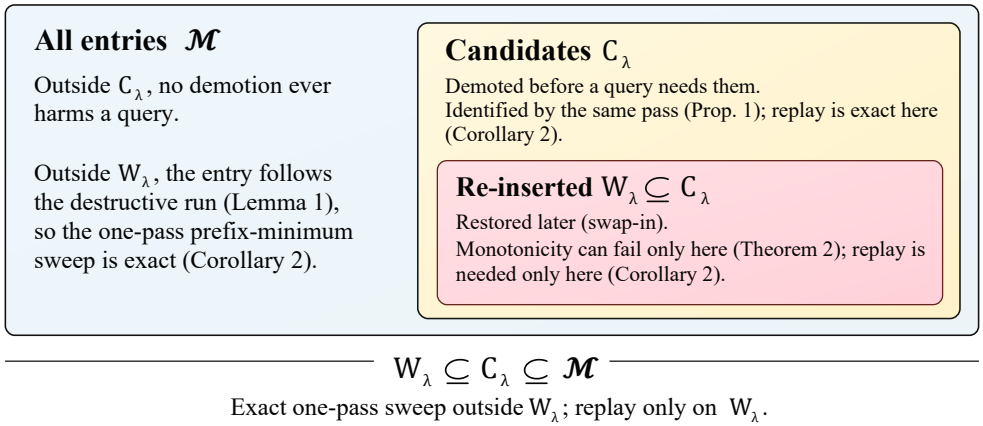


Figure 7: Where each computation is exact. The one-pass sweep is exact for every entry outside the re-inserted set W_λ, which the same pass identifies; per-entry replay is needed only on W_λ; monotonicity between λ and λ' can fail only on $W_{\lambda'}$.

Proof of Theorem 1. (b) The replay criterion is Proposition 1 applied to $\mathcal{R}_\theta^{\neg S}$: masking changes no score (M1) and touches no other entry, so the masked automaton is exact, and (q, m) fails iff m is dead at t_q, i.e. iff some demotion before t_q expired without a rescue. Sufficiency of $S = \rho_\lambda(q, m)$: every element of the run exceeds $r + p$, so masking it leaves the history up to and including the rescue at r unchanged. Afterwards m can be demoted only at unmasked $d \in B_\lambda(m)$ with $d > r+p$, all of which have $\mathrm{end}(d) \geq t_q$; as end is non-decreasing, every later demotion also expires no earlier than t_q, so m is not dead at t_q. This argument uses only that the whole system is unchanged before $\min \rho_\lambda(q, m)$ and that every demotion of m is at an eligible decision with $s_d(m) < \lambda$; it holds for any request policy and any λ-gated rule, including the capacity trigger. For (a), with no rescue $\iota(m) = a(m)$ throughout, so eligibility holds at every $d \in B_\lambda(m)$; the entry is demoted at the first unmasked element of $B_\lambda(m)$, stays cold, and is dead at t_q iff that element lies in $\rho_\lambda(q, m)$. Hence the failure persists iff $\rho_\lambda(q, m) \setminus S \neq \emptyset$. □

With rescues the run need not be the unique minimal cause. On the paper's clock ($p = 20$, $\pi = 25$, $\theta = 100$, $a(m) = 1$), let the score be below λ exactly at decisions $100, 150, 175, 200$, and let requests for m arrive at 140, 170 and 310. Actual run: demoted at 100, rescued at 140 ($\iota = 140$), protected at 150, live at 170, demoted at 175, dead at 300; the request at 310 fails and $\rho_\lambda = \{175\}$ (since $\mathrm{end}(200) = 325 > 310$). Suppressing only the demotion at 100: m is demoted at 150, rescued at 170 ($\iota = 170$), protected at 175, demoted at 200 and still cold at 310, where it is rescued. A single demotion outside the run is thus a but-for cause, even though $p < \pi$: a shifted rescue need only protect one decision. (Checked with the replay engine.)

Suppression is not monotone. On the same clock let the score be below λ exactly at $25, 50, 75, 100$, with requests for m at 30 and 40 and a query needing m at 320. Actual run: demoted at 25, rescued at 30 ($\iota = 30$), protected at 50, demoted at 75, dead at $\mathrm{end}(75) = 200$; $\rho_\lambda = \{75, 100\}$, and suppressing it repairs the failure. Suppressing $\{25, 75, 100\} \supset \rho_\lambda$ does not: m stays live at 25, so the request at 30 is no rescue, m is demoted at 50 and dead at 175. A superset of the run can therefore fail to repair; Theorem 1(b) is a statement about the run itself.

Proof of Proposition 2. (i) By definition $\rho_\lambda(q, m) \subseteq \{\tau_j > r + p : \mathrm{end}(\tau_j) < t_q\}$, with equality when every such decision is sub-threshold; since $\mathrm{end}(\tau_j) < t_q$ iff some decision lies in $(\tau_j + \theta, t_q)$, the count is $\lceil (t_q - r - p - \theta - \pi)/\pi \rceil$ when θ is a multiple of π, since then $\mathrm{end}(\tau_j) = \tau_j + \theta + \pi$ (running example: $\lceil 45/25 \rceil = 2$). (ii) By Theorem 1(a), a suppression set repairs (q, m) iff it contains $\rho_\lambda(q, m)$; witnesses of different entries need disjoint sets, so each repair costs $|\rho|$ and maximizing the number repaired under a budget is a knapsack with unit values, solved by taking costs in increasing order. A single suppression contains ρ only if $|\rho| = 1$. □

Proof of Corollary 3. Losses in $\{0, 1\}$ are allowed. If the certificate accepts α on some sample, monotonicity makes it accept α on the all-zero sample. For Bernoulli losses with mean $\mu \in (\alpha, 1 - \delta^{1/n})$ the all-zero sample has probability $(1 - \mu)^n > \delta$, so the procedure would wrongly accept with probability above δ, contradicting validity. Hence $\alpha \geq 1 - \delta^{1/n}$; and $1 - \delta^{1/n} = 1 - e^{-\ln(1/\delta)/n} \approx \ln(1/\delta)/n$. □

Proof of Lemma 1. The argument does not use the requests: never swapped in, m has $\iota(m) = a(m)$ throughout, so its eligibility equals $\mathcal{D}$'s at every decision; by M1 its scores coincide; it is therefore demoted exactly when $\mathcal{D}$ evicts it, at $d = \min B_\lambda(m)$. Being never rescued, no request for it falls in $(d, \mathrm{end}(d)]$, so it stays cold until $\mathrm{end}(d)$ and is dead afterwards; a query needing m after $\mathrm{end}(d)$ is harmed under both semantics. □

Proof of Theorem 2. Let $(q, m) \in H_\mathcal{R}(\lambda)$ and suppose m is not rescued before t_q in the λ' run. In the λ run m is dead at t_q, so it was demoted at least once; its first demotion is at $d_1 = \min B_\lambda(m)$, before any rescue, and its fatal demotion $d^* \geq d_1$ has $\mathrm{end}(d^*) < t_q$. In the λ' run m is live with $\iota = a(m)$ until its first demotion, which occurs at $d' = \min B_{\lambda'}(m) \leq d_1$ because $B_\lambda(m) \subseteq B_{\lambda'}(m)$; not being rescued before t_q, it stays cold and dies at $\mathrm{end}(d') \leq \mathrm{end}(d^*) < t_q$, so $(q, m) \in H_\mathcal{R}(\lambda')$. Hence a violation needs a rescue of m before t_q in the λ' run. No property of the requests is used. The bound counts the remaining pairs; at $\theta = \infty$ nothing dies. □

Proof of Proposition 3. In the λ run m is dead at t_q, so there is a demotion d^* with $\mathrm{end}(d^*) < t_q$ and no request for m in $(d^*, \mathrm{end}(d^*)]$; as a demotion it is eligible and sub-threshold, so $d^* > a(m) + p$ and $s_{d^*}(m) < \lambda < \lambda'$ (M1). Requests are exogenous, so the λ' run sees the same request times. Consider m in the λ' run right after decision d^*. If it is dead, it is dead at t_q. If it is cold, it was demoted at some $c \leq d^*$ and not rescued up to d^*; with no request in $(d^*, \mathrm{end}(d^*)] \supseteq (d^*, \mathrm{end}(c)]$ it dies at $\mathrm{end}(c) \leq \mathrm{end}(d^*) < t_q$. Either way $(q, m) \in H_\mathcal{R}(\lambda')$, a contradiction. So m is live after d^* although $s_{d^*}(m) < \lambda'$: it was ineligible, $d^* - \iota' \leq p$ for its latest insertion ι'. Since $d^* > a(m) + p$, $\iota' \neq a(m)$, so $\iota' = r_j$ is a λ' rescue with $r_j \leq d^* \leq r_j + p$ and $r_j < t_q$; in particular $m \in W_{\lambda'}$. □

Proof of Proposition 6. A second demotion requires a swap-in followed by a later decision; with every query after the final decision, no swap-in precedes a decision, so every entry is demoted at most once, at $d = \min B_\lambda(m)$ (computed from s^0 for an entry-local rule), and the three attribution rules blame the same demotion. The entry is dead at t_q iff a decision $\mathrm{end}(d)$ exists; if none does, it

stays cold and is swapped in. Since $\min B_\lambda(m)$ is non-increasing in λ and end is non-decreasing, harm at λ implies harm at $\lambda' > \lambda$. □

Proof of Lemma 2. By Proposition 1 consider one entry, with the same request times in both runs. The only θ-dependent transition is expiry, so the θ and θ' runs coincide until the first decision τ^* at which m expires under θ but not under θ' (if there is none, they coincide throughout). From τ^* on, m is dead in the θ run, because dead is absorbing, whatever the θ' run does later. Hence at every request time, m dead under θ' implies m dead under θ. With coupled rules or state-dependent requests, a different θ can change other entries or the requests, and the argument fails: take $\pi = 1$, $p = 0$, $\lambda = 0.5$, all scores 0.1, entries n and m ingested at 1 and 2, a query at 5 that requests n if n is not dead and m otherwise, and a query at 8 that requests m. With $\theta = 1$, n is dead at 5, the first query rescues m and the second succeeds; with $\theta = 3$ the first query requests n, m dies at 7 and the second fails. □

Proof of Theorem 4. A deterministic replay maps each conversation X_i to a loss vector over the grid, so i.i.d. conversations give i.i.d. losses whatever the policy. The bounds use only that and boundedness; neither M1, monotone coupling nor single attribution enters their validity (they enter only the one-pass computation, which Corollary 2 restores outside W_λ for entry-local rules). □

Proof of Corollary 2. Before its first rescue m follows the never-rescued history, so it is first demoted at $d_1 = \min B_\lambda(m)$ (with $\min \emptyset = \infty$) and is first rescued iff a request falls in $(d_1, \mathrm{end}(d_1)]$; this defines W_λ. Outside W_λ, m dies at $\mathrm{end}(d_1)$, so (q, m) is harmed iff $\mathrm{end}(d_1) < t_q$. Inside W_λ, replay the single-entry automaton (Proposition 1). The prefix minima of each entry's score give d_1 for every λ in one pass. □

Proof of Theorem 3. Proposition 1 holds by construction. An entry that is never rescued has the never-rescued history throughout, so it is demoted at the first eligible decision with $s^0_\tau(m) < \lambda$, i.e. at $\min B_\lambda(m)$ computed from s^0, and dies at $\mathrm{end}(\min B_\lambda(m))$: this is Lemma 1, and Theorem 2 and Corollary 2 use only it and $B_\lambda(m) \subseteq B_{\lambda'}(m)$, which holds for the fixed score s^0. For Theorem 1, if demotion after the latest insertion ι depends only on (τ, ι) and the stream, then after the last rescue r the entry is demoted at the first unmasked eligible decision whose post-r score is below λ, and the proofs of (a) and (b) go through with B_λ replaced by that post-rescue set. Proposition 3 compares the score at d^* in two runs with different insertion histories, which uses M1; the following example shows that without M1 a post-rescue score change is a second route. □

A single-entry example of the second route. Take $\pi = 1$, $p = 0$, $\theta = 1$ and the recency-only scorer $s_\tau(m) = 2^{-(\tau - \iota(m))}$, which ages from the latest insertion and is entry-local with $s^0_\tau(m) = 2^{-(\tau - a(m))}$. One entry m is ingested at $a(m) = 1$; query q_1 needs it at $t = 4$ and q_2 at $t = 8$. At $\lambda = 0.1$: s^0 is $0.5, 0.25, 0.125$ at decisions $2, 3, 4$ (kept), q_1 finds m live, $s^0_5 = 0.0625 < 0.1$ demotes it at 5, it is dead at 7, and q_2 fails. At $\lambda' = 0.3$: $s^0_3 = 0.25 < 0.3$ demotes m at 3; q_1 rescues it at 4 ($\iota = 4$); at 5 it is eligible and, with $s_5 = 2^{-1} = 0.5 \geq 0.3$, kept; it is demoted at 6 and still cold when q_2 arrives at 8, which succeeds. So $(q_2, m) \in H_\mathcal{R}(0.1) \setminus H_\mathcal{R}(0.3)$ with $m \in W_{\lambda'}$, as Theorem 3 requires, but the killing decision 5 lies outside the window $[4, 4]$ opened by the λ' rescue, so the protect window plays no part: under the M1 score $s^0_5 = 0.0625 < 0.3$ would demote m at 5, kill it at 7 and harm q_2. The violation exists only because the rescue raised the entry's own score.

Proof of Proposition 4. Take $\pi = 1$, $p = 1$, $\theta = 1$, capacity $K = 2$ (a decision fires when more than two entries are live), and three entries with constant scores: c ingested at $t = 1$ with score 0.10, b at $t = 2$ with score 0.50, a at $t = 3$ with score 0.03; one query needs a at $t_q = 8$. An entry is eligible at τ when $\tau - \iota > 1$, and a demotion at d expires at $d + 2$. At $\lambda = 0.05$: the decision at 3 fires (three live), c is the only eligible entry and $0.10 \geq 0.05$ keeps it; at 4 the decision fires again, c and b are eligible and kept; at 5 it fires, a is eligible, $0.03 < 0.05$, and a is demoted; a is dead at 7 and (q, a) fails at 8. At $\lambda' = 0.20$: the decision at 3 fires and demotes c ($0.10 < 0.20$); two entries remain live, so no later decision fires, a is never demoted, and (q, a) succeeds. Hence $(q, a) \in H_\mathcal{R}(0.05) \setminus H_\mathcal{R}(0.20)$ while a is never rescued at either threshold ($W = \emptyset$). No swap-in occurs anywhere in the construction, so the same trajectories arise under $\mathcal{D}$. For general $K \geq 2$, add $K - 2$ entries with score 1 ingested at $t = 1, \ldots, K - 2$ and shift every other time above by $K - 2$: the added entries are never demoted, so every live count exceeds the $K = 2$ count by exactly

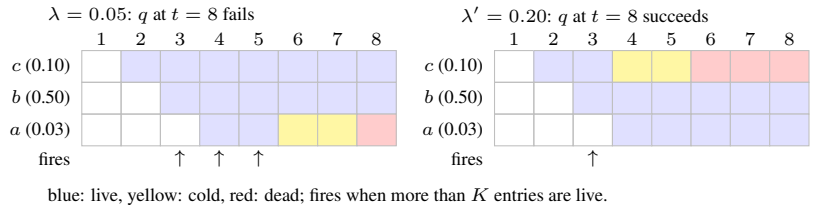


Figure 8: The three-entry stream of Prop. 4 at two thresholds ($\pi = p = \theta = 1$, $K = 2$); a decision fires when more than K entries are live and the query needs a. At λ', demoting c at 3 drops the live count to K, no later decision fires and a is never examined; at λ the decisions at 4 and 5 still fire, a is demoted at 5, and it is never rescued at either threshold.

$K - 2$, the same decisions fire, and the trajectories of a, b, c are unchanged. (Decisions continue on the clock after the last ingest.) □

Proof of Proposition 5. Expiry is checked at every decision whether or not it fires, so $\mathrm{end}(\cdot)$ is the same in both runs.

Let $(q, m) \in H_{\mathcal{R}}(\lambda) \setminus H_{\mathcal{R}}(\lambda')$ with $m \notin W_\lambda \cup W_{\lambda'}$. Never rescued, m has $\iota(m) = a(m)$ in both runs, so it is eligible at the same decisions in both. In the λ-run it is demoted at the first $d \in \mathrm{Fire}(\lambda)$ that is eligible with $s_d(m) < \lambda$, and harm means $\mathrm{end}(d) < t_q$. Suppose $d \in \mathrm{Fire}(\lambda')$. Then d is eligible, fires and has $s_d(m) < \lambda < \lambda'$ in the λ'-run, so m is demoted there at some $d' \le d$, is never rescued, and dies at $\mathrm{end}(d') \le \mathrm{end}(d) < t_q$: $(q, m) \in H(\lambda')$, a contradiction. Hence $d \in \mathrm{Fire}(\lambda) \setminus \mathrm{Fire}(\lambda')$ and $m \in T_{\lambda,\lambda'}$. In the construction of Proposition 4 (Fig. 8), $\mathrm{Fire}(0.05) = \{3, 4, 5\}$ and $\mathrm{Fire}(0.20) = \{3\}$, and a is first demoted at 5, so $a \in T$. □

C Theory boundaries: coupled rules and retrievers

This appendix measures what happens when the two locality conditions of §3 are relaxed (scope summarised in Table 1): design variants of the state machine, the exactness of per-entry replay, the capacity trigger, and real retrievers.

C.1 Design variants: do the choices of the swap mechanism matter?

Table 4; violation counts are also plotted in Fig. 3b. The state machine of §3 makes three choices that a system could make differently: a swap-in restarts the protect window; the scorer keeps ageing recency from the original ingest (M1); demotion is a threshold gate rather than a capacity rule. We run each alternative; the capacity alternative here is a *bottom-k* rule that demotes the lowest-scoring eligible entries until at most K are live, with no λ gate, and its monotonicity is checked over a grid of K. The capacity *trigger* of §5.1, which gates on λ, is measured separately in Appendix C.3. *No protect reset.* Every LoCoMo violation disappears ($7 \to 0$) while repeated demotion ($4.1 \to 4.5\%$), harm and sufficiency are unchanged, as Prop. 3 predicts for M1 scores. *Recency reset.* The rule breaks M1 but stays entry-local, so Theorem 3 still guarantees containment while allowing a second mechanism. Repeated demotion (4.1%) and sufficiency are unchanged; violations rise from 7 to 20, all inside W. *Bottom-k.* Entries are coupled and Proposition 1 fails. Demotions are fewer, sufficiency is higher (0.28; 0.54 on PerLTQA), and the 5 (LoCoMo) and 20 (PerLTQA) violations are again all inside W.

Across all eight variant runs no violation falls outside the re-inserted set; for bottom-k this is observed, not guaranteed.

C.2 Factorisation boundary: where does per-entry replay stop being exact?

Proposition 1 and Theorem 3 predict that a scorer reading only the entry's own state keeps the factorisation while a capacity trigger breaks it. Replaying each entry alone (all other entries frozen live) and comparing with the full replay: under the recency-resetting scorer the harmed-pair and re-inserted sets agree exactly on both corpora at $\lambda \in \{0.3, 0.6\}$ (0 symmetric difference on 112 and 353

Table 4: Design variants; LoCoMo $\theta = 100$, PerLTQA $\theta = 20$.

corpus	variant	≥ 2 demotions	harmed q.	witnesses	suff. last	violations (in W)
LoCoMo	baseline	4.1%	12.3%	354	0.121	7 (7)
LoCoMo	no-protect-reset	4.5%	12.5%	359	0.117	0 (0)
LoCoMo	recency-reset	4.1%	12.2%	353	0.119	20 (20)
LoCoMo	bottom-k (K=150)	2.7%	7.3%	192	0.281	5 (5)
PerLTQA	baseline	28.7%	1.4%	112	0.321	0 (0)
PerLTQA	no-protect-reset	33.4%	1.4%	113	0.327	0 (0)
PerLTQA	recency-reset	24.8%	1.4%	112	0.321	0 (0)
PerLTQA	bottom-k (K=30)	33.1%	0.3%	28	0.536	20 (20)

LoCoMo failures, 67 and 112 PerLTQA failures); under the bottom-k rule they do not: on LoCoMo per-entry replay reports 332 failures against the full replay's 192 (140 in the symmetric difference), and on PerLTQA per-entry replay reports 90 failures against the full replay's 28 (66 in the symmetric difference). Per-entry replay over-predicts failures there because a frozen population cannot displace the entry. Cluster-bootstrap intervals for the candidate and re-inserted sets: $|C_\lambda|/|\mathcal{M}| = 0.086$ $[0.064, 0.113]$ and $|W_\lambda|/|\mathcal{M}| = 0.044$ $[0.033, 0.058]$ on LoCoMo at $\lambda = 0.6$; 0.144 $[0.128, 0.160]$ and 0.094 $[0.082, 0.106]$ on PerLTQA (cluster means, bootstrap seed 3; the pooled $|C_\lambda|/|\mathcal{M}|$ plotted in Fig. 3c is 0.147). On ten LongMemEval streams (49,000 entries, $\theta = 250$, $\lambda = 0.3$) the full replay takes 1.1 s and per-entry replay on $C_\lambda \supseteq W_\lambda$ (0.1% of entries) 3 ms with identical output; the one-pass index build (62 s) dominates either. At these sizes Corollary 2 is therefore a structural statement (it names the only entries a threshold change can affect) rather than a speed-up.

C.3 The capacity trigger on the threshold grid

We implement the rule of §5.1: a decision fires only when more than K entries are live, and a firing decision demotes every eligible live entry below λ. For each adjacent pair on the 59-point grid we classify every violation by whether its entry is in $W_\lambda \cup W_{\lambda'}$, only in $T_{\lambda,\lambda'}$, or in neither (Table 5; oracle requests, LoCoMo $\theta = 100$, PerLTQA $\theta = 20$). No violation lies outside $W \cup T$, as Prop. 5 requires. Unlike the threshold gate, violations do escape W, and increasingly so as the trigger fires less often: the rule approaches the threshold gate, and T empties, when every decision fires. Coupling also multiplies violations (29–156 against 7 for the gate on LoCoMo) and makes the localization set larger: $|T|/|\mathcal{M}|$ averages 2–7% and reaches 38% for some threshold pairs. PerLTQA has no violation at any K, as under the gate.

Table 5: Capacity trigger on LoCoMo ($\theta = 100$): monotonicity violations over the 59-point λ grid by location; $|T|/|\mathcal{M}|$ is averaged (maximum) over adjacent threshold pairs.

K	decisions firing	violations	in W	only in T	in neither	$\lvert T\rvert/\lvert\mathcal{M}\rvert$
60	87%	29	29	0	0	1.7% (14.5%)
100	70%	127	125	2	0	5.6% (33.8%)
150	56%	156	148	8	0	7.2% (37.5%)
250	36%	132	96	36	0	7.0% (26.0%)

C.4 When does a retriever couple entries?

With exogenous requests the results of §5 hold verbatim for the failed-request set $J_{\mathcal{R}}$ with E_q replaced by A_q. The BM25 retriever of Appendix D.10 ranks only non-dead entries, so its requests depend on which other entries have expired. We compare it with a *tier-independent* variant that ranks every ingested key (a tombstone keeps the key; a fetched expired entry is an observable failure rather than a swap-in).

- *Exactness.* Under tier-independent retrieval, replaying each entry alone with A_q as its requests reproduces the stateful simulation's failed requests and re-inserted set exactly at all five tested (λ, θ).
- *Cost.* The two retrievers give the same storage (0.7426, 0.8135, 0.7428 for $\mathcal{D}_{0.49}$, $\mathcal{R}_{10}$ at 0.74, $\mathcal{R}_5$ at 0.84), the same gold deaths (258, 191, 292) and swap-ins within one. The

Table 6: Structural census (reuse = gold pairs per needed entry; load-bearing = needed / all entries; density = queries / entries); the last two rows are the record-level and utterance-level re-granulations of PerLTQA. LongMemEval's 50 units are constructed pseudo-clusters, not established independent conversations; reuse is 1 by construction because each compiled history serves one question. Re-granulating PerLTQA to utterances moves density only.

Corpus	units	entries	load-bearing	reuse [95% CI]	density
LongMemEval-S	50	245,789	0.0036	1.000 [1.000, 1.000]	0.0019
LoCoMo	10	5,882	0.242	1.648 [1.586, 1.701]	0.260
PerLTQA (records)	31	2,079	0.871	4.536	3.95
PerLTQA (utterance)	31	12,265	–	4.555 [4.379, 4.761]	0.678

storage drift of Appendix D.10 is therefore caused by non-gold requests, not by coupling, and a storage match must be made under the deployed request policy.

- *Observability.* The state-dependent retriever never requests an expired entry, so it records no failure at all; the tier-independent one records 700, 360 and 868 failed requests at the same three arms.
- *Monotonicity.* Over the 59-point grid, the failed-request set has 48, 8 and 6 violations at $\theta = 5, 10, 20$, all in W, and gold harm has 8, 6 and 3, all in W, as Theorem 2 guarantees for any request policy.

Ranking over the full key index thus restores both the guarantees and the observability at no measurable cost in storage or evidence loss.

C.5 Deployed-style paging mechanisms

Two capacity-driven rules model deployed memories: *overflow paging* (MemGPT-like), which pages the oldest-ingested eligible live entries to the cold tier whenever more than K are live, and *LRU tiering*, which demotes the least recently used ones (every request refreshes recency). Both recall cold entries on request and delete them after θ; both couple entries, so no guarantee applies. Sweeping the capacity K (more capacity should not add harm), every observed violation lies in the re-inserted set: $5/5$ and $3/3$ for overflow paging on LoCoMo ($\theta = 100, 250$), 0 for LRU, and $93/93$, $23/23$ (paging) and $11/11$, $3/3$ (LRU) on PerLTQA ($\theta = 10, 20$). At the middle capacity, blaming the last demotion repairs 11–15% of LoCoMo and 44–48% of PerLTQA failures, as under the threshold gate, and 2–5% (LoCoMo) and 31–37% (PerLTQA) of demoted entries are demoted at least twice. The structural picture thus carries over to deployed-style rules.

D Additional replay experiments

Table 7: Scorer ablation at compression 0.5; LoCoMo $\theta = 100$, PerLTQA $\theta = 20$.

corpus	scorer	λ	≥ 2 demotions	witnesses	suff. first	suff. last	violations (in W)
LoCoMo	fusion	0.35	5.4%	181	0.249	0.249	7 (7)
LoCoMo	recency	0.15	4.8%	96	0.146	0.156	7 (7)
LoCoMo	frequency	0.70	2.6%	193	0.275	0.275	0 (0)
LoCoMo	semantic	0.05	4.0%	174	0.339	0.368	2 (2)
LoCoMo	rec+freq	0.55	5.2%	157	0.229	0.242	2 (2)
PerLTQA	fusion	0.70	44.0%	138	0.370	0.370	0 (0)
PerLTQA	recency	0.85	39.8%	58	0.483	0.517	0 (0)
PerLTQA	frequency	0.95	44.6%	165	0.297	0.309	0 (0)
PerLTQA	semantic	0.45	41.6%	134	0.448	0.455	0 (0)
PerLTQA	rec+freq	0.95	57.4%	184	0.277	0.310	5 (5)

D.1 Scorer ablation

Table 7 (violations plotted in Fig. 3a), and, for the 59-point grid counts, Table 8. We repeat the measurements of §8.1 and §8.2 with five policy-independent scorers (the fusion scorer; recency only;

Table 8: Monotonicity violations on the 59-point grid at (query, entry) granularity (harmed pairs: distinct pairs harmed at some threshold of the grid, hence more than the 354 witnesses at $\lambda = 0.6$), conversation-cluster bootstrap intervals for the count, and how many involve a re-inserted entry (Theorem 2).

corpus	θ	harmed pairs	violations	cluster 95% CI	in W
LoCoMo native	100	359	7	[3,9]	7/7
LoCoMo native	250	182	2	[2,2]	2/2
LoCoMo native	∞	0	0	[0,0]	0/0
PerLTQA uniform	20	112	0	[0,0]	0/0
PerLTQA uniform	50	6	0	[0,0]	0/0
PerLTQA uniform	∞	0	0	[0,0]	0/0

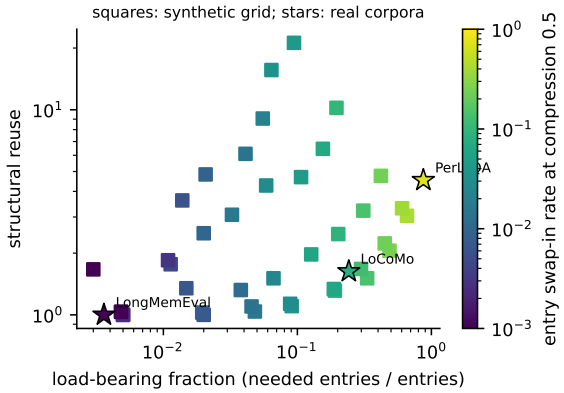


Figure 9: Entry swap-in rate at compression 0.5 over the (load-bearing, reuse) plane; squares synthetic, stars real.

re-mention frequency only; semantic overlap only; recency+frequency), each at the smallest coarse-grid λ with compression at least 0.5 (actual 0.52–0.67 on LoCoMo, 0.57–0.73 on PerLTQA). Under every scorer the picture is the same: repeated demotion (2.6–5.4% of demoted entries on LoCoMo, 40–57% on PerLTQA), counterfactual sufficiency of the blamed demotion between 0.15 and 0.52, and every monotonicity violation inside W (7/7, 7/7, 0, 2/2, 2/2 on LoCoMo; $0, 0, 0, 0, 5/5$ on PerLTQA). Scorers differ in how long exile runs are, not in whether a single decision is a cause. (Frequency-only cannot compress PerLTQA past 0.23; its row is at that compression.)

D.2 Synthetic plane: does it interpolate the real corpora?

Fig. 9. Synthetic streams with a density knob and a Zipf reuse knob (10 clusters each, 600 turns, random vocabulary so the semantic term is noise) cover load-bearing 0.003–0.66 and reuse 1–21. At matched compression 0.5 the three real corpora sit on the synthetic surface (0.0005, 0.073, 0.70); the plane also covers reuse above 4.5, which no annotated corpus reaches.

D.3 Cold lifetime: coverage and TTL-cache theory

Fig. 10. The lag from demotion to the next request has median 115–219 turns on LoCoMo across the threshold grid (122 at $\lambda = 0.3$), and serving 90% of requests needs θ between 250 (71%) and 500 (98%) at $\lambda = 0.3$ (the 90th-percentile lag is 377 turns), comparable to a whole conversation (369–689 turns, mean 588); on LongMemEval $\theta = 2{,}500$ serves 89% of a 4,900-turn stream. No short cold lifetime saves storage without forfeiting most swap-ins, so the cold lifetime is not a cost lever; its cost is the cold-tier storage of Appendix D.4, and the tombstone's role is to make failures observable for §7.

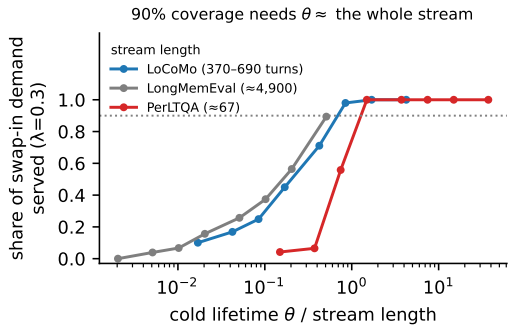


Figure 10: Share of swap-in demand served as a function of cold lifetime relative to stream length.

TTL-cache theory. A cold lifetime θ plays the role of a cache TTL. Under the classical Poisson model (Che et al., 2002; Choungmo Fofack et al., 2014), an entry requested at rate r is rescued within θ with probability $1 - e^{-r\theta}$; conditioning on a later request within the remaining stream R gives $(1 - e^{-r\min(\theta,R)})/(1 - e^{-rR})$. We estimate r as the entry's number of requests over its time in the stream and compare with the empirical share of first demotions of needed entries whose next request falls within θ, for θ from 2% to 100% of the stream. Under PerLTQA's uniform random schedule the conditional model is accurate (mean absolute gap 0.018 and 0.029 at $\lambda = 0.3, 0.6$); under LoCoMo's interleaved schedule, where queries cluster after their evidence sessions, it misses by 0.057 and 0.064, and the unconditional model by up to 0.19 on PerLTQA. Classical TTL analysis therefore transfers to reversible memory when requests are schedule-free, and the query schedule is what breaks it, as in §6.

D.4 Harm–storage frontier: is the cold tier worth its storage?

Fig. 11, Table 9. Storage is the live-plus-cold entry count summed over decision checkpoints, relative to keep-all (dead entries and tombstone keys excluded; unit weight per entry, so a proxy for bytes or tokens). We sweep λ in steps of 0.01 for $\mathcal{D}$ and for $\mathcal{R}_\theta$ ($\theta \in \{50, 100, 250, 500\}$ on LoCoMo, 380 points; $\{10, 20, 35, 50\}$ on PerLTQA, 455 points) and pair each Reprieve point with the destructive point of nearest storage (both in $(0.10, 0.95)$, gap ≤ 0.01).

Table 9: Storage-only matches (harm in %, difference Reprieve minus destructive with 95% cluster interval). A and B are the end-task pairs of §8.4.

corpus	match	$\mathcal{D}$	$\mathcal{R}_\theta$	storage	harm $\mathcal{D}$ / $\mathcal{R}$ (diff.)
LoCoMo	A	$\lambda = 0.23$	$\mathcal{R}_{50}, 0.27$	0.775	4.96 / 4.77 (-0.20 $[-1.38, +1.13]$)
LoCoMo	B	0.30	$\mathcal{R}_{100}, 0.40$	0.597	10.52 / 9.99 (-0.52 $[-1.42, +0.51]$)
LoCoMo	C	0.35	$\mathcal{R}_{50}, 0.44$	0.433	13.65 / 12.08
PerLTQA	exact	0.49	$\mathcal{R}_{10}, 0.74$	0.7426	3.14 / 4.30

On LoCoMo 93 of 169 near-matches favour Reprieve, by 0.07–2.2 points, but not all ($\mathcal{D}_{0.25}$ beats $\mathcal{R}_{250}$ at 0.60 by 1.37 at storage 0.746). On PerLTQA the sign reverses on every band ($+0.17$ to $+1.17$ points against Reprieve at band representatives from 0.70 to 0.95 storage; $\theta \geq 35$ keeps ≥ 0.95 storage and has no match). Holding storage equal forces Reprieve to demote more, and the extra demoted entries are lost when their next request falls after θ, which is most of them on both corpora (Appendix D.3); with a load-bearing fraction of 0.87 PerLTQA pays for it, and at its exact match Reprieve is 13.1 strict-accuracy points lower on divergent questions (Appendix F.1).

Differential pricing. The frontier charges a cold entry like a live one, although cold storage is usually cheaper. Let a cold entry cost c times a live one and compare the best harm reachable at each budget from 0.30 to 0.95 (Table 10). On LoCoMo Reprieve is never worse than $\mathcal{D}$ for $c \leq 0.85$; on PerLTQA, where it is worse at $c = 1$, for $c \leq 0.75$. At $c = 0.25$ it is 9.0 (LoCoMo) and 11.2 (PerLTQA) points better on average. The no-gain result above is thus specific to equal pricing. Script `run_p2_cold_pricing.py`.

Table 10: Reprieve minus destructive harm (points), averaged over storage budgets 0.30–0.95, with the share of budgets where Reprieve is worse, as a function of the cold-to-live price ratio c.

c	1	0.9	0.85	0.75	0.5	0.25
LoCoMo	−0.95 (6%)	−1.48 (2%)	−1.88 (0%)	−2.94 (0%)	−6.11 (0%)	−8.98 (0%)
PerLTQA	+0.51 (64%)	−0.25 (34%)	−0.94 (19%)	−2.49 (0%)	−6.63 (0%)	−11.16 (0%)

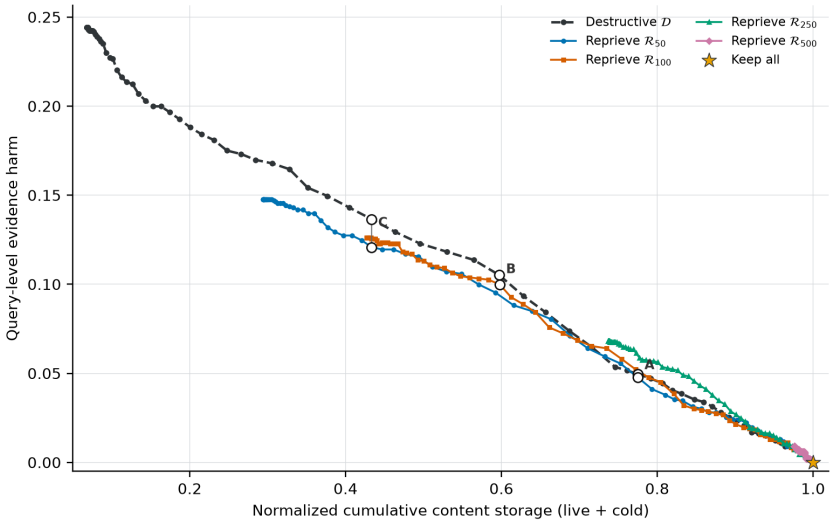


Figure 11: Harm–storage frontier on LoCoMo, dense sweep: destructive eviction (dashed) versus Reprieve semantics at four cold lifetimes; matches A–C are storage-only (A and B are the end-task pairs of §8.4). Storage is the cumulative live-plus-cold entry count relative to keep-all.

D.5 Attribution as a diagnostic signal

A reversible store observes its failures (tombstones) and can log each entry's scores, so after a run it can compute, for every witness, the decisions a rule blames and keep the entry live there. We suppress the blamed decisions of all witnesses at once, replay, and count repaired witnesses, new failures, the number of suppressed (entry, decision) pairs (the budget), and the change in normalized storage (Table 11). We compare the three single-decision rules, the exile run, the exile run restricted to the budget of blaming the last demotion (cheapest runs first), and a global baseline that lowers λ on the 0.01 grid until storage rises at least as much as under full run repair. No repair created a new failure. Appendix D.6 applies the diagnosis prospectively. At equal budget the run repairs 1.7–

Table 11: Budgeted repair of observed failures: share of witnesses repaired (budget = suppressed pairs; Δ storage in points). "Run @ last" uses at most the budget of "last". The global baseline lowers λ until storage rises at least as much as under full run repair and reports the reduction in failure pairs.

setting (θ, λ)	witnesses	first	last	run @ last	run (budget, Δst.)	global λ (λ', Δst.)
LoCoMo 100, 0.3	114	31.6%	33.3%	58.8%	100% (460, +1.4)	11.4% (0.29, +1.7)
LoCoMo 100, 0.6	354	10.5%	12.1%	37.3%	100% (2,347, +4.7)	9.0% (0.50, +4.8)
LoCoMo 250, 0.6	180	12.8%	16.7%	47.2%	100% (842, +2.0)	7.8% (0.51, +2.3)
PerLTQA 20, 0.3	67	37.3%	37.3%	62.7%	100% (179, +2.7)	59.7% (0.26, +4.0)
PerLTQA 20, 0.6	112	32.1%	32.1%	64.3%	100% (293, +4.3)	61.6% (0.27, +6.0)

3.1× more witnesses than the last demotion, because a single suppression only shifts the removal to the next decision in the run while a short run is repaired outright. Global threshold changes spend storage on entries that never fail: on LoCoMo, the storage that repairs every run removes only 8–11% of failures when spent uniformly. On PerLTQA, where most entries are load-bearing, the global baseline is competitive but still needs 1.4–1.5× the storage.

Is the unit problem specific to reversibility? Corollary 1 predicts that a single removal is also insufficient under $\mathcal{D}$ whenever the score stays below λ at a later eligible decision. On the same streams, suppressing a failure's only removal under $\mathcal{D}$ repairs 39.0% ($\lambda = 0.3$) and 21.8% (0.6) of LoCoMo failures and 28.3% and 35.7% of PerLTQA failures, against 12.1–33.3% and 32.1–37.3% for blaming the last demotion under $\mathcal{R}_\theta$. The single-decision problem is therefore a property of any repeated threshold check. Reversibility changes where the run starts: after the last rescue, which precedes 7.0–13.0% of LoCoMo failures and none on PerLTQA, and there it lowers single-decision sufficiency further (LoCoMo, $\lambda = 0.6$: 12.1% vs 21.8%). Script `run_p2_destructive_attribution.py`.

D.6 Prospective repair calibrated by exile runs

A deployed system cannot undo a failure, but it can use diagnosed runs to change its rule. We use the renewal rule *request-renewed protection*: every request for m, a rescue or a hit on a live entry, sets $\iota(m) = t_q$, and an entry requested at least once is protected for $p + k\pi$ after its latest request (never-requested entries keep p). It is entry-local, so Theorem 3 applies, and its exile run starts at the latest request r': a witness is repaired by any $k \geq (\max \rho' - r' - p)/\pi$. For each held-out conversation we set k to the 80th percentile of this bound over the observed failures of all *other* conversations, so the evaluated conversation contributes nothing to calibration; the single-decision view corresponds to $k = 1$. Two global baselines, a lower λ and a longer θ, are traced as full harm–storage curves and read at the rule's storage (Table 12; oracle requests). On LoCoMo, where 54–60% of failing entries were requested before they failed, the run-calibrated rule removes 41–48% of failures, 21–32 points more than a lower λ at the same storage (conversation-cluster intervals exclude zero), while $k = 1$ removes 7–15%. On PerLTQA only 1.5–5.4% of failing entries were requested before, calibration finds no run to cover ($k = 0$), and the rule does no better than the baselines: runs there start at ingestion, where no request-triggered rule can act.

Table 12: Prospective repair, leave-one-conversation-out. Failure reduction relative to the unmodified policy; storage change in points. "Same storage" reads the global curves at the run rule's storage; the last column is the run rule's extra reduction over a lower λ with a 95% conversation-cluster interval.

setting (θ, λ)	failures	k (folds)	run rule	$k = 1$	lower λ	longer θ	run − lower λ
LoCoMo 100, 0.3	114	11–14	41% (+3.3)	7%	20%	15%	21 [12, 31]
LoCoMo 100, 0.6	354	10–11	48% (+9.0)	8%	15%	16%	32 [27, 37]
LoCoMo 250, 0.6	180	6–8	43% (+3.5)	15%	17%	16%	26 [20, 36]
PerLTQA 20, 0.3	67	0–0	0% (+0.2)	0%	3%	2%	–
PerLTQA 20, 0.6	112	0–0	4% (+0.4)	4%	5%	3%	–

Rule class, calibration and tuning. Table 13 reads the three families at common storage increments. Up to $+4$ points the renewal rule removes 1.5–$7\times$ as many failures as a lower λ or a longer θ; it saturates (all failures it can reach are repaired), and at $\theta = 100$, $\lambda = 0.3$ a lower λ has overtaken it by $+6$ points. Moving the calibration quantile from 0.8 to 0.9 changes the reduction by at most 6 points. Choosing k on the other conversations by harm plus storage, which needs harm labels, is far worse at $(100, 0.3)$ (10% against 41%) but 1 and 8 points better in the other two settings; under BM25 it picks $k = 0$ (Table 14).

Table 13: LoCoMo, oracle requests: failure reduction (%) of request-renewed protection / lower λ / longer θ at equal storage increments; run calibration at quantiles 0.5, 0.8, 0.9; k tuned on the other conversations' harm plus storage (storage change in points).

(θ, λ)	$+2$ points	$+4$ points	$+8$ points	runs ($q = 0.5, 0.8, 0.9$)	tuned on harm
100, 0.3	20 / 13 / 9	41 / 25 / 17	41 / 52 / 35	27, 41, 41	10 (+1.3)
100, 0.6	9 / 1 / 4	18 / 5 / 8	40 / 14 / 14	34, 48, 51	49 (+9.3)
250, 0.6	22 / 8 / 9	49 / 19 / 17	52 / 34 / 32	33, 43, 49	51 (+4.1)

A real retriever on LoCoMo. With BM25 requests over every ingested key (top 10; exogenous), which place 43.8% of gold entries in the request set, gold-harm violations stay in W ($7/7$ and $3/3$ at

$\theta = 100, 250$) and blaming the last demotion repairs 17.5% of failures. Retrieval-triggered swap-ins already rescue many entries (baseline failures fall from 114, 354, 180 to 81, 251, 80), and the run-calibrated rule removes 62–63% of the remaining ones against 48–53% for a lower λ at the same storage: the gain persists but narrows to 10–15 points.

D.7 Access-aware cache baselines

The renewal rule is a pin-after-access policy: a requested entry is protected for $p + k\pi$ after its latest request. We compare it at equal storage with standard access-aware policies on the same streams (Table 14): LRU-style recency reset (a request resets the entry's recency feature, so its score ages from the latest access; λ swept), an LFU-style frequency boost (each request adds δ to the entry's score; δ swept at the baseline λ), the best fixed horizon (the whole $k = 0, \ldots, 12$ frontier), and k tuned leave-one-conversation-out on the other conversations' harm rate plus storage. Each family is read at the storage of the run-calibrated rule.

Table 14: Failure reduction (%) at the storage of the run-calibrated rule (its storage change in points in parentheses). Frequency boost and best fixed k are frontiers tuned in hindsight; "k tuned on harm" is chosen on the other conversations (its own storage change in parentheses).

requests	(θ, λ)	failures	run rule	lower λ	LRU reset	freq. boost	best fixed k	k tuned on harm
oracle	100, 0.3	114	41 (+3.2)	19	25	46	41	10 (+1.3)
oracle	100, 0.6	354	48 (+9.0)	15	18	51	48	49 (+9.3)
oracle	250, 0.6	180	43 (+3.5)	17	20	45	44	51 (+4.1)
BM25	100, 0.3	81	63 (+7.4)	49	53	63	63	2 (+0.5)
BM25	100, 0.6	251	62 (+19.0)	53	55	66	62	4 (+1.2)
BM25	250, 0.6	80	62 (+4.9)	48	44	60	63	5 (+0.3)

The run-calibrated rule lands on the frontier of fixed horizons (within one point) and beats LRU recency reset by 16–29 points under oracle requests and 7–18 under BM25. A frequency boost tuned in hindsight matches it (-2 to $+4$ points): the gain belongs to access-aware retention in general, not to this rule. What the runs add is the horizon without harm labels: choosing k from the other conversations' harm and storage picks $k = 0$ under BM25 (2–5%). On PerLTQA every access-aware policy gains at most 5%, as its failures precede any request. Script `run_p2_access_baselines.py`.

D.8 A learned retention scorer

We replace the fusion heuristic by a learned scorer: a logistic regression on semantic overlap, recency and re-mention frequency at each (entry, decision), trained to predict whether some later query needs the entry. Each conversation is scored by a model fitted on the other conversations only, and the features use the stream up to the decision, so the scorer satisfies M1. It is no better than the heuristic at matched storage (LoCoMo harm 0.097, 0.076, 0.054 at storage 0.6, 0.7, 0.8, against 0.099, 0.068, 0.046), but the structural picture is unchanged. At compression 0.5, blaming the first, last or all demotions repairs 15–16% of LoCoMo and 34% of PerLTQA failures; exile runs average 5.0 and 2.4 decisions, and 15% and 32% of witnesses have $|\rho| = 1$. On a 59-point grid of learned-score quantiles every violation lies in W ($3/3$ and $8/8$). The prospective rule of Appendix D.6 removes 43% of LoCoMo failures, against 16% for a lower λ at the same storage, and again does not help on PerLTQA (6% against 38%).

D.9 Robustness checks from the frozen replay ledger

Table 15 varies one protocol component at a time (violation counts with cluster intervals are in Table 8).

D.10 Retrieval sensitivity

We replace oracle requests with deterministic BM25 (Okapi, $k_1 = 1.2$, $b = 0.75$) top-10 retrieval; every cold entry in the top 10 is swapped in, and the top five entries are shown to the reader. At the frozen PerLTQA destructive point ($\lambda = 0.49$) gold evidence is dead for 258 of the $8{,}214$ queries and there are no swap-ins. Reprieve at $\lambda = 0.74$, $\theta = 10$ performs $2{,}392$ swap-ins, $2{,}005$ of them entries

Table 15: Robustness checks, one protocol component varied at a time. Token storage uses a proxy tokenizer.

Check	Perturbation	Result
Additional scorers	Retrieval-feedback; store-normalized; dense embedding	The entry-local retrieval-feedback and dense-embedding scorers give LoCoMo $2/2$ and $8/8$, and PerLTQA 0 and $21/21$, violations in W. The coupled store-normalized scorer gives $8/8$ and $23/23$; only this last containment is observed rather than guaranteed by Theorem 3.
PerLTQA schedule	Uniform schedules, seeds 1–5	Destructive storage is 0.7426 and harm 0.0307–0.0324; Reprieve storage is 0.7381–0.7427 and harm 0.0430–0.0489. Violation counts are $0, 1, 0, 1, 0$, with both violations in W.
Re-inserted-set size	Compression targets $0.5/0.8$	$\lvert W\rvert/\lvert\mathcal{M}\rvert$ is $2.6/3.2\%$ on LoCoMo, $0.03/0.03\%$ on LongMemEval, and $34.1/56.3\%$ on PerLTQA. These are sizes of W, the set Corollary 2 replays; Fig. 3c also shows the superset C_λ used in our replays.
Decision clock	$(\pi, p) \in \{(20, 15), (25, 20), (30, 25)\}$	LoCoMo violation counts are $4, 7, 5$ at $\theta = 100$ and $2, 2, 5$ at $\theta = 250$; every observed violation is in W.
Token footprint	chars/4 and word-count proxies	At the frozen entry-storage matches, Reprieve minus destructive proxy-token storage is -1.28, -4.81, and -3.31 points on LoCoMo and -3.76 on PerLTQA.

Table 16: Cold-lifetime selection on real corpora; median selected θ over splits; "finite θ" is the share of splits selecting a lifetime shorter than ∞. LoCoMo and PerLTQA use native independent units. LongMemEval rows use pseudo-clusters with shared sessions and are superseded by Appendix E.3.

corpus	n (cal)	α	bound	median θ	finite θ	held-out risk (mean / p90)	coverage
PerLTQA	31 (19)	0.02	betting	∞	0%	0.000 / 0.000	1.00
PerLTQA	31 (19)	0.02	Hoeffding	∞	0%	0.000 / 0.000	1.00
PerLTQA	31 (19)	0.05	betting	∞	0%	0.000 / 0.000	1.00
PerLTQA	31 (19)	0.05	Hoeffding	∞	0%	0.000 / 0.000	1.00
PerLTQA	31 (19)	0.1	betting	∞	0%	0.000 / 0.000	1.00
PerLTQA	31 (19)	0.1	Hoeffding	∞	0%	0.000 / 0.000	1.00
PerLTQA	31 (19)	0.2	betting	10	100%	0.051 / 0.059	1.00
PerLTQA	31 (19)	0.2	Hoeffding	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.02	betting	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.02	Hoeffding	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.05	betting	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.05	Hoeffding	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.1	betting	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.1	Hoeffding	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.2	betting	∞	0%	0.000 / 0.000	1.00
LoCoMo	10 (6)	0.2	Hoeffding	∞	0%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.02	betting	∞	0%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.02	Hoeffding	∞	0%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.05	betting	∞	0%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.05	Hoeffding	∞	0%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.1	betting	5000	100%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.1	Hoeffding	∞	0%	0.000 / 0.000	1.00
LongMemEval	50 (30)	0.2	betting	2000	100%	0.073 / 0.152	1.00
LongMemEval	50 (30)	0.2	Hoeffding	5000	100%	0.000 / 0.000	1.00

that are not gold evidence for the query; its storage rises from 0.743 to 0.814 and its gold-death count falls to 191 (2.33%, against 4.30% under oracle requests). Oracle harm is therefore neither an upper nor a lower bound on harm under a ranking retriever: spurious swap-ins rescue evidence and consume storage at the same time. Re-pairing by storage alone selects $\lambda = 0.84$, $\theta = 5$ (0.74282 versus 0.74261; 3,084 swap-ins, 2,653 non-gold; gold dead for 292 queries). On the 237 questions of the oracle difference set, Recall@5 of the gold entry is 54.9% for this point, 49.8% for destructive eviction (cluster interval for the difference $[-2.5, +19.5]$ points) and 74.3% under keep-all, so the retriever bounds a large part of what any policy can deliver. Across retrieval-triggered threshold sweeps, all observed violations remain in W ($8/8$, $6/6$, and $3/3$ for $\theta = 5, 10, 20$; $9/9$, $5/5$, $2/2$ when swap-in is triggered on use rather than on retrieval), as Theorem 2 guarantees for any request policy, although this retriever couples entries through the store. The answer-level consequences are in Appendix F.1.

Table 17: The same procedure on synthetic streams (density 0.25, Zipf 1.0) with $n = 50, 100, 200$ independent conversations: the certificate starts to bite at $\alpha = 0.1$ at $n = 50$ and $\alpha = 0.05$ at $n = 100$ (and, in an untabulated run, $\alpha = 0.03$ at $n = 200$), while held-out coverage stays at or above $1 - \delta$.

n (cal)	α	median θ	finite θ	held-out risk (mean / p90)	coverage
50 (30)	0.02	∞	0%	0.000 / 0.000	1.00
50 (30)	0.05	∞	0%	0.000 / 0.000	1.00
50 (30)	0.1	400	100%	0.002 / 0.003	1.00
100 (60)	0.02	∞	0%	0.000 / 0.000	1.00
100 (60)	0.05	600	100%	0.000 / 0.000	1.00
100 (60)	0.1	250	100%	0.032 / 0.034	1.00
200 (120)	0.02	∞	0%	0.000 / 0.000	1.00
200 (120)	0.05	400	100%	0.002 / 0.002	1.00
200 (120)	0.1	250	100%	0.032 / 0.034	1.00

E CERTIFICATES AT REAL AND SYNTHETIC SAMPLE SIZES

E.1 STORAGE AT THE CERTIFIED LIFETIME

At the λ used for certification, the certified lifetime keeps most of the keep-all storage when the target is tight: on the content-disjoint LongMemEval units $\theta = 250$ ($\alpha \leq 0.1$) keeps 99.2% of storage (harm 0.2%), and on PerLTQA $\theta = 50$ ($\alpha = 0.15$) keeps 99.3%; only the loose target $\alpha = 0.2$ selects $\theta = 10$ on PerLTQA, which keeps 72.6% (harm 5.1%). Destructive eviction at the same λ keeps 25–28% with harm near 48%. The certificate is therefore a tool for auditing the risk of a chosen configuration, and the cheap cold tiers of Appendix D.4 are not certified at these sample sizes.

E.2 REAL CORPORA AND SYNTHETIC STREAMS

On PerLTQA (31 character streams), the betting bound selects finite θ in every split only at $\alpha = 0.2$ among the tabulated targets (and at $\alpha = 0.15$ in a finer grid); on LoCoMo (10 native conversations), no finite lifetime is selected at the reported targets. Selected lifetimes on the real corpora are in Table 16; synthetic streams are in Table 17.

E.3 CERTIFICATES ON CONTENT-DISJOINT LONGMEMEVAL UNITS

The 50 LongMemEval pseudo-clusters of the main protocol share filler sessions and are not independent. We hash every session's content and remove each session whose content occurs in more than one question's haystack (4,126 contents), drop the 38 questions whose answer sessions are shared, and keep the 442 question histories with a valid query (≈30 private sessions, ≈307 turns each on average). No session id and no session content occurs in two units (asserted in the script), so the units are as independent as the original questions; they form a modified, shorter benchmark. With $\lambda = 0.55$ (destructive compression 0.82) and otherwise the protocol of §8.4 (fixed sequence from $\theta = \infty$ down over 14 values, $\delta = 0.1$, 200 splits of $265/177$ units), the betting bound selects a finite θ in every split for every $\alpha \geq 0.02$: median $\theta = 250$, held-out risk 0.002 and coverage 1.00 for $\alpha \leq 0.05$; Hoeffding needs $\alpha = 0.075$. Grouping the same histories into 93 five-question units (≈1,530 turns) certifies at $\alpha = 0.1$ (median $\theta = 1{,}000$, held-out risk 0.032) but not at $\alpha \leq 0.05$, again showing that resolution follows the number of independent units.

F END-TASK EVALUATION PROTOCOL

Sampling. Ten development questions (one per conversation) were used to choose the reader, the judge and the output length and are excluded from every reported number. The held-out sample of 100 questions was drawn with a fixed seed, stratified ten per conversation and 25 per LoCoMo question category 1–4 (category 5, adversarial, is outside harm attribution). The diagnostic set is the union of the questions on which the two policies of pair A (33) or pair B (66) differ in whether every gold entry is alive at query time, minus one development question (83; 15 questions belong to

both pairs). Membership was computed from the replay before any answer existed; the 8 diagnostic questions already in the held-out sample reuse its answers and judgments by identifier.

Arms and prompts. For every question and setting the replay yields the store at query time; cold gold entries are swapped in by identifier. The reader sees the retained entries and the question and nothing that names the setting (an audit found no policy word in any prompt). Reader: `gemini-2.5-flash`, temperature 0, 1,024 output tokens (512 truncated visibly during development). Rubrics: `gpt-5-mini-2025-08-07` produced necessary answer points from the question, reference and evidence; every set was audited against those sources only (rules: drop points that narrate provenance, merge derivation chains into the conclusion they support, drop facts absent from both reference and evidence, restore omitted reference items, strip quotations) and frozen with a hash before any answer was judged. Judge: the same model, reasoning effort minimal, sees the question, the frozen points and one candidate answer; it does not see the setting, the reference, the evidence or other candidates. Each point is marked covered or not with an exact quotation; unsupported substantive additions and ambiguity are flagged. Strict accuracy is 1 only when every point is covered, no substantive unsupported addition is flagged and the answer is not ambiguous.

Execution. All 500 held-out and 279 diagnostic answers and judgments were obtained (28 diagnostic ones reused by identifier). Provider failures were retried by identifier and truncated answers regenerated under a rule fixed in advance; judge outputs that violated the JSON schema (62) were repaired by a deterministic script that changed no decision. One ambiguous diagnostic judgment is excluded from Table 19. Cost: $12.55. Logs of every retry and repair are in the supplementary release.

Review of rubrics and judgments. Every frozen rubric (100 held-out, 75 diagnostic questions) was audited against the question, reference answer and cited evidence only, before any answer was judged, by an AI coding assistant applying the fixed correction rules (59 and 33 rubrics corrected); a fixed-seed blind sample of 20 and 18 rubrics was re-checked. A fixed-seed blind sample of 20 held-out and 18 diagnostic judgments was re-checked with the setting hidden: all 38 strict labels agreed; point decisions disagreed or were borderline on 2 and 1 rows and the unsupported-addition flag on 2 and 4 rows, none changing a strict label. These audits were performed by an AI assistant, not by independent human annotators; all decisions and corrections are released.

Data quality. Four LoCoMo reference answers disagree with the conversation or with their own gold evidence: `conv-48:q87` (reference "Thailand, Brazil"; evidence supports Bali), `conv-49:q37` (two of six listed dishes never appear), `conv-42:q12` and `conv-44:q33` (open-domain references with no support in the conversation). Scoring followed the frozen reference in all four.

Held-out sample. On 100 held-out questions (500 answers; Table 18) strict accuracy is 56% under keep-all and 50–56% under the compressed settings. Both matched pairs give small positive point estimates for Reprieve (+4.0 and +1.0 points) whose conversation-cluster intervals include zero. This is in line with the offline harm differences of 0.2 and 0.5 points (Appendix D.4). A random sample does not resolve harm rates of 5–13%: the two destructive arms differ by five points in the direction opposite to their harm.

Reader bottleneck. On 67% (pair A) and 52% (pair B) of divergent questions the arm holding every gold entry is still wrong (Table 19), so on most questions the reader, not the store, is the binding constraint; conversely, 5 (pair A) and 14 (pair B) questions are answered correctly with a needed entry dead, from redundant context or parametric knowledge. The pairs share 15 questions and rest on eight and nine of the ten conversation clusters. Per-question outcomes for pair A: destructive wrong and Reprieve right 12, the reverse 8, both right 42, both wrong 38; pair B: 11, 10, 45, 34.

F.1 Conditional evidence-state checks on PerLTQA

Two further answer-level checks reuse the design of the diagnostic set of Table 19 on a second corpus. A question enters the set if and only if, at query time, exactly one arm of a storage-matched pair has its gold evidence entry dead (every PerLTQA question has one gold entry). Membership

Table 18: End-task check, held-out sample of 100 LoCoMo questions. Harm is the share of sample questions with a needed entry dead; differences are in points with 95% conversation-cluster intervals (ten clusters).

setting	storage	harm (sample)	strict accuracy	coverage
keep-all	1.000	0/100	56%	59.6%
$\mathcal{D}$, $\lambda = 0.23$	0.775	5/100	50%	57.1%
$\mathcal{R}_{50}$, $\lambda = 0.27$	0.775	5/100	54%	57.7%
$\mathcal{D}$, $\lambda = 0.30$	0.597	11/100	55%	59.6%
$\mathcal{R}_{100}$, $\lambda = 0.40$	0.598	13/100	56%	59.9%
pair A, $\mathcal{R}$ minus $\mathcal{D}$			$+4.0$ $[-4.0, +12.0]$	$+0.6$ $[-5.7, +7.3]$
pair B, $\mathcal{R}$ minus $\mathcal{D}$			$+1.0$ $[-5.0, +7.0]$	$+0.3$ $[-5.9, +6.0]$

Table 19: End-task check, mechanism diagnostic on the questions where the two policies of a matched pair hold different evidence at query time. One pair-B observation with an ambiguous judgment is excluded and reported in Appendix F.

pair	evidence held by	n	follows	opposes	both right	both wrong
A	Reprieve	18	5	0	3	10
A	destructive	15	2	1	1	11
B	Reprieve	35	13	1	4	17
B	destructive	29	7	2	7	13
A	all	33	7	1	4	21
B	all	64	20	3	11	30

is computed from the replay before any answer exists and reads neither answers nor retrieval quality. The sets are therefore *conditional*: they are not random samples, and their arm-level accuracy difference is not a corpus-level comparison of the two semantics.

Table 20 reports both checks. *Oracle check.* The pair is the exact PerLTQA storage match of Appendix D.4 ($\mathcal{D}$ at $\lambda = 0.49$ against $\mathcal{R}_{10}$ at $\lambda = 0.74$, storage 0.7426 each), where gold evidence is dead for 258 (3.14%) and 353 (4.30%) of the 8,214 queries. The difference set has 237 questions in 31 clusters: 166 on which only the Reprieve lost the evidence, 71 on which only destructive eviction did, and none on which both did.

BM25 check. Under the stateful retriever of Appendix D.10 (every cold entry in the top 10 is swapped in; the reader sees the top 5), the oracle pair is no longer storage-matched (Reprieve 0.814), and its corpus-level gold-death rate falls from 4.30% to 2.33% because retrieval-triggered swap-ins also rescue cold gold entries. We re-pair by storage alone on the fixed grid $\lambda \in \{0.74, 0.78, 0.82, 0.84, 0.86, 0.90, 0.94, 0.95\}$, $\theta \in \{0, 5, 10\}$, which selects $\mathcal{R}_5$ at $\lambda = 0.84$ (storage 0.74282 against 0.74261; gold dead for 292 versus 258 queries). The re-computed difference set has 168 questions in 30 clusters, 101 unfavourable and 67 favourable to Reprieve.

Reader, judge, rubric construction and scoring are those of Appendix F; necessary points were audited against the sources and frozen before any answer existed. Intervals use 10,000 cluster resamples.

Reading. In both checks correctness follows the arm holding the evidence far more often than it opposes it (79:6 and 44:4; two-sided sign tests $p = 2.4 \times 10^{-17}$ and 1.5×10^{-9}), on more questions and clusters than the LoCoMo diagnostic and, in the second check, under a real retriever. This is the evidence-error mechanism that the harm proxy assumes. The arm-level differences, by contrast, are set by composition. In the oracle check the holder's advantage is nearly symmetric (31.3 versus 29.6 points; difference $+1.7$, post-hoc cluster interval $[-14.4, +17.1]$), so the -13.1-point gap is what 166 versus 71 lost questions imply: Reprieve loses evidence more often at this operating point, exactly as its offline harm (4.30% against 3.14%) predicts. Under BM25 Reprieve still loses more (101 versus 67) yet ties on strict accuracy, because retrieval coupling changes *which* evidence survives: retrieval-triggered swap-ins keep alive the entries the retriever surfaces, so 54 of the 67 Reprieve-held gold entries (81%) reach the prompt against 53 of the 101 destructive-held

Table 20: Conditional evidence-state checks on PerLTQA (strict accuracy, %). Upper block: arm level; the Reprieve-minus-destructive difference is driven by how many questions each arm loses (composition), not by a per-question advantage. Lower block: stratified by which arm holds the evidence; "advantage" is holder minus non-holder accuracy in points.

check	n (clusters)	keep-all	$\mathcal{D}$	$\mathcal{R}$	$\mathcal{R}-\mathcal{D}$ strict	$\mathcal{R}-\mathcal{D}$ coverage	follows : opposes
oracle, $\mathcal{R}_{10}$ (0.74)	237 (31)	57.4	41.8	28.7	$-13.1\ [-21.1, -4.0]$	$-15.5\ [-22.6, -7.4]$	79 : 6
BM25, $\mathcal{R}_5$ (0.84)	168 (30)	41.1	34.5	34.5	$0.0\ [-9.0, +8.4]$	$-4.4\ [-13.9, +4.1]$	44 : 4
	evidence held by	n	$\mathcal{D}$	$\mathcal{R}$	advantage		
oracle	$\mathcal{D}$	166	51.8	20.5	+31.3		
oracle	$\mathcal{R}$	71	18.3	47.9	+29.6		
BM25	$\mathcal{D}$	101	43.6	23.8	+19.8		
BM25	$\mathcal{R}$	67	20.9	50.7	+29.9		

ones (52%). The retriever also bounds what any memory policy can deliver: even under keep-all only 63.1% of these 168 gold entries reach the top-5 prompt, and both arms are wrong on 86 questions. Neither check shows that one semantics is better; together they show that answer error tracks evidence loss, and that which policy loses more evidence depends on the corpus, the operating point and the request policy.

Execution. Oracle check: $711/711$ answers and judgments; BM25 check: $504/504$; no ambiguous judgment. Rubrics flagged by the source-only audit (44 and 27) were corrected before freezing, and 98 serialisation defects were repaired without changing any decision. One judgment was adjudicated (a quoted "evidence" that was the point text itself) and one Reprieve answer, truncated because reasoning tokens counted against the output budget, was regenerated once at 4,096 tokens, the only arm-asymmetric retry. Cost: \$14.56.

G Additional related work

Retention scores. DAM (Sun et al., 2025) frames memory management as sequential decision making; TraceRetain (Reddy, 2026) and the multi-factor value model (Chen & Cheng, 2026) score entries with interpretable features and study retention under controlled distractors or blind forgetting. Our formal results condition on the score trajectory rather than claiming that one scorer is best.

Evaluation methodology. MemDelta (Wang, 2026) shows that the embedding model alone shifts LongMemEval accuracy by 6 points; we add the schedule confound and its blindness to re-insertion. Cluster-level intervals follow (Miller, 2024; Bowyer et al., 2025).

Cache replacement. Belady's anomaly (Belady et al., 1969) is the classical failure of inclusion: FIFO can miss more with a larger cache. Theorem 2 locates the analogous failure for reversible threshold eviction on the re-inserted set and Proposition 3 names its single mechanism (a rescue restarting protection). Exclusive two-level caches demote evicted blocks to a victim or lower-level cache (Jouppi, 1990; Wong & Wilkes, 2002), as the cold tier does; hierarchical TTL analyses (Che et al., 2002; Choungmo Fofack et al., 2014) give hit probabilities rather than per-failure attribution. Ghost lists of ARC, 2Q and LIRS (Megiddo & Modha, 2003; Johnson & Shasha, 1994; Jiang & Zhang, 2002) retain evicted keys, like tombstones, and use ghost hits to adapt the policy; our prospective rule (Appendix D.6) uses tombstone hits in the same spirit, and Appendix D.7 compares it with access-aware cache policies.