DUDE, WHERE'S MY STATE? EXECUTION INFORMATION REQUIREMENTS FOR STATEFUL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-running agents must preserve information that later steps depend on. We introduce the *Execution Information Requirement* (EIR), a lower bound on the information that must remain accessible for correct completion under specified task and access conditions. We develop *LACUNA*, a framework that generates tasks with known dependencies and varies information demand, retention, and recovery separately from the difficulty of individual operations. Across four models, restoring a missing result raises accuracy on affected recall steps to 100%, compared with 0% for equal-length irrelevant information. Sufficient storage alone does not ensure success: retention policies can discard required results, errors can propagate through later computations, and agents can stop before recovery is complete. We also introduce *VESTIGE*, which constructs semantic graphs from agent execution traces to estimate information requirements retrospectively. Together, these contributions provide a basis for evaluating whether agents preserve and recover the information their tasks require.

1 INTRODUCTION

Long-running agents must preserve information across interdependent steps and recover it when needed. An agent can have ample context capacity and be capable of its next operation, yet lack the information that operation requires. Figure 1A illustrates the problem: an OpenHands agent edits a file after a clipped view omits a line later changed by the reference fix, and the next test fails.

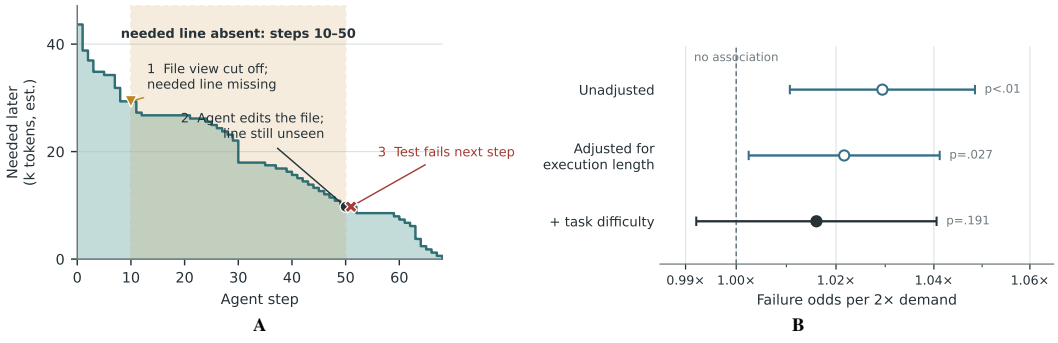


Figure 1: **Missing information and task difficulty in real agent tasks. A:** The agent edits a file without seeing a clipped line; the next test fails. Teal shows estimated information used later. **B:** Higher memory demand is associated with more failures, but the association is no longer statistically significant after accounting for task difficulty ($p = .191$). Dots show failure odds ratios; bars show 95% confidence intervals. The dashed line at 1 means no association.

Memory systems use summarization, retrieval, and learned retention policies to manage information (Packer et al., 2023; Kang et al., 2025; Sun et al., 2026; Zhang et al., 2026; Wu et al., 2026). Evaluations measure recall, task completion, and memory use (Liu et al., 2024; Vodrahalli et al., 2024; Jimenez et al., 2024; Yao et al., 2024; Merrill, 2026; Shen, 2026; He, 2026). To explain memory-management failures, we also need to establish what information a task requires and separate the

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.

effects of missing information from the difficulty of the computation. We must also understand the cost of recovery, since reconstructing one missing result may require repeating several earlier computations.

We introduce the *Execution Information Requirement* (EIR), a lower bound in bits on the information from earlier steps that must remain accessible for correct completion. EIR depends on what later steps may require and what information they can access, independently of how an agent represents it. For example, retaining a sum suffices if only that sum is needed; answering queries about individual values requires more information. We define *information pressure* as EIR divided by accessible storage capacity, both measured in the same units. Tasks that demand more memory also tend to be harder. Among 2,018 analyzed OpenHands task executions, estimated peak demand stays below one-eighth of the 128k-token window in 91% of cases. After accounting for task difficulty, we find no statistically significant link between demand and failure (Figure 1B).

We therefore introduce *LACUNA* to vary how much information a task requires and what the agent can access, independently of the difficulty of individual operations. Across four frontier models, restoring the exact missing result raises accuracy on the affected step to 100%, compared with 0% for matched irrelevant information and 11.3% on average without an injection. These interventions establish that unavailable information causes the tested errors. We evaluate four memory architectures under increasing information pressure and define *recovery cones* to describe the earlier results and computations needed to reconstruct a missing result.

We connect these controlled experiments to real tasks through *VESTIGE*, a method that constructs a semantic graph of actions and state dependencies from agent execution traces to estimate execution information requirements. The graph links observed information to later activity involving the reference solution, allowing us to examine how access to relevant information relates to failure. We apply VESTIGE to SWE-bench, τ-Bench, and Terminal-Bench. In SWE-bench Verified, failed task executions have longer gaps between visits to solution-relevant files: doubling the longest such gap is associated with $1.44\times$ higher failure odds, controlling for task and execution length.

Our contributions are:

- **A foundation for studying agent memory.** EIR quantifies what information must remain accessible; recovery cones describe the computation needed to reconstruct missing results.
- **A controlled evaluation framework.** LACUNA varies required and available information independently of the difficulty of individual operations, enabling causal tests and comparisons of retention and recovery across four models and four memory architectures.
- **VESTIGE.** A method that constructs a semantic graph of actions and state dependencies from agent execution traces to estimate execution information requirements.

2 Execution Information Requirement

The information an agent must preserve depends on what the remaining steps may require. It can discard details that cannot change a required response, but must preserve the distinctions that can. Following the communication view of Kushilevitz & Nisan (1997), we define the *Execution Information Requirement* (EIR) as the minimum information that must pass from earlier execution to the remaining work.

An example. Figure 2 shows six exploration histories for a timezone bug. After the dashed line, the entire remaining task is to identify the file to edit and whether to return an aware or naive datetime. Histories h_1 and h_2 both require $(\texttt{dates.py}, \text{aware})$ despite different exploration paths, so they can share a memory representation. History h_3 requires $(\texttt{dates.py}, \text{naive})$ and must remain distinguishable. The six histories yield four distinct answer pairs. Retaining which pair applies is necessary and sufficient, giving $\text{EIR}_t = \log_2 4 = 2$ bits.

Definition. Consider a point after step t, which we call an *execution cut*. It separates the completed history from the remaining work $\mathcal{W}_t$. We specify the possible histories, future inputs, required outputs, and sources the agent can access. We call these assumptions the *workload contract*.

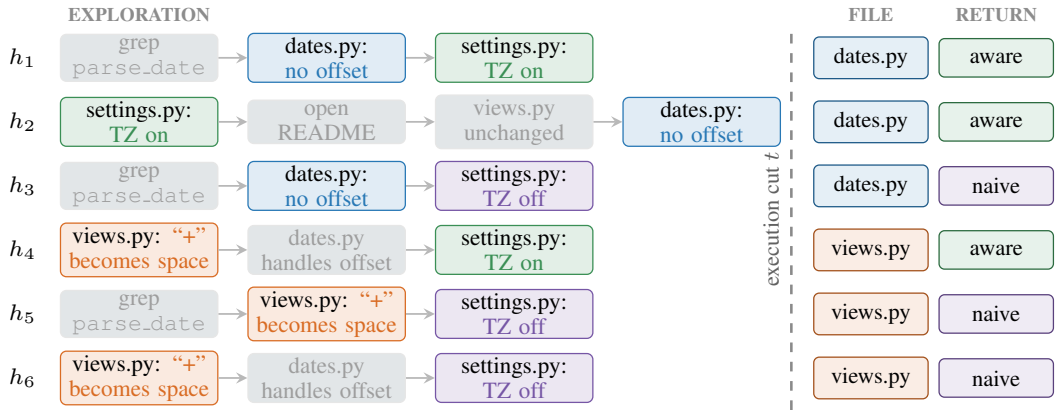


Figure 2: **Different histories can require the same information.** The remaining task asks which file to edit and whether to return an aware or naive datetime. Histories with the same answers form one group; grey observations do not change either answer.

Fix information Z_t supplied separately, such as the task specification and public parameters. All other information about the past accessible after the cut must be represented in the retained state, including external files and memory that the agent can retrieve. Let $\mathcal{X}_t$ contain the histories consistent with this information at the cut, and let $\mathcal{U}_t$ be the common set of allowed future input sequences. For workloads with uniquely specified outputs, let $F_t(x, u; Z_t)$ be the required output sequence after history x and future inputs u. Two histories are *continuation-equivalent* if every allowed future input requires the same output after either history:

$$x \equiv_{\mathcal{W}_t, Z_t} x' \quad \Longleftrightarrow \quad F_t(x, u; Z_t) = F_t(x', u; Z_t) \quad \text{for all } u \in \mathcal{U}_t. \tag{1}$$

Let N_t be the number of equivalence classes, or groups of histories with identical future responses. We define

$$\mathrm{EIR}_t(\mathcal{W}_t; Z_t) = \log_2 N_t \quad \text{bits}. \tag{2}$$

A sufficient state must distinguish these N_t classes. Otherwise, two histories requiring different responses would leave the agent in the same state, making a correct response impossible for at least one of them. Conversely, the class index determines every required future response. Thus, for finite N_t, the minimum fixed-length binary representation uses $\lceil \log_2 N_t \rceil$ bits. This argument concerns information; it does not bound the computation needed to encode or use that state.

What EIR measures. EIR is a worst-case, zero-error requirement: correctness must hold for every allowed history and future input. Any compression is permitted if it preserves the required distinctions. EIR is measured in bits; its conversion to tokens depends on the representation. Appendix A.1 gives the finite enumeration procedure.

Capacity and availability. Let $C_t > 0$ be the capacity of the available storage in bits, so it can represent 2^{C_t} distinct states. Using the same access assumptions as EIR, define *information pressure* as

$$P_t = \frac{\mathrm{EIR}_t(\mathcal{W}_t; Z_t)}{C_t}. \tag{3}$$

If $P_t > 1$, the storage cannot distinguish all required classes, so no agent can guarantee success on every allowed case. If $P_t \leq 1$, sufficient storage is possible, but the agent must still preserve the right information and perform the computation correctly. For example, if m independent b-bit values must be retained and storage holds h such values, the ratio is $mb/(hb) = m/h$.

EIR specifies what must remain accessible; testing whether missing information causes errors requires controlling what the agent can access. Section 3 introduces LACUNA, our evaluation framework for this purpose. Section C then introduces VESTIGE for estimating relevant information from recorded task executions.

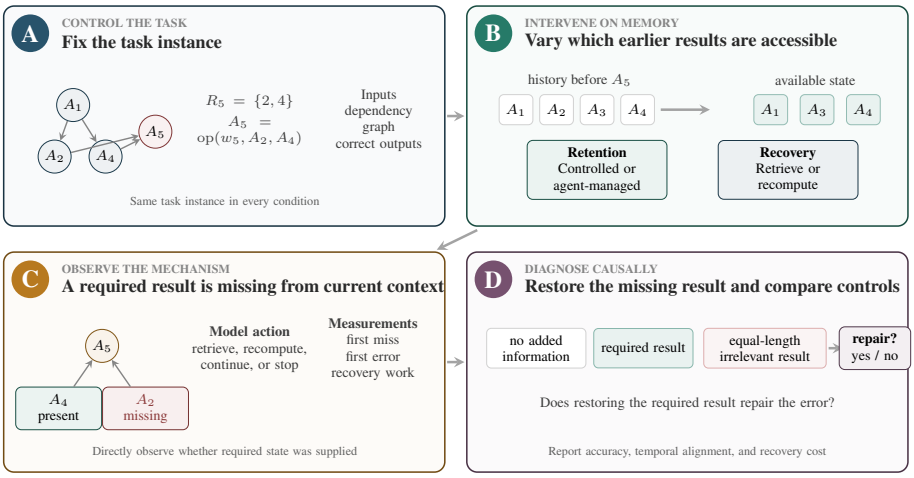


Figure 3: **LACUNA tests whether missing information causes errors.** Fix a task instance (A), vary access to earlier results (B), and observe the agent when a required result is missing from context (C). Compare restoring that result with no added information and an equal-length irrelevant result (D).

3 LACUNA: A Framework for Evaluating Agent Memory

LACUNA is a framework for generating multi-step tasks and testing how agents retain and recover information. Tasks are built recursively: each step combines a new input with results from earlier steps. Specifying the operations and dependencies lets us generate task instances of varying length and information demand, with known correct outputs. We then vary access to earlier results while keeping the task fixed, and test whether restoring missing information repairs errors. Figure 3 shows this evaluation procedure.

3.1 Task construction

Each task is a directed acyclic graph: nodes are steps, and an edge $r \to t$ means that step t uses the result of step r. Given new input w_t and identifiers R_t naming earlier results, the agent computes

$$A_t = \mathrm{op}(w_t, (A_r)_{r \in R_t}), \qquad R_t \subseteq \{1, \ldots, t-1\}. \tag{4}$$

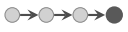
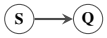
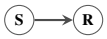
Identifiers distinguish results even when their values coincide.

A *workload* specifies how inputs and dependencies are generated, how results are computed, and which outputs are scored. It also defines when dependencies are revealed and what may be retained. In online workloads, the agent learns a step's dependencies only when that step arrives. A *task instance* is generated with a fixed random seed, fixing its inputs, dependencies, and correct outputs before execution. We compare memory conditions on the same instances.

Generation is controlled by the number of steps n, operation op, dependency-window size m, maximum lookback ℓ (default m), and maximum dependencies per step f_{in}. Each result record contains s value slots with d decimal digits per slot. These parameters control task length, dependency structure, and the amount of information in each result.

Workloads used in this study. We evaluate six examples of this construction (Table 1), chosen to test different memory demands. *Running Maximum* needs only one summary value, whereas *Stepwise Maximum* evaluates intermediate maxima that may be needed again. This contrast illustrates how future use changes what must be retained. *Stepwise Sum* tests retention and reconstruction when earlier results are requested only as each step arrives; *Stepwise Top-k* tests tracking collections of values across steps. *Full Lookup* separates storage from later queries, while *Store–Recall* alternates between them. These choices illustrate the framework; other operations and dependency patterns can be specified in the same way.

Table 1: **Workloads evaluated in this study.** Rules, dependencies, evaluated outputs, and EIR for the evaluated settings. Here $b = \log_2 q$ for a value domain of size q, and † marks EIR that depends on the dependency graph.

Workload	Operation	Dependencies	Evaluated outputs	EIR
Stepwise Sum	Modular sum over two dependencies sampled from a preceding window		Every step; identifiers revealed only when used	$16b$
Running Maximum	Update the maximum along a fan-in-one chain		Final global maximum	b
Stepwise Maximum	Take the maximum over two randomly selected earlier results		Every intermediate local maximum	graph-dependent†
Stepwise Top-k	Compute the top-k values over two earlier results		Every frontier after warm-up	not characterized
Full Lookup	Store labelled values, then query arbitrary labels; 16 STORE + 48 QUERY steps		Every query	mb
Store–Recall	Alternate STORE and RECALL steps over eligible stored identifiers		Every recall	$7b$–$8b$

We vary the record limit for five workloads and use Running Maximum as a one-value control. Exact EIR for Stepwise Maximum depends on the ancestor relationships among results still needed by future steps (Appendix B.2). Stepwise Top-k EIR is not characterized, so we report its record limits without an exact EIR-to-capacity ratio.

Table 1 reports the evaluated settings, not universal workload formulas. At saturated $m = 16$ cuts, Stepwise Sum requires $16b$ (generally $|W_t|b$), while Store–Recall requires $8b$ after STORE steps and $7b$ after RECALL steps. Query and recall identifiers are revealed only when their step arrives. Stepwise Top-k record-limit results are descriptive because its numeric EIR is not characterized; Appendix A gives the formal scope.

3.2 Controlling information availability and recovery

Once the task is fixed, we vary what the agent can retain and recover. We distinguish results needed by future steps (*required state*), results preserved by the memory system (*retained state*), and information accessible through the permitted interfaces and recovery budgets (*available state*). A stored result may require retrieval, while a result in context may no longer be needed.

The limit h specifies how many complete labelled result records may be available at once. Under *Controlled Retention*, the harness resets model-visible context at each step and supplies the most recent h submitted results, or all results if fewer exist. Under *Agent-Managed Retention*, the agent system chooses what to retain, summarize, store externally, and retrieve through its permitted interfaces.

Conditions also specify whether missing results can be retrieved directly or reconstructed. The budget B_{oracle} limits the number of directly retrieved results; B limits the number of recomputed steps. Reconstructing a result may require recovering its dependencies first; these form its *recovery cone*. A result is unavailable if it cannot be obtained through the permitted interfaces within these budgets.

3.3 Evaluation controls

The controls first establish that the agent can perform the computation, then test whether missing information causes errors.

Computational capability. The *full-information control* supplies all required earlier results and disables recovery. For window-based workloads under Controlled Retention, $h = m = \ell$ covers

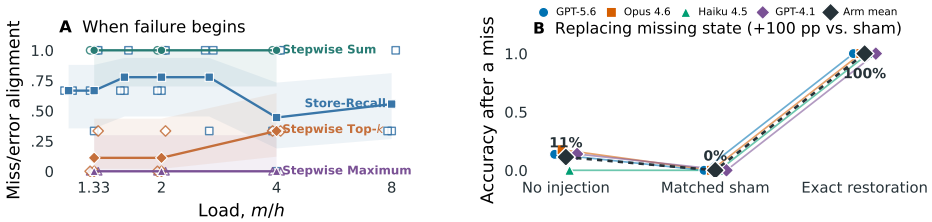


Figure 4: **Missing inputs predict errors; restoring them repairs answers. A:** Fraction of trials whose first missing dependency coincides with the first error. Bands show 95% confidence intervals. **B:** Restoring the required result repairs affected Store–Recall answers; adding an irrelevant value of the same length does not.

the eligible predecessor window. Remaining errors reflect computation, use of the wrong result, formatting, or protocol handling. We attribute errors to information loss only in conditions that pass this control.

Causal intervention. For a step error, we compare no added information, restoration of the required result, and an irrelevant value of the same length. If only restoration repairs the answer, the comparison identifies the missing information as the cause rather than the added token count. All controlled prompts fit within the model's context window; reducing h shortens the input while increasing the demand relative to available storage.

Errors and measurements. We distinguish an incorrect answer with correct inputs available (*computation error*), an unavailable required result (*state-supply error*), and reuse of an incorrect earlier result (*cascade error*). An *execution failure* is an incorrect final task outcome. LACUNA records required, retained, and available results; retrievals and recomputed steps; recovery depth; latency; and subsequent errors. We report both step and final-task accuracy to capture how intermediate mistakes affect task completion.

Interface checks. We check that the retention interface does not carry information between steps through unintended channels. Across 18,068 checks on 100 generated instances, the audit finds no violations in payload width, identifiers, occupancy, ordering, serialization, message length, or isolation between context resets. The appendix provides the full audit, experiment configurations, model settings, and trial counts.

4 Results

We test the effects of missing information, recovery cost, retention behavior, and memory architectures. Information-loss claims use conditions that pass the full-information control; residual computation errors in Stepwise Top-k are analyzed separately. Detailed comparisons and additional figures appear in Appendix B.

4.1 RQ1: Does unavailable required state predict and cause execution errors?

Restoring missing information repairs errors. In Store–Recall ($m = 8$, $h = 4$, recovery disabled), supplying the required result raises accuracy on affected recall steps to 100% across four models, compared with 0% for an equal-length irrelevant value and 11.3% without added information (Figure 4B). The intervention isolates information content from added prompt length. A Stepwise Maximum replication also reaches 1.000 with exact restoration, versus 0.642 without restoration and 0.714 with irrelevant information (Appendix Table 6).

The first missing dependency coincides with the first error in all 27 Stepwise Sum trials with restricted memory. Alignment is weaker when a missing value can be guessed or does not change the maximum. Across 96 Full Lookup trials, query accuracy falls from 0.997 at $h = 16$ to 0.092 at $h = 2$, while queries for retained values remain nearly perfect. Errors can also propagate: one 64-step Stepwise Sum execution contains 19 errors at missing-input steps and 26 cascade errors, with

no independent computation errors. Later steps correctly apply the operation to already incorrect inputs. Appendix B.3 reports the retention curves and error decomposition.

4.2 RQ2: What determines reconstruction cost?

Recovery work follows the missing dependencies. Direct retrieval costs one read. Reconstruction with reuse evaluates each node in the recovery cone once; without reuse, shared ancestors are evaluated repeatedly. Mean work on the tested chain, balanced, and bushy graphs is 13.0, 38.5, and 45.3 node evaluations with reuse. Without reuse, the bushy-graph mean rises to approximately 1.1×10^{10}. Cone size measures total work; depth describes the longest sequential dependency chain.

In 88 agent-executed probes, the model selects reads while the harness performs arithmetic. All 69 successful recoveries use exactly the minimum number of reads; the other 19 stop before reconstructing the target. Thus, the observed failures involve incomplete recovery rather than redundant traversal. Appendix B.4 provides the graph comparisons and per-workload results.

4.3 RQ3: How does retention behavior change under information pressure?

Adaptive retention benefits from repeated queries, while storage format affects reliability. In Full Lookup with Zipf-distributed queries and $h = 4$, hit rates are 0.493 for least-frequently-used retention (LFU), 0.438 for least-recently-used retention (LRU), and 0.111 for the fixed policy. Uniform queries show no adaptive advantage. Direct retrieval keeps query accuracy at least 0.998, so hit rate exposes the policy difference.

Model-directed scratch storage improves over LRU for Claude Opus 4.6 (0.762 versus 0.664) and GPT-5.6 (0.750 versus 0.650), but reduces accuracy for Claude Haiku 4.5 (0.438 versus 0.616). Holding LRU fixed, key–value storage improves accuracy over text scratch storage by 25.9–33.0 percentage points across all four models. Storage format provides the more consistent benefit in these tests (Appendix B.5).

4.4 RQ4: How closely do architectures approach the EIR floor?

Enough capacity does not ensure correct retention. At $h = 16$, Stepwise Sum can retain all 16 eligible results, yet LRU scores 0.410 compared with 0.934 for exact-window retention. Restoring LRU's missing results raises its score to 0.934; irrelevant information reaches only 0.444. The restoration test identifies discarded information as the source of this gap (Appendix Table 11). Offline comparisons use future-aware Belady as a hindsight benchmark, distinct from the worst-case online requirement defined by EIR.

Complete architectures also depend on correct retrieval and execution. On Store–Recall, unbounded forced paging reaches 1.000, Mem0 0.996, FIFO 0.811, and Letta 0.370. These are system-level results under different storage conditions, not equal-capacity policy comparisons. Stepwise Top-k remains difficult even with forced paging (0.501), consistent with its computation limitation. Appendix B.6 separates policy comparisons from end-to-end architecture results.

5 Discussion and Limitations

What the framework makes measurable. EIR describes the information the remaining computation requires at an execution cut; it does not describe how a memory system stores, retrieves, or recomputes that information. LACUNA uses this separation to turn an otherwise ambiguous failed run into a set of testable possibilities. Offline policy simulations measure whether a retention rule preserves the required state, while end-to-end runs also include model outputs, tool actions, and step computations. The gap between these measurements separates retention-policy errors from errors in end-to-end execution.

What this implies. EIR is a lower bound on the information required to guarantee correct continuation, not a predictor of whether one run will fail. A run's outcome also depends on the retention policy, external storage and retrieval, model capability, and correct use of returned state. Two workloads with the same input length can therefore occupy different feasibility regimes.

Implications for memory systems. The Mem0 and Letta results illustrate why memory quality cannot be inferred from the presence of an external store alone. Mem0 nearly matches the forced-paging upper bound on Store–Recall, whereas Letta performs substantially worse in the same agent-executed setting. These runs do not establish a universal ranking of memory products: they combine each system's interface, the model's decisions, and answer submission. They do show that task-aligned retention and reliable access are measurable properties, and that a system can fail even when storage is available.

Scope of the trajectory estimator. The offline estimator also works when a task has no declared workload graph. From a completed tool trace, it measures the file information that the recorded run used again later. This lets us compare demand profiles across tasks and agents and find high-demand parts of a run. Processing all 72,562 eligible public SWE trajectories shows that this operational signal can be reconstructed at scale from heterogeneous agent logs; applying the same construction to 318 τ-Bench trajectories extends the measurement from file state to transactional entities, while a shell-command adapter recovers file-state demand for 166 Terminal-Bench runs. Its interpretation is anchored by LACUNA validation: under known dependencies, the graph retains required state and removes injected distractors. Because the graph uses the completed continuation, it neither predicts future need nor certifies the minimum information required by every correct solution. Reread behavior likewise does not establish what the model retained internally or why an observed run failed. It cannot distinguish context loss from successful retention, deliberate reinspection, delayed work, or wrong-path reasoning. The raw software-agent traces also reveal a separate supply limit: scaffolds clip long command or file-view outputs before they enter the conversation. Because the exports do not identify which omitted lines were required, clipping prevalence and outcome associations do not establish a causal state-supply error. Future audits should retain exact per-call prompts and unabridged tool outputs so required omitted content can be tested directly. Those are separate prospective questions, not requirements for the present offline measurement contribution.

Design implications and limitations. The decomposition suggests retaining state according to predicted future use, encoding it without irrelevant identities, tracking what has left resident memory, and measuring retrieval by the dependency work it triggers. The same view applies beyond language agents to sequential computations such as streaming algorithms and database execution.

The substrate is engineered to isolate information pressure, so it does not capture all per-step difficulty or content priors of a realistic task. Exact EIR values require a specified workload contract and are easiest to obtain for deterministic computations with known outputs. The formal values use an unrestricted encoding assumption, whereas experiments use fixed-width answer records; the former is a lower bound on the latter. Results should therefore not be read as an intrinsic ranking of workload families or memory products. Capacity calibration is workload-specific, and the strict equal-capacity comparison is limited to systems with an auditable common capacity unit. Letta, A-MEM, and Graphiti are therefore reported in a separate native-system extension with scored counts; pending or infrastructure-error cells are not treated as behavioral failures. Real trajectories also contain capability, exploration, and tool-use failures that logs alone cannot isolate. Direct clipping establishes that supplied tool output can be incomplete, but observational logs do not identify its causal effect on success. Extending the framework to stochastic workloads, richer tool interfaces, and learned recovery policies is an important direction for future work.

6 Related Work

Predictive and task-relevant state. Several theories define a compact state by asking which pasts have the same future consequences. Myhill-Nerode equivalence merges prefixes that admit the same continuations (Nerode, 1958). Computational mechanics similarly groups histories that give the same distribution over future observations into *causal states* (Crutchfield & Young, 1989; Shalizi & Crutchfield, 2001). EIR uses the same basic idea: merge pasts that the future cannot distinguish. It applies this idea at a finite execution boundary with a specified remaining workload. It measures the number of states needed to guarantee correct continuation in the worst case. In contrast, the information bottleneck and predictive-information frameworks optimize average-case mutual information under a probability distribution (Tishby et al., 1999; Bialek et al., 2001). EIR there-

fore requires no distribution over execution prefixes: the future computation itself determines which prefixes must remain distinct.

Long-context evaluation and agent memory. Long-context benchmarks reveal that models can fail even when the required information is present: accuracy varies with evidence position and with how reasoning is distributed across the context (Liu et al., 2024; Vodrahalli et al., 2024). However, their aggregate recall or task-accuracy scores do not identify why a failure occurred. They cannot generally distinguish insufficient memory capacity from retaining the wrong information, failing to retrieve retained information, or failing to use information that was available. Evaluations of external storage, hierarchical memory, trajectory summaries, and learned retention policies face the same attribution problem (Park et al., 2023; Packer et al., 2023; Kang et al., 2025; Sun et al., 2026; Zhang et al., 2026): an improved final score shows that a memory mechanism helps, but not which state-availability error it prevents. More targeted evaluations test long-run state consistency, detect disagreement between an agent and its environment, separate stages of memory use, or test whether remembered information supports later actions (Samiei et al., 2026; Song & Cai, 2026; Zhu et al., 2026; Shen, 2026; He, 2026). These evaluations localize memory problems, but do not jointly identify the information required at a failed step and test whether its absence caused the failure. LACUNA makes these causes separately measurable by controlling what information later steps need, what information the agent can access, and whether missing information can be reconstructed. LACUNA therefore lets developers stress-test memory systems and policies under controlled memory requirements. Our retrospective demand proxy uses estimates of these requirements to assess state availability in completed real-agent runs.

Context compaction and task-aware memory. Recent theory studies what a bounded memory should keep. Context-compaction work relates the information retained in a summary to communication complexity (Tirmazi et al., 2026), while hierarchical and rate-distortion formulations study how memory can trade representation size against downstream performance (Talebirad et al., 2026; Colaco & Lahjouji, 2026). Related decision-centric work keeps distinctions between histories only when merging them would reduce future decision quality (Zou et al., 2026). Together, these approaches motivate task-aware memory: a useful representation should keep the parts of the past that matter to future behavior. They focus on mechanisms for constructing such representations or on the lossy trade-offs between memory size and downstream performance. Their evaluations can show that a memory method improves performance, but do not directly measure whether a particular failure arose from unavailable required information. EIR specifies what information the task requires, and LACUNA uses it to create and diagnose state-unavailability errors while trajectory length, graph structure, and per-step difficulty are held fixed.

Recomputation and information lower bounds. A compiler marks a value as *live* when the program may use it again (Kildall, 1973), although multiple live values may admit a joint encoding when the future workload does not need their individual identities. Pebbling games study how computation graphs trade storage for recomputation and data movement (Paul et al., 1977; Hong & Kung, 1981), and activation rematerialization applies this trade-off in neural-network training (Chen et al., 2016; Gruslys et al., 2016). These ideas inform our analysis of the cost of reconstructing needed information after it is lost. Classical streaming and one-way communication lower bounds provide the tools for our requirement results (Alon et al., 1999; Miltersen et al., 1998; Kushilevitz & Nisan, 1997): arbitrary labelled access requires information that grows linearly with the input, whereas aggregate and extremum workloads need only a constant-size state.

7 Conclusion

Reliable agent execution requires both sufficient memory and access to the right information. EIR formalizes the information requirement, while LACUNA enables controlled tests of retention and recovery. Our experiments show that missing information causes errors even when models can perform the underlying computation. They also show why capacity alone is insufficient: retention choices determine what remains accessible, dependencies determine how errors spread, and recovery requires completing the necessary reconstruction. VESTIGE extends this analysis to recorded agent executions through retrospective estimates of relevant information. These findings motivate memory

systems that track future information needs and evaluate retention, retrieval, and recomputation together.

AI Use Statement

Generative AI tools were used for manuscript editing, writing, finalization, and software implementation.

Reproducibility Statement

The appendix states workload contracts, proofs, controls, experiment settings, and observational limitations. The anonymous supplementary archive contains the LACUNA and trajectory-analysis code, tests, compact aggregate results, theory artifacts, figure scripts, checksums, and instructions for validation. Raw public trajectories are not redistributed; the archive identifies their public dataset sources and the scripts used to process them.

References

Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

William Bialek, Ilya Nemenman, and Naftali Tishby. Predictive information and the emergence of simple irreducible representations. *Neural Computation*, 13(11):2409–2463, 2001. doi: 10.1162/089976601753195950.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Ashwin Gerard Colaco and Nada Lahjouji. What to keep, what to forget: A rate–distortion view of memory compaction in LLMs and agents. *arXiv preprint arXiv:2607.08032*, 2026.

James P. Crutchfield and Karl Young. Inferring statistical complexity. *Physical Review Letters*, 63 (2):105–108, 1989. doi: 10.1103/PhysRevLett.63.105.

Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Jun He. MemoryArena, 2026.

Jia-Wei Hong and H. T. Kung. I/o complexity: The red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pp. 326–333. Association for Computing Machinery, 1981. doi: 10.1145/800076.802486.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*, 2024.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025. URL https://arxiv.org/abs/2510.00615.

Gary A. Kildall. A unified approach to global program optimization. In *Proceedings of the 1st ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp. 194–206, 1973. doi: 10.1145/512927.512945.

Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997. doi: 10.1017/CBO9780511574948.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

William Merrill. Terminal-Bench 2.0, 2026.

Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of the ACM*, 45(4):682–720, 1998. doi: 10.1145/285055.285057.

Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958. doi: 10.2307/2033204.

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.

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*, 2023. doi: 10.1145/3586183.3606763.

Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10(3):239–251, 1977. doi: 10.1007/BF01683270.

Mahdi Samiei, Mahdi Mansouri, and Mahdieh Soleymani Baghshah. The illusion of procedural reasoning: Measuring long-horizon fsm execution in llms. In *AAAI 2026 Bridge Program on Language Models and Reasoning*, 2026. URL `https://arxiv.org/abs/2511.14777`.

Cosma Rohilla Shalizi and James P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of Statistical Physics*, 104(3–4):817–879, 2001. doi: 10.1023/A:1010388907793.

Kai Shen. Mem2ActBench, 2026.

Xinyuan Song and Zekun Cai. World-model collapse as a phase transition. *arXiv preprint arXiv:2606.31399*, 2026. URL `https://arxiv.org/abs/2606.31399`.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL `https://arxiv.org/abs/2510.11967`.

Yashar Talebirad, Ali Parsaee, Csongor Y. Szepesvari, Amirhossein Nadiri, and Osmar Zaiane. Toward a theory of hierarchical memory for language agents. *arXiv preprint arXiv:2603.21564*, 2026.

Hayder Tirmazi, Sam Markelon, Allison Bishop, and Michael Mitzenmacher. Context compaction theory. *arXiv preprint arXiv:2608.01326*, 2026.

Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, pp. 368–377, 1999.

Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, Jeffrey Hui, Nishanth Dikkala, Mehran Kazemi, Bahare Fatemi, Rohan Anil, Ethan Dyer, Siamak Shakeri, Roopali Vij, Harsh Mehta, Vinay Ramasesh, Quoc V. Le, Ed H. Chi, Yifeng Lu, Orhan Firat, Angeliki Lazaridou, Jean-Baptiste Lespiau, Nithya Attaluri, and Kate Olszewska. Michelangelo: Long context evaluations beyond haystacks via latent structure queries. *arXiv preprint arXiv:2409.12640*, 2024.

Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. Remember when it matters: Proactive memory agent for long-horizon agents. *arXiv preprint arXiv:2607.08716*, 2026.

Shunyu Yao, Noah Yu, Yikai Zhao, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

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

Jianing Zhu, Yeonju Ro, John Robertson, Kevin Wang, Junbo Li, Haris Vikalo, Aditya Akella, and Zhangyang Wang. Your agents are aging too: Agent lifespan engineering for deployed systems. *arXiv preprint arXiv:2605.26302*, 2026.

Mingxi Zou, Zhihan Guo, Langzhang Liang, Zhuo Wang, Qifan Wang, Qingsong Wen, Irwin King, Lizhen Qu, and Zenglin Xu. Remember the decision, not the description: A rate-distortion framework for agent memory. *arXiv preprint arXiv:2605.10870*, 2026. URL https://arxiv.org/abs/2605.10870.

A Proofs for Exact EIR Workloads

Let $\mathcal{A} = \{0, \dots, q-1\}$ be the value alphabet and let $b = \log_2 q$. The arguments below count continuation-equivalence classes under the workload contracts used in the paper and establish the exact EIR results reported in Table 1.

A.1 Exact EIR by finite enumeration

Algorithm 1 computes exact EIR for finite, nonempty sets of admissible prefix states and future input sequences, given an evaluator for the correct outputs of the fixed suffix workload. Here each u_i is an admissible future input sequence (or query in a one-step suffix), and $F_t(x, u_i; Z_t)$ is its required answer or output trace; the two are not interchangeable.

Algorithm 1 Exact EIR by enumeration

Require: Prefix states $\mathcal{X}_t$; future sequences $\mathcal{U}_t = (u_1, \dots, u_m)$ in a fixed order; fixed information Z_t; correct-output evaluator F_t for workload $\mathcal{W}_t$
Ensure: $\mathrm{EIR}_t(\mathcal{W}_t; Z_t)$ in bits
1: $\mathcal{S} \leftarrow \varnothing$ {Set of distinct output signatures}
2: **for all** $x \in \mathcal{X}_t$ **do**
3: $\quad s_x \leftarrow \big(F_t(x, u_1; Z_t), \dots, F_t(x, u_m; Z_t)\big)$
4: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{s_x\}$
5: **end for**
6: $N_t \leftarrow |\mathcal{S}|$
7: **return** $\log_2 N_t$

Two states have the same signature exactly when they are continuation-equivalent, so $|\mathcal{S}|$ counts the required equivalence classes. The procedure requires $|\mathcal{X}_t|\,|\mathcal{U}_t|$ evaluations of F_t, in addition to signature comparison and storage, and may be computationally expensive. When enumeration is impractical, an exact result requires a continuation-sufficient representation and a matching lower bound on the number of distinguishable context states.

A.2 Stepwise Sum (T-AGG-DAG)

The Stepwise Sum task scores all 64 answers, with reward equal to the number of correct answers divided by 64. At step u, the agent receives a new scalar w_u and references R_u, and must output

$$A_u = \left(w_u + \sum_{r \in R_u} A_r \right) \bmod q, \qquad q = 10^4.$$

References are revealed only when the step arrives. They are selected from the preceding $m = 16$ answers. Consequently, the final-aggregate-only contract does not characterize this experiment. This is the final-aggregate contrast highlighted in Section 2: a suffix that asks only for one final sum can retain an aggregate sufficient statistic, whereas this online suffix may expose different references at every remaining step and must preserve the eligible labelled values.

Contract for the online bound. Consider a cut after a correct prefix of t steps, before the next references are revealed. Let $n = \min(t, m)$ and let $x \in \mathbb{Z}_q^n$ be the ordered window of eligible answers. We require exact answers for every admissible future continuation, with no recovery or accessible copy of the prefix in the model-facing decision channel. The evaluated model receives no tool access and its decision prompt does not reveal the generator seed. The task container does contain the seed and generator; this restriction therefore does not automatically describe a shell-capable agent in that container. Primitive inputs range freely over $\mathbb{Z}_q$, independently of the graph. For $n \geq 2$, assume every unordered pair of distinct eligible indices is an admissible next reference set; for $n = 1$, the sole index is referenced. These assumptions define a full-support task family, not the finite seeded sample used for evaluation. The bound is a worst-case family-level zero-error requirement, not the entropy or finite-support EIR of the sampled trials. Hiding a deterministic seed does not itself establish full support or independence; the full-support family is the declared theoretical model. The supplied implementation check reports exactly two distinct references whenever two predecessors exist. If singleton queries were also permitted, the full-window lower bound would follow immediately from those queries.

Claim: interior cuts. If $n \geq 3$ and at least two steps remain, the online zero-error requirement is

$$\mathrm{EIR}_t^{\mathrm{on}} = n \log_2 q.$$

Thus, for $16 \leq t \leq 62$, the scalar AGG workload has requirement $16 \log_2(10^4) \approx 212.603$ bits as a log-cardinality. A jointly coded fixed-length binary representation of the window requires 213 bits; storing each scalar in its own integer-width binary slot requires $16 \times 14 = 224$ physical bits. The same full-window requirement holds for $n = 1$, and for $n = 2$ with at least two steps remaining in this $m = 16$ workload.

Proof. Fix a graph prefix. Every answer vector is attainable: choose primitive inputs successively as $w_u = A_u - \sum_{r \in R_u} A_r \pmod q$. The recurrence is triangular, so recursive dependence does not restrict the full answer-vector domain under the stated input assumption. Suppose two window states x, x' share a sufficient message. Set $\delta_i = x'_i - x_i \pmod q$. Equality of the next answer for every admissible pair requires $\delta_i + \delta_j = 0$ for every $i \neq j$. For $n \geq 3$, comparison of overlapping pairs implies all $\delta_i = c$ and $2c = 0$. For even q, the only possibilities are $c = 0$ and $c = q/2$. In the latter case, the first future answer is identical for the two states. At the following step, select that newly produced answer and any surviving old answer. Their sum differs by c, distinguishing the states. Such an old answer survives because $m = 16$. Therefore $c = 0$ is necessary, and all q^n states are distinct continuation classes. Retaining the ordered window achieves this bound: answer the next query, append its result, and discard the expired oldest value. For $n = 2$, equality of the first pair sum followed by a query combining the new result and either old value also distinguishes every state. For $n = 1$, the next answer distinguishes that value directly. □

Terminal boundary. If only one step remains, $n \geq 3$, and references always contain exactly two distinct indices, the indistinguishable shifts are precisely the constants c satisfying $2c = 0 \pmod q$. There are $\gcd(2, q)$ such shifts, giving

$$\mathrm{EIR}_t^{\mathrm{on}} = n \log_2 q - \log_2 \gcd(2, q).$$

For odd q there is no reduction. For the final scored cut with $q = 10^4$,

$$\mathrm{EIR}_{63}^{\mathrm{on}} = 16 \log_2(10^4) - 1 \approx 211.603 \text{ bits}.$$

If singleton reference sets are admissible, this exception disappears. After the last scored answer, the remaining requirement is zero.

Interpretation of the existing RQ1 results. The reported accuracy values and experiments are unchanged. At full-window interior cuts, a closed payload of h ordered q-ary scalar slots has log-cardinality capacity $h \log_2 q$, and the ideal online pressure is then $P_t = \mathrm{EIR}_t / C_t = m/h$. We call m/h the interior online pressure; before the window fills, the corresponding ratio is n/h, and the terminal cut has the correction above. For $h = 16, 12, 8, 4$, these pressures are $1, 4/3, 2, 4$. This identification requires that all prefix-dependent storage and metadata be accounted for; it does not certify the implemented channel from its name. For $h < m$, perfect execution cannot be guaranteed over all admissible continuations. This worst-case statement does not specify an average accuracy curve or require every sampled trajectory to fail. The observed curves still measure the combined effects of missing operands, computation, and propagation.

The statement that Stepwise Sum has a 13-bit requirement and therefore fails despite ample intrinsic capacity is withdrawn. The proposed AGG-versus-ARB reversal of intrinsic EIR is also not established by comparing this online task with the offline final-aggregate contract. Finally, Belady has access to future references: its advantage over an online policy is a hindsight comparison, not by itself avoidable waste for an agent given the same information as the tested policy.

Verification status. The implementation-inspection report supplied by the user confirms that the verifier scores all 64 answers, the step interface reveals only current references, the generator selects two distinct eligible predecessors once two exist, and the evaluated controlled-retention runs prohibit recovery. It also reports that the container contains the deterministic generator and seed, while the model has no tool access and its decision prompt exposes neither. These are reported source checks, not an independent source inspection in this workspace. The theorem concerns that restricted decision interface under the declared full-support family. Exact implemented channel capacity still requires accounting for all prefix-dependent fields and storage. The theorem does not identify the entropy of the three sampled seeds or guarantee their empirical failure rate.

A.3 Running Maximum (T-EXT-01)

Statement. If the sole future consumption of a nonempty prefix is its final maximum, then $\mathrm{EIR} = b$.

Proof. Two prefixes are continuation-equivalent exactly when their maxima agree. Each $a \in \mathcal{A}$ is attained by the constant prefix $(a, \ldots, a)$, so there are at least q classes. Retaining the maximum is sufficient, so there are at most q classes. Hence the class count is q and $\mathrm{EIR} = \log_2 q = b$. Reversing the order proves the corresponding minimum result. □

A.4 Stepwise Top-k (C-TOPK-DAG-BISIM)

At each step, the implemented frontier workload returns the sorted top-k values from the fresh segment and two referenced earlier frontiers. Every intermediate frontier is scored, while future reference labels are hidden at the cut. Its exact continuation classes are therefore the greatest fixed-point bisimulation classes of this labelled transition system. This characterizes when two retained states are equivalent, but does not provide a closed-form class count. Exhaustive bisimulation is practical only for reduced alphabets and windows; the evaluated $m = 16$, $k = 2$, $q = 10^4$ state space is not directly enumerable. Consequently, no numeric EIR or EIR-based pressure is claimed for Stepwise Top-k in this study.

A.5 Store–Recall (T-ARB-STORE)

The implemented workload alternates STORE and RECALL steps. Odd step u stores an independent value $A_u = w_u$. Even step u recalls one odd STORE id from $[\max(1, u - m), u - 1]$, with the id revealed only when the step arrives. Let $e(t)$ be the first even step after cut t. If $e(t)$ exceeds the task horizon, set $R_t = \varnothing$; otherwise define

$$R_t = \{s \le t : s \text{ is odd and } \max(1, e(t) - m) \le s\}.$$

Statement. Under full support over eligible future recall ids, $\mathrm{EIR}_t^{\mathrm{on}} = |R_t| \log_2 q$.

Proof. Only the STORE values with ids in R_t can influence future outputs. If two prefixes agree on their labelled values in R_t, every future recall of a prefix value agrees, while later STORE values are common future inputs. Conversely, if the prefixes differ at any $s \in R_t$, choosing s at step $e(t)$ distinguishes the continuations immediately. The independent STORE values therefore induce exactly $q^{|R_t|}$ continuation classes, and retaining their labelled values attains the bound. Thus $\mathrm{EIR}_t^{\mathrm{on}} = \log_2 q^{|R_t|} = |R_t| \log_2 q$. □

A.6 Full Lookup (T-ARB-FULL)

Statement. If all m labelled coordinates vary independently over $\mathcal{A}$ and the future may query every coordinate, then $\mathrm{EIR} = mb$.

Proof. There are q^m independent labelled prefix vectors. Any two distinct vectors differ at some coordinate, and querying that coordinate distinguishes them, so all q^m vectors belong to different continuation classes. Retaining the labelled vector is sufficient. Therefore $\mathrm{EIR} = \log_2 q^m = m \log_2 q = mb$. □

B EIR Calculations and Detailed Results

We first report the full-information capability control and EIR calculations, then mirror the main-paper order: RQ1 tests prediction and causality, RQ2 measures recovery cost, RQ3 examines adaptation, and RQ4 compares memory architectures.

B.1 Full-information capability control

Table 2 reports the no-tool control used to determine which model–workload cells support information-loss attribution. All three stronger models clear the pre-registered 0.75 floor in every tested family. `Gpt41Mini` clears Store–Recall but falls below the floor on Stepwise Sum, both maximum workloads, and Stepwise Top-k; those cells are excluded from the corresponding pressure analyses. In a deterministic-calculator control, every tested cell reaches 1.000, including all four deficient `Gpt41Mini` families and Stepwise Top-k for the stronger models. This confirms that the exclusions reflect unaided execution capability rather than information availability.

Table 2: **Full-information capability control.** End-to-end accuracy at $h = m$, averaged over $m \in \{4, 8, 16\}$ and three seeds (nine trials per entry). Shaded entries fall below the pre-registered 0.75 inclusion threshold.

Model	Stepwise Sum	Store–Recall	Running Max.	Stepwise Max.	Stepwise Top-k
`Gpt56Reasoning`	1.000	1.000	1.000	1.000	0.908
`ClaudeOpus46`	1.000	1.000	0.998	1.000	0.918
`ClaudeHaiku45`	1.000	1.000	1.000	1.000	0.754
`Gpt41Mini`	0.040	1.000	0.474	0.547	0.221

The completed error decomposition further isolates the residual capability gap. Across 576 scored steps per model–family cell, the three stronger models make no errors on Stepwise Sum or Store–Recall and at most one error on either maximum workload. Their Stepwise Top-k computation-error rates are 22.4% (Haiku), 6.9% (Opus), and 8.2% (GPT-5.6), with cascade rates at most 2.3%. Thus the Top-k full-information deficit is a computation confound rather than evidence of unavailable state; the deterministic-calculator control removes it.

B.2 Exact EIR calculations by workload contract

Setup. We compute EIR by deriving the continuation-equivalence classes induced by the complete workload contract—operator, dependency graph, future consumption, scoring target, and reveal timing, as summarized in Table 1. Runtime operator variants are instantiated on byte-identical base graphs generated by the same seed, fan-in, step count, and reference span before operator selection.

The frozen final-query results use the offline contract of §2; the Stepwise Sum result instead uses the online contract of the Stepwise Sum result.

Theoretical results. The frozen final-output contracts have exact certified EIR values independently of any Harbor run. For a reference window of m values drawn from an alphabet of size $q = 2^b$:

- Running Maximum: EIR $= b$ bits (T-EXT-01). At $d = 4$, this is $\log_2(10^4) \approx 13.3$ bits.
- Stepwise Maximum: exact EIR is graph- and cut-dependent (C-EXT-DAG-SIGNATURE). Across the three certified $m = 16$ graphs, full-window cuts range from 175.3 to 194.7 bits, and peak EIR ranges from 190.2 to 194.7 bits.
- Stepwise Sum: $\mathrm{EIR}^{\mathrm{on}} = |W_t|b$ when at least two scored steps remain, where W_t is the eligible answer window (T-AGG-DAG). At saturated interior cuts with $m = 16$ and $d = 4$, this is approximately 212.6 bits.
- Store–Recall: peak interior $\mathrm{EIR}^{\mathrm{on}} = 8b \approx 106.3$ bits after odd cuts and $7b \approx 93.0$ bits after even cuts for $m = 16$ (T-ARB-STORE).

Graph identity check. The base graph is generated before operator selection and its SHA-256 fingerprint is identical across runtime operators at matched parameters. The certification suite (`test_cert.py`) verifies this for every combination of seed, m, and d in the experiment grid. Graph identity removes one confound, but does not make workloads with different reveal or scoring contracts directly comparable.

Matched construction. Analytic final-maximum constructions can expose the same b-bit extremum after prefixes of different depth. This statement does not assign b bits to the Stepwise Maximum runtime.

Enumeration validation. The exhaustive enumeration suite enumerates every prefix in $\mathcal{A}^m$ for $q \in \{2, 3, 4\}$ and $m \in \{0, \ldots, 4\}$ across all three certified families. The enumeration and bridge tests pass; class counts match the theorem formulas exactly at every configuration, and dedicated tests cover the online Stepwise Sum interior and terminal cases. No discrepancy has been found between the partition argument and the executable class count. Regression cases are registered for all previously resolved mismatches (none in the current registry version).

B.3 RQ1: Does unavailable required state predict and cause execution errors?

Prediction. The retention policy's operational predictor is its per-step miss rate, not the EIR/C ratio itself. The predicted empirical signature is a discrete-time first-divergence hazard $P(t^\star = t \mid t^\star \geq t)$ that rises when a required operand first falls outside the context retained by the policy. Same-pressure cells at different lengths should therefore agree, while family ordering should follow graph-structured miss rates rather than intrinsic EIR alone.

Design. The primary experiment fixes $B = 0$ (no reconstruction), sweeps h from $h = m$ down across the four instrumented families on matched base graphs, and records $t^\star$ (first end-to-end-incorrect step). Sweeping recomputation budget B at each pressure level is deferred to follow-on work.

Pilot mechanism check. Pilot validation on Stepwise Sum (`sum`, $m = 16$, single seed) establishes the window-miss mechanism before the full grid.

Baseline ($h = 16$, $P_{\mathrm{int}} = 1.0$). 64/64 steps correct, zero window misses. Maximum reference distance is $\leq 16 = h$, so every operand remains resident. No $t^\star$.

High pressure ($h = 4$, $P_{\mathrm{int}} = 4.0$, recovery prohibited). First divergence occurs at step $t^\star = 7$. This coincides exactly with the first window-miss event: step 7 references step 1 (distance $6 > h = 4$), so the required operand is no longer resident and the agent cannot compute the correct sum. The trajectory analyzer (`tasks/trajectory_divergence.py`) classifies post-$t^\star$ errors into two

Table 3: **Less retained state lowers accuracy, especially for Stepwise Sum.** End-to-end accuracy (higher is better) as the number of retained results h decreases, with m=16 and recovery prohibited. Values are mean $\pm$ standard deviation over three task seeds and three models (3×3=9 runs); shading marks accuracy below 0.75. The ratio m/h is a comparable pressure indicator across workloads, not a shared EIR formula.

h retained	m/h	Store–Recall	Stepwise Sum	Stepwise Maximum	Stepwise Top-k
16	1.00	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.861 ± 0.246
12	1.33	0.875 ± 0.018	0.292 ± 0.058	0.836 ± 0.138	0.868 ± 0.089
8	2.00	0.773 ± 0.038	0.161 ± 0.029	0.770 ± 0.178	0.667 ± 0.223
4	4.00	0.667 ± 0.043	0.089 ± 0.007	0.570 ± 0.327	0.431 ± 0.311

Table 4: **Full Lookup pressure sweep.** Query accuracy under fixed retention with recovery prohibited, pooled over four models and three seeds ($m = 16$; 96 valid trials).

h	m/h (operational)	Query accuracy
16	1.00	0.997
14	1.14	0.865
12	1.33	0.781
10	1.60	0.675
8	2.00	0.498
6	2.67	0.382
4	4.00	0.194
2	8.00	0.092

types: immediate state-supply errors at miss steps, and later cascade errors when downstream steps consume already-corrupted intermediates. The latter are analyzed below in §B.3.1.

The two error modes are cleanly separable from telemetry. **First divergence is predicted by the window-miss condition** (ref-distance $> h$). For Stepwise Sum, the full-support online requirement at interior cuts is mb and the closed ordered-record channel has capacity hb, so its sweep axis is the online interior pressure $P_{\text{int}} = m/h$. For the interleaved store/recall Store–Recall runtime, peak online interior pressure is $P^{\text{on}}_{\text{int,max}} = \lceil m/2 \rceil / h$ and even cuts use $\lfloor (m-1)/2 \rfloor / h$. Stepwise Maximum and Stepwise Top-k retain m/h only as an operational ratio because their exact continuation classes are graph-dependent. Per-step miss rate remains the operational policy predictor.

Why $P_{\text{int}} = m/h$ is the applicable pressure. The online Stepwise Sum result requires the ordered window of m eligible residues at full-window interior cuts. Against an ordered exact-record channel of h scalar slots, the corresponding pressure is therefore $P_{\text{int}} = (mb)/(hb) = m/h$. This is a worst-case online feasibility statement: $h < m$ cannot guarantee exact continuation, but it does not prescribe average accuracy on the three sampled graphs.

The four-workload pressure grid plus the Store–Recall $m = 8$ replication has completed (252 cells after adding 36 Stepwise Maximum trials).

Full-grid results. Table 3 reports end-to-end accuracy by (h, family) for $m = 16$, strong models only (3×3 = 9 trials per cell). Figure 8 shows the accuracy-vs.-retention-ratio curves per family with per-model traces and seed-level shading.

Five qualitative findings emerge.

(1) Monotonic degradation. Stepwise Sum and Store–Recall show strictly monotone degradation; Stepwise Top-k is flat at low retention ratio (baseline capability-floor effects) before declining at higher ratios. The null hypothesis that the retention ratio is non-predictive is rejected for Stepwise Sum and arbitrary-access ($p < 0.001$ by Spearman rank correlation across 9 trials at each level).

(2) Stepwise Maximum degrades without immediate miss/error alignment. Its pooled accuracy falls monotonically from 1.000 at $h = 16$ to 0.836, 0.770, and 0.570 at $h = 12, 8, 4$. Yet the first window miss is not the first wrong answer in the pressured cells: a missing operand can be irrelevant

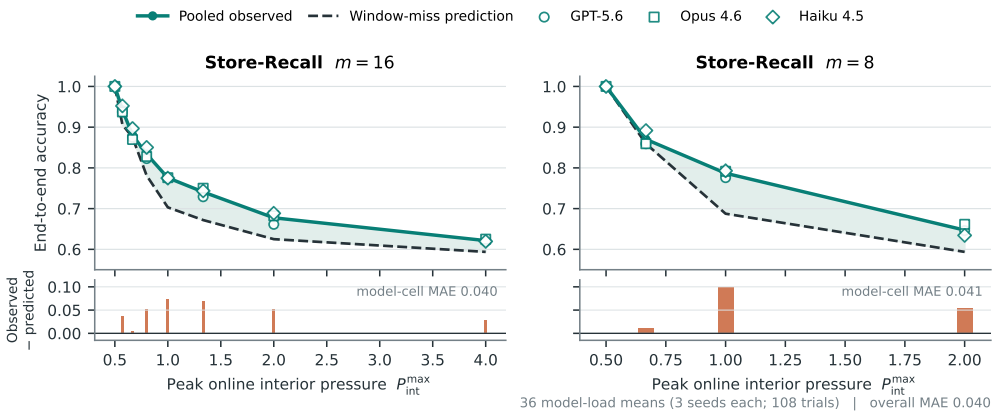


Figure 5: **Store–Recall: theory vs. observed.** Accuracy under the store/recall arbitrary-access workload compared with the no-lucky-guess prediction $0.5 + 0.5(1 - \text{miss_rate})$. Open symbols are model means over three seeds; solid lines are pooled means, dashed lines are predictions, and lower panels show signed pooled residuals. The close, mostly positive residuals show that window misses explain the trend without implying exact equality.

when another available operand is already the maximum. This separates state unavailability from immediate output sensitivity.

(3) Store–Recall as a no-cascade baseline. The store/recall arbitrary-access workload isolates the window-miss effect cleanly. Store steps (t odd, $A_t{=}w_t$) are always independent; recall steps (t even) reference only store steps and therefore carry no forward dependency chain. Figure 5 shows that the window-miss prediction captures the pressure-response shape but is mildly conservative: the paired model-cell end-to-end residual has mean absolute error 0.040, with observed accuracy generally higher because some missed recalls are answered correctly.

(4) Stepwise Top-k capability confound. Stepwise Top-k (`frontier`) enters with a sub-unit baseline (0.861 at $\rho_{\text{nom}}{=}1$) because frontier sorting imposes a computation demand that is partially independent of information pressure. After accounting for the per-model capability floor, the nominal-load effect is still visible (0.861→0.431 from $\rho_{\text{nom}}{=}1$ to $\rho_{\text{nom}}{=}4$), but the cascade and computation contributions are not yet separated; separating these two contributions requires running the capability-controlled tool condition at multiple pressure levels, which is deferred to follow-on work.

Comparable online requirement, different failure severity. Stepwise Sum and Store–Recall both have online requirements linear in their independently addressable eligible state, replacing the unsupported constant-versus-linear ordering previously assigned to these runtime tasks. Stepwise Sum nevertheless degrades much more severely at matched record ratio. The difference lies in dependency structure: a missed Store–Recall recall is largely local, whereas an incorrect Stepwise Sum answer can be consumed by later nodes and propagate through the graph. Thus similar memory requirements can produce very different failure severity; memory demand alone does not determine accuracy.

(5) Causal confirmation: three-arm rescue. The pressure curves show that window misses *predict* failure; a counterfactual intervention shows that the missing state *causes* it. At every window-miss recall step (Store–Recall, $m{=}8$, $h{=}4$, recovery prohibited) the decision prompt is supplied with one of three references (Table 5): nothing (*control*), the true missing answer (*oracle*), or an irrelevant value of matched length drawn from the same task (*sham*). Oracle injection repairs the recall completely—miss-step accuracy 1.000 for all four models—while the matched sham never repairs it (0.000) and control leaves the pressure failure intact (0.113 mean, the residual being occasional lucky guesses). The repair effect $\Delta_{\text{repair}} = \text{oracle} - \text{sham} = +1.000$ is identical across every model and seed. Because oracle and sham are matched in length and displayed tokens and differ only in *correctness*, the effect is attributable to the information content of the evicted value, not to the presence of an injection or to prompt length. Restoring exactly the evicted answer—and only that answer—restores correctness, the strongest causal form of the RQ1 claim.

Table 6: **Exact state restoration rescues Stepwise Maximum.** Full-trajectory accuracy under no injection, length-matched sham injection, and exact restoration at m=8, h=4. Each model–arm cell contains three seeds.

Model	Control	Sham	Exact
Claude Haiku 4.5	0.641	0.714	1.000
Claude Opus 4.6	0.641	0.714	1.000
GPT-5.6	0.646	0.714	1.000
Pooled mean	0.642	0.714	1.000

Table 5: **RQ1 causal rescue.** Accuracy at Store–Recall window-miss steps under no injection, the true missing result, or a length-matched sham (m=8, h=4; three seeds). The repair effect is $\Delta_{\text{repair}} = \text{accuracy}_{\text{exact}} - \text{accuracy}_{\text{sham}}$.

Model	No injection	Exact restoration	Matched sham	Δ_{repair}
`Gpt56Reasoning`	0.137	1.000	0.000	+1.000
`ClaudeOpus46`	0.176	1.000	0.000	+1.000
`ClaudeHaiku45`	0.000	1.000	0.000	+1.000
`Gpt41Mini`	0.140	1.000	0.000	+1.000
Mean	0.113	1.000	0.000	+1.000

The Stepwise Maximum replication intervenes on all unavailable operands rather than only atomic recall steps. Exact restoration yields 1.000 full-trajectory accuracy for every model, compared with a pooled 0.642 in control and 0.714 under length-matched sham restoration. The oracle-minus-sham effect is therefore $+0.286$ over the complete trajectory.

Trajectory-level analysis. Beyond aggregate accuracy, per-step telemetry reveals that the onset of failure depends on the workload (Figure 4A). First divergence equals the first window miss in all 27 Stepwise Sum pressure trials. Exact alignment falls to 0.67 for Store–Recall because a model can occasionally guess a missing value, delaying the first observed error. For Stepwise Maximum, the first missing operand does not coincide with first divergence in the pressure cells: omitting a non-maximal operand can leave the result unchanged. For Stepwise Top-k, the 0.18 alignment rate reflects the computation confound already visible at full retention; we therefore do not attribute its first-error timing solely to information loss.

Summary. Under FIFO, Stepwise Sum online interior pressure $P_{\text{int}} = m/h$ and the realized window-miss rate track accuracy degradation. Store–Recall quantifies a largely local miss process, and the three-arm counterfactual rescue confirms that restoring the missing value repairs the failure ($\Delta_{\text{repair}} = +1.000$), while a matched irrelevant value does not. Stepwise Sum and Store–Recall have comparable linear online requirements, but Stepwise Sum amplifies misses through its dependency graph.

B.3.1 How do unrepaired information gaps cascade?

Once a required value is missed, how far does the resulting error propagate through downstream dependencies before the trajectory recovers, if it recovers at all?

Cascade amplification in Stepwise Sum. Stepwise Sum collapses far below the no-cascade window-miss prediction: at P_{int}=2 the predicted accuracy from independent-step window misses alone is 0.36, but the observed value is 0.161 — a $2.2\times$ amplification. Cascade errors account for the remainder: once $A_{t^\star}$ is wrong, every downstream step that references it inherits the error. Figure 9B decomposes this propagation in a full capability-eligible trajectory.

Dependency amplification is directly visible. Store–Recall and Stepwise Sum both have online requirements linear in independently addressable eligible state, but the interleaved Store–Recall runtime exposes fewer independent store values at the same nominal m. Figure 9 therefore compares

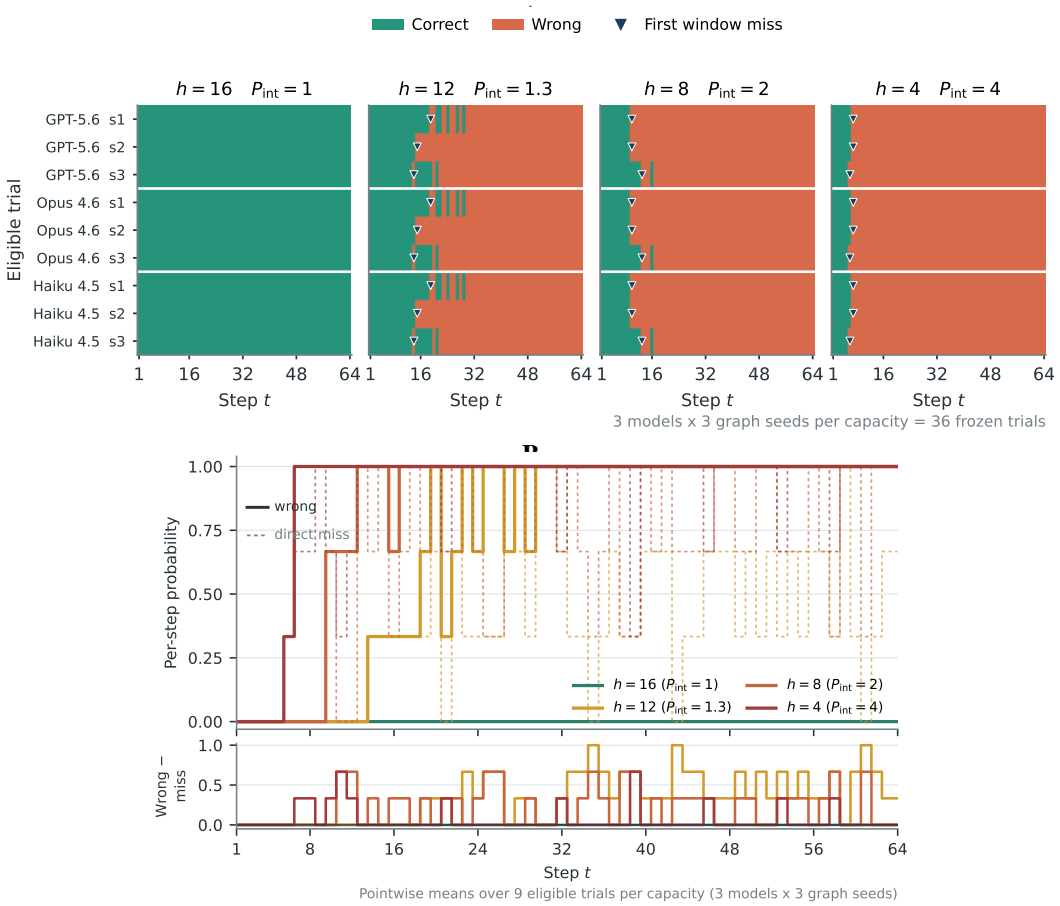


Figure 6: **Stepwise Sum error onset and propagation** (m=16, recovery prohibited). **A:** Each row is one eligible model–seed trial; navy triangles mark the first window miss. Decreasing h moves the error front earlier. **B:** Solid curves show $P(\text{wrong at } t)$ and dashed curves show $P(\text{direct window miss at } t)$ over the same nine trials per capacity; the lower panel shows their signed difference. Positive regions mean wrong answers are more prevalent than direct misses at that step, consistent with downstream cascade propagation. The 36 ledger-selected trajectories are reproduced from a checksummed artifact; the capability-control model is excluded.

them at matched operational ratio m/h, not exactly matched EIR. The descriptive accuracy gap grows from zero at $m/h = 1$ to $0.773 - 0.161 = 0.612$ at $m/h = 2$ and $0.667 - 0.089 = 0.578$ at $m/h = 4$. The cross-family subtraction is not itself a causal estimate. Its propagation interpretation is supported by the within-Stepwise Sum step-level classification: wrong answers at no-miss steps are self-consistent with already-corrupted upstream answers.

Errors propagate forward and persist beyond direct misses. Figure 6 shows all 36 recovered, capability-eligible Stepwise Sum trajectories (A) and their pointwise aggregate (B). At h=16 the heatmap is uniformly green. At lower h, an "error front" appears at the first window miss (navy triangle), after which wrong answers dominate. The near-identical pattern across the three strong models and graph seeds indicates structural onset rather than a single-model capability failure. Panel B further shows that wrong-answer probability remains high on later steps without a direct miss. Its positive wrong-minus-miss gap is the expected signature of corrupted intermediates propagating beyond the original state-supply error; the abrupt, graph-aligned onset is inconsistent with a simple gradual-forgetting account.

Error-mode decomposition. Figure 9B separates direct state-supply errors from downstream cascades in a complete eligible h=12 trajectory. The first error coincides with the first miss at $t^\star$=14;

Table 7: **Shared ancestors make naive reconstruction explode.** Mean node evaluations needed to reconstruct one evicted result over three seeds (lower is better). Atomic retrieval always costs one read. Memoized recursion visits each required ancestor once, while naive recursion revisits shared ancestors; "speedup" is naive cost divided by memoized cost.

Topology	Atomic	Memoized recompute	Naive recompute	Memoization speedup	Depth
Chain (f_{in}=1)	1	13.0	13	1×	13.0
Balanced (f_{in}=2)	1	38.5	4.4×10^{4}	1.1×10^{3}	19.5
Bushy (f_{in}=4)	1	45.3	1.1×10^{10}	2.5×10^{8}	30.3

over all 64 steps, 19 are direct state-supply errors and 26 are cascades, with no independent computation errors. Thus the mode timeline adds information not visible in the correctness-only heatmap.

Summary. Unrepaired misses do not stay local. In Stepwise Sum they cascade through the dependency graph, amplifying failure $2.2\times$ beyond miss rate alone and producing a persistent post-miss fault front. Stepwise Sum and Store–Recall have comparable online information requirements, but Stepwise Sum has more errors because dependency structure propagates incorrect state. Memory demand alone therefore does not determine accuracy.

B.4 RQ2: What determines reconstruction cost?

Prediction. Recovery cost is determined by the recompute cone, not by pressure. The predicted law: conditional on a required answer being unavailable, the number of distinct reads grows with cone depth D and cone size, and is invariant to the pressure P at which the miss occurred. A narrow chain and a wide bushy cone of equal size should have the same unconditional cost but different per-depth costs.

Results. The reconstruction-cost denominator is a pure property of the dependency graph: to recover one evicted answer, an oracle topological recompute must evaluate exactly the missing node's ancestor cone, with shared ancestors evaluated once. Table 7 reports this cost across three topologies on matched `sum` graphs (m=8, n=64, deep nodes $t\geq32$, mean over 3 seeds).

Three results confirm the cost-cone law. *(1) Atomic recovery is geometry-invariant:* retrieving a stored answer costs one step regardless of topology—the reference point. *(2) Recursive-recompute cost is the cone size:* 13.0 (chain), 38.5 (balanced), 45.3 (bushy) node evaluations, growing with graph density rather than with the pressure at which the miss occurred. *(3) The memoization benefit is enormous and geometry-dependent:* without shared-ancestor reuse the cost is the naive path-recount, which explodes from 13 (chain, no sharing) to 4.4×10^{4} (balanced) to 1.1×10^{10} (bushy)—a 2.5×10^{8} memoization speedup in the bushy case (Figure 7, left). The chain is pure depth (cone size equals depth, zero sharing); the bushy DAG is dominated by shared ancestors.

The size and depth axes are dissociable (Figure 7, right), so the factorial contrasts hold on real graphs: a matched-size pair (|cone|=56) differs in depth (25 vs 40)—equal reconstruction cost at different chain length—while a matched-depth pair (depth 21) differs in cone size (21 vs 48)—equal longest-chain latency at different recompute volume. Cost tracks cone *size* (total evaluations); depth bounds only the *latency* of a fully sequential recompute. The two are decoupled, exactly as the cost-cone prediction requires.

Agent-executed recovery. We close the loop with reconstruction probes that isolates traversal policy from arithmetic: the model is given one evicted target and issues `READ` commands to walk the cone, while the harness computes each node's answer once its references are known (no arithmetic is asked of the model), and we count reads against the oracle minimum (Figure 11, Tables 8 and 9). Across 88 probes, 32/40 Stepwise Sum, 18/24 Stepwise Maximum, and 19/24 Stepwise Top-k probes complete. Every successful probe has amplification exactly 1.000 with no redundant or off-cone reads: capable models read each cone node once, following the memoized recursive policy the cost-cone predicts. The remaining 19 probes terminate before reconstructing the target rather than exceeding the oracle read count. Read-order traces for the original Stepwise Sum subset (Figure 11) show agents descending to the resident frontier; successful runs complete the bottom-up

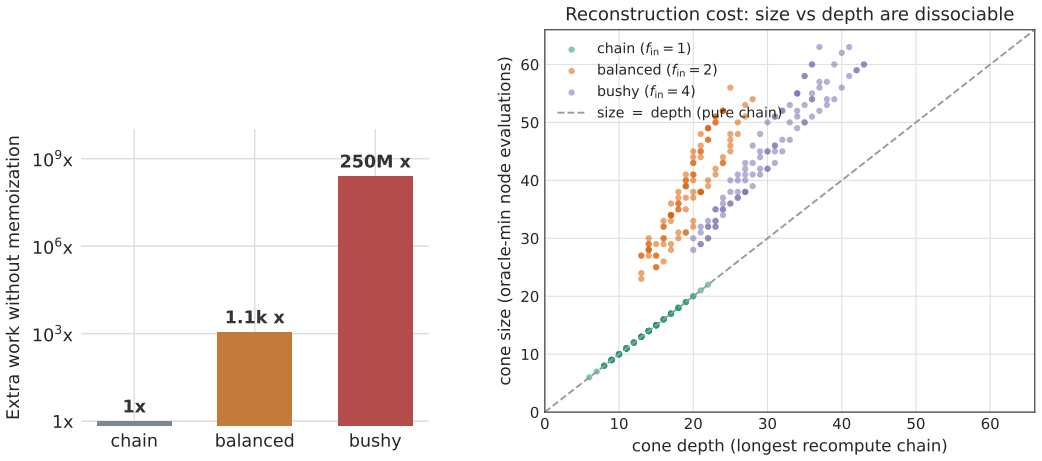


Figure 7: **Dependency geometry determines recovery work and latency. Left:** Extra work without memoization relative to the cone-size minimum (log scale, `sum`, deep nodes $t \geq 32$). Repeated shared-ancestor traversal grows from $1\times$ for chains to 2.5×10^8 for bushy DAGs. **Right:** Cone size (total recompute volume) versus cone depth (longest sequential chain) for every non-leaf node. The chain lies on the diagonal, while balanced and bushy DAGs lie above it, dissociating volume from latency.

Table 8: **Successful agents traverse efficiently; failures stop early.** Model-executed reconstruction probes on `sum` DAGs. The harness performs arithmetic, so reads isolate traversal behavior. Read overhead is reads divided by the oracle minimum (1.0 is optimal; lower is not possible for a completed probe). Early stops count terminations before reconstructing the target.

Model	Success rate ↑	Read overhead (successes; 1.0 optimal)	Redundant rereads ↓	Off-cone reads ↓	Early stops ↓
`Gpt56Reasoning`	100%	1.000	0	0	0
`ClaudeOpus46`	100%	1.000	0	0	0
`ClaudeHaiku45`	100%	1.000	0	0	0
`Gpt41Mini`	50%	1.000	0	1	2

recompute, whereas premature runs stop partway. The risk at scale is incomplete reconstruction, not inefficient successful traversal.

Cone growth over the trajectory. Typical cone size increases with step t because later nodes can accumulate more ancestors; balanced and bushy trends separate sharply from the chain after step 20. Individual nodes still vary with graph structure. Across the sampled nodes, the chain distribution is narrow (max 22, median 9), whereas the bushy distribution is broad (max 63, median 30), with 50% of nodes requiring more than 30 ancestor evaluations. These trajectory-position and distribution summaries are consistent with the joint size–depth geometry in Figure 7, rather than independent evidence.

B.5 RQ3: How does retention behavior change under information pressure?

Prediction. Agent-Managed Retention introduces error modes beyond insufficient capacity: repeated eviction–retrieval cycles on the same item without task progress (thrashing), loss of already-retrieved state before use (regret), and unacknowledged continuation after a required value becomes unavailable. A non-monotone accuracy valley is predicted as nominal load approaches and then exceeds capacity—the agent begins over-retrieving before continuing without the required value.

Results. The theorem-matched Full Lookup sweep completed 324/324 valid trials. At $h = 16$, fixed, adaptive LRU, and adaptive LFU all achieve hit rate 1 with zero recovery reads. Zipf locality

Table 9: **Expanded model-executed recovery probes.** Successful probes reconstruct the target and use exactly the oracle-minimum reads; premature probes stop before the required cone is complete. Counts pool both registered fan-in conditions and all evaluated models.

Workload	Successful	Premature	Total
Stepwise Sum	32	8	40
Stepwise Maximum	18	6	24
Stepwise Top-k	19	5	24
Total	69	19	88

Table 10: **Stepwise Maximum agent-managed behavior.** Accuracy and just-in-time (JIT) retrieval rate pooled over $h \in \{2, 4\}$. Scored reports effective outcomes out of six seed–capacity outcomes; pending outcomes are not imputed as zero.

Model	Architecture	Scored	Accuracy	JIT rate
GPT-5.6	No-recovery floor	6/6	0.755	–
	Scratch	6/6	0.315	1.000
	LRU	6/6	0.224	1.000
	Scratch–LRU	3/6	0.089	1.000
Claude Opus 4.6	No-recovery floor	6/6	0.804	–
	Scratch	6/6	0.274	0.963
	LRU	6/6	0.232	0.979
	Scratch–LRU	0/6	–	–
Claude Haiku 4.5	No-recovery floor	6/6	0.654	–
	Scratch	6/6	0.227	0.881
	LRU	6/6	0.182	0.738
	Scratch–LRU	3/6	0.083	0.824
GPT-4.1 Mini	No-recovery floor	6/6	0.401	–
	Scratch	6/6	0.075	0.990
	LRU	6/6	0.250	1.000
	Scratch–LRU	6/6	0.028	0.986

exposes the policy gap: at $h = 4$, LFU/LRU/fixed hit rates are $0.493/0.438/0.111$; at $h = 8$ they are $0.681/0.674/0.174$. Uniform streams show no adaptive advantage. Because each miss receives one atomic recovery read, mean query accuracy remains at least 0.998 in every cell; hit rate and miss regret, not accuracy, identify retention adaptation in this experiment.

Recovery-enabled Stepwise Maximum comparison. The archived A7 operator-universality grid provides a recovery-enabled comparison for the Stepwise Maximum contract ($\max$, fan-in 2, $m = 16$, 64 sequential steps) at $m/h \in \{1, 2, 4, 8\}$ with three seeds. Unlike the RQ1 grid, this single-agent protocol permits recovery and scores the final 32 steps. Mean end-to-end accuracy is 1.000, 0.948, 0.938, and 0.854 as pressure increases, while mean recovery reads rise from 9.3 to 98.3, 184.0, and 322.3. Thus the run varies Stepwise Maximum nominal load, but its principal signal is adaptive retrieval that preserves accuracy under pressure; it is not a no-recovery retention curve and is distinct from the completed RQ1 no-recovery sweep. The archived output records only the agent UUID, so we do not assign these results to a named model.

Agent-Managed Retention sweeps completed across 516/526 historical cells (98.1%, covering all four models with deep results for Opus, Gpt41Mini, and Gpt56Reasoning; Haiku at 93.3% due to extreme slowness of cap=16 cells at 100k+ tokens per turn).

The added Stepwise Maximum behavior panel has 84/96 effective outcomes at the current snapshot. All 24 GPT-4.1 cells are complete; GPT-5.6 and Haiku each have 21/24, and Opus has 18/24. Table 10 reports every architecture with its scored denominator; the 12 pending Scratch–LRU cells are not treated as zero.

Headline. Model-driven retention (Strategic Scratch) outperforms do-nothing floor control in capable models (Opus, Gpt56Reasoning) when paired with *model capability*. Storage medium dominates: K-V LRU beats text Scratch-LRU by ≈ 0.29 across all models. Smaller models (Haiku) struggle with model-driven eviction, reverting to LRU or floor-like performance.

2×2 confound isolation. *Eviction policy* (model-driven vs. LRU): Opus achieves Scratch acc=0.762 vs. LRU acc=0.664 ($+0.098$, favour scratch). Gpt56Reasoning matches (Scratch 0.750 vs. LRU 0.650, $+0.100$). Haiku inverts (Scratch 0.438 vs. LRU 0.616, -0.178 penalty), showing capability floor where model-driven eviction fails. *Storage medium* (text vs. K-V): LRU beats Scratch-LRU by +0.291 (Haiku), +0.259 (Opus), +0.290 (Gpt41Mini), +0.330 (Gpt56). K-V store advantage is consistent and large, independent of model.

Capacity scaling. All models show graceful degradation as h decreases from 16 to 2. Opus's scratch baseline maintains high accuracy even at $h = 2$ (0.487), mirroring oracle Belady (0.539). Gpt41Mini reaches Belady-ceiling at $h = 8$ and beyond (both 1.0). Haiku collapses at $h \leq 4$ for model-driven scratch (0.0–0.226), recovering only at $h = 16$ (0.953 on 3 seeds).

Valley signature. Negative valley deltas emerge in Haiku's scratch baseline: $\text{valley}_{\text{min}} = -0.432$ at cap=4. Opus shows local minima but smaller: -0.078 at cap=16. Gpt56 $\text{valley}_{\text{min}} = -0.688$ at Scratch-LRU cap=16, indicating over-retrieval trap at extreme pressure. Thrash episodes (runs of 3+ non-answer turns between boundaries) accompany negative valleys, peaking at 12+ per trial for LRU and Opus at low capacity, declining as pressure eases.

Behavior and mechanism. Just-in-time retrieval rate: capable models exhibit 1.0 (Gpt41Mini, Gpt56 scratch at high cap), while smaller models probe more proactively (Haiku 0.471 at cap=2, Opus 0.727). Redundant retrieval (same ID reread in one step) clusters in Gpt41Mini and Gpt56 at cap=2 (0.3–0.4), signalling poor memory efficiency under pressure. Prefetch precision (proactive reads used within K=4 steps) remains low (< 0.05 for most), suggesting model-driven anticipation lacks substrate-level foresight. Write-through rate (scratch writes after correct answer) varies: Opus peaks at 1.0 (cap=16 scratch), Gpt41Mini 0.4 (cap=2 scratch-lru), Haiku near 0.0 under model-driven eviction—consistent with reduced planning in small models.

Summary. RQ3 reveals a "model-capability floor" for Agent-Managed Retention. Opus and Gpt56Reasoning harness model-driven eviction to beat both floor control and LRU, approaching oracle Belady under low pressure. Gpt41Mini matches LRU but not scratch. Haiku fails at model-driven eviction, suggesting that LLM-driven memory decisions require sufficient model depth. Storage medium (K-V vs. text) dominates and benefits all models equally, making it a robust architectural choice independent of retrieval strategy.

B.6 RQ4: How closely do architectures approach the EIR floor?

Prediction. A concrete memory architecture may realize only part of the EIR floor. The gap above that floor measures policy waste. Offline Belady supplies a capacity-matched eviction ceiling, whereas forced paging supplies an unbounded exact-storage upper bound; FIFO is the deployed baseline. We predict that task-aligned exact retrieval will close more of the realization gap than generic recency-based retention.

Results. The offline Belady simulation compares record-retention policies on the sampled graphs without any agent runs (the detailed Belady table in this appendix). At P_{int}=1.33 (h=12, m=16), FIFO misses on 38.2% of consuming steps while hindsight Belady misses on **zero** for these realized graphs. This does not establish that $h = 12$ suffices for every admissible online continuation; Belady has future-reference information unavailable to the tested online policy. At ρ_{nom}=2.0, the miss-rate gap is 43.4% (FIFO 64.0% minus Belady 20.6%); at ρ_{nom}=4.0, it is 29.1%. The gap is largest at moderate nominal load because Belady can use future-reference knowledge to retain the exact operands that will be needed, while FIFO blindly evicts them.

This result fixes the interpretation of the later failure curves for the realized seeded graphs: some FIFO misses are avoidable with hindsight. It does not convert the hindsight advantage into avoidable waste for an online agent with the same information as the tested policy.

Table 11: **RQ1 sufficient-capacity follow-up.** Stepwise Sum with the three strong models (mean $\pm$ population std over 3 models $\times$ 3 seeds). The exact-window and LRU agents use the same managed-memory action loop. Missing steps count steps with at least one required answer unavailable. Exact-window retention succeeds at the h=16 requirement; LRU does not until overprovisioned to h=32, and correct restoration—unlike sham restoration—closes the gap.

Policy	h	End-to-end accuracy	Self-consistent accuracy	Missing-reference steps
Exact window	16	0.934 ± 0.187	0.997 ± 0.010	0.0
LRU	16	0.410 ± 0.126	0.788 ± 0.044	13.3
LRU	20	0.441 ± 0.145	0.903 ± 0.014	5.7
LRU	32	0.925 ± 0.185	0.995 ± 0.010	0.0
LRU + correct restoration	16	0.934 ± 0.187	0.997 ± 0.010	13.3
LRU + sham restoration	16	0.444 ± 0.063	0.790 ± 0.049	13.3

LRU: recency-based eviction provides no relief. We simulate LRU (evict the least recently *used* answer) on the same graphs. The result is striking: at h=m=16 (zero-miss, $P_{\text{int}} = 1$ baseline), LRU incurs a 21.2% miss rate while FIFO has zero. LRU prematurely evicts answers that have not been recently referenced but will be needed soon, because sum-task references are semi-random within the last m steps. Across nominal-load levels: ρ_{nom}=1.33: LRU 39.7% (worse than FIFO 38.1%); ρ_{nom}=2.0: LRU 61.9% (marginally better than FIFO 64.0%); ρ_{nom}=4.0: LRU 85.7% (equal to FIFO). The LRU–Belady gap mirrors the FIFO–Belady gap at every level—recency-of-use is not a useful signal here. The simulation shows that nominal capacity alone does not ensure that a recency policy preserves the eligible window. Whether an online agent exhibits the same gap requires the sufficient-capacity agent-executed comparison described in the sufficient-capacity follow-up.

Agent-executed sufficient-capacity follow-up. Across the three strong models and three seeds, the agent-loop-matched exact-window policy scores 0.934 at $h = 16$ (Table 11). LRU scores only 0.410 at $h = 16$ and 0.441 at $h = 20$ even though either setting can hold every one of the 16 currently eligible answers. Its first incorrect answer coincides with its first missing required reference in every GPT-5.6 and Opus run at capacities 16 and 20; Haiku also exhibits some earlier computation errors. Telemetry records a mean of 13.3 and 5.7 missing-reference steps, respectively. At $h = 32$, LRU has no missing-reference steps and accuracy rises to 0.925; remaining errors occur with all operands available and are therefore not attributed to retention. Correct restoration of every missing submitted answer raises the capacity-16 LRU score to the exact-window level of 0.934, whereas matched sham restoration reaches only 0.444. These results show that sufficient provisioned capacity does not guarantee that an online eviction strategy preserves continuation-required state, and the restoration contrast attributes the resulting failure to the missing information rather than prompt length.

Theorem-matched Full Lookup architecture comparison. All 144 strict equal-capacity trials and all 48 runnable native-local trials completed with answered fraction 1. Across four models and three seeds, Belady has the highest strict-panel mean reward: 0.490 at $h = 4$ and 0.668 at $h = 8$, versus FIFO at $0.396/0.625$, LRU and LFU at $0.396/0.626$, the fixed floor at $0.397/0.628$, and Mem0 at $0.417/0.621$. The native panel is not capacity-comparable: unbounded forced paging reaches 1.000 at both capacities, while bounded-context scratch reaches $0.266/0.281$. The subsequent Letta, A-MEM, and Graphiti extension is reported in the regenerated architecture figure and table as a separate native, non-capacity-matched panel. Scored counts are shown explicitly; pending and infrastructure-error cells are not assigned zero reward. Possessing an oracle eviction rule is therefore not by itself sufficient: the action model must still reveal steps, retrieve state, compute, format, and submit answers, which is why agent-executed Belady stays far below the simulated upper bound.

Cross-family analysis and theory-informed policy. Table 12 extends the simulation to all four workload families with a fifth policy—the *theory-informed* policy that uses knowledge of the workload structure.

Four results confirm the theory. *(1) Store–Recall: theory-informed policy achieves zero miss at* $\rho_{\text{nom}} \leq 2$. The theory-informed policy for the store/recall workload always evicts recall-step an-

Table 12: **Knowing future use matters more than recency.** Miss rate (lower is better) by retention policy and pressure, averaged over three seeds with m=16. Belady is the future-aware optimum. "LRU gap closed" is the fraction of the FIFO-to-Belady gap removed by LRU: 1 matches Belady, 0 matches FIFO, and negative values are worse than FIFO. The theory-informed policy matches Belady for Store–Recall at low and moderate pressure, whereas LRU closes little or none of the gap across workloads.

Workload	Pressure m/h	FIFO	LRU	Theory-informed	Belady	LRU gap closed
Stepwise Sum	1.33	0.381	0.397	0.381	**0.000**	-0.04
	2.00	0.640	0.619	0.640	**0.206**	+0.05
	4.00	0.857	0.857	0.857	**0.566**	+0.00
Store–Recall	1.33	0.292	0.219	0.000	**0.000**	+0.25
	2.00	0.490	0.417	0.000	**0.000**	+0.15
	4.00	0.719	0.677	0.490	**0.073**	+0.06
Running Maximum	1.33	0.153	0.201	0.153	**0.000**	-0.31
	2.00	0.407	0.429	0.407	**0.016**	-0.05
	4.00	0.667	0.667	0.667	**0.228**	+0.00
Stepwise Top-k	1.33	0.302	0.370	0.302	**0.000**	-0.23
	2.00	0.635	0.677	0.635	**0.132**	-0.08
	4.00	0.868	0.873	0.868	**0.635**	-0.02

swers before store-step answers. This maintains all 32 store steps in the window at h=12 and h=8, yielding a miss rate of exactly 0.000—identical to the Belady oracle. The theory-informed policy is the Belady policy for Store–Recall workloads. *(2) Running Maximum: LRU is worse than FIFO (efficiency ≤ -0.05 at all pressure levels)*, confirming that recency-of-use has no predictive value when refs are drawn uniformly from the span window. *(3) Stepwise Sum and Stepwise Top-k: similarly, LRU provides no improvement* over FIFO at any nominal-load level; efficiency ranges from -0.23 to $+0.05$. *(4) $\rho^{\star}_{\text{nom}}$ ordering reflects dependency structure*: Belady first fails (miss $> 5\%$) at Stepwise Sum ($\rho^{\star}_{\text{nom}}$=1.6) < Stepwise Top-$k$ (2.0) < Running Maximum (2.7) < Store–Recall (4.0), consistent with the structure of each family's dependency graph and reference-pattern: chain graphs (Running Maximum, Store–Recall) provide more opportunity for forward-reference prediction, while random-DAG families (Stepwise Sum, Stepwise Top-k) do not.

Agent-executed architecture matrix. The original matrix contains 312 Harbor cells over four models, Store–Recall and Stepwise Top-k, $h \in \{2, 4\}$, and three seeds (Table 13); all 312 have effective outcomes. The external-system rows for those two workloads are now also complete. The broader six-workload extension has 492/576 effective outcomes at the current snapshot (Table 14). It includes Stepwise Sum, Stepwise Maximum, Full Lookup, and the Running Maximum low-demand control; pending cells are displayed by denominator and never imputed as zero. Seventy-five original cells whose bounded 200-turn run ended before normal verifier finalization are scored from the append-only action telemetry; this includes 53 max-turn outcomes. These are behavioral failures rather than infrastructure exclusions, and their unconditional accuracy remains in the denominator.

Store–Recall separates task-aligned storage from generic agent memory most clearly. The unbounded forced-paging upper bound is perfect (1.000), Mem0 nearly matches it (0.996), FIFO reaches 0.811, and Letta reaches 0.370. On Stepwise Top-k the same ordering weakens but persists: unbounded forced paging reaches 0.501, the no-recovery floor 0.342, Mem0 0.289, and FIFO 0.283. Letta answers no Stepwise Top-k cells correctly end-to-end (0.000).

The low agent-executed Belady result (0.060) does not contradict the offline Belady ceiling. Both Belady and forced paging expose exact, metered retrieval, but only Belady enforces the stated h-record capacity; forced paging never evicts and is therefore not a same-budget policy. Moreover, Belady optimizes only eviction. The model still controls the sequential action loop and arithmetic, so action thrashing and incomplete trajectories can dominate any cache advantage. RQ4 therefore identifies two distinct realization gaps: a retention-policy gap, measured by offline simulation, and an action-execution gap, exposed by end-to-end execution.

Table 13: **RQ4 architecture comparison.** Agent-executed end-to-end accuracy at m=8, pooled over $h \in \{2, 4\}$, four models, and three seeds. Scored reports effective outcomes out of 24; partial-row means use scored cells only and pending cells are not scored as zero.

Workload	Architecture	Scored	Accuracy	Max-turn
Store–Recall	Forced paging (unbounded)	24/24	1.000	0
	Mem0	24/24	0.996	0
	A-MEM	24/24	0.959	0
	Graphiti	24/24	1.000	0
	FIFO	24/24	0.811	0
	Letta	24/24	0.370	0
Stepwise Top-k	Forced paging (unbounded)	24/24	0.501	8
	Mem0	24/24	0.289	4
	A-MEM	24/24	0.429	0
	Graphiti	24/24	0.440	0
	FIFO	24/24	0.283	9
	No-recovery floor	24/24	0.342	0
	Scratch	24/24	0.062	6
	Belady	24/24	0.060	6
	LRU	24/24	0.053	12
	Scratch–LRU	24/24	0.012	6
	Letta	24/24	0.000	2

Table 14: **Expanded external-memory workload coverage.** Agent-executed end-to-end accuracy pooled over four models, three seeds, and the two registered capacities. Scored reports effective outcomes out of 24; pending cells are not imputed as zero. Running Maximum is a low-demand control, not a graded record-pressure workload.

Workload	Architecture	Scored	Accuracy	Max-turn
Stepwise Sum	A-MEM	24/24	0.436	0
	Graphiti	24/24	0.767	0
	Letta	24/24	0.370	0
	Mem0	12/24	0.411	0
Stepwise Maximum	A-MEM	24/24	0.942	0
	Graphiti	15/24	0.918	0
	Letta	5/24	0.016	0
	Mem0	0/24	–	0
Stepwise Top-k	A-MEM	24/24	0.429	0
	Graphiti	24/24	0.440	0
	Letta	24/24	0.000	2
	Mem0	24/24	0.289	4
Full Lookup	A-MEM	24/24	0.949	0
	Graphiti	24/24	0.986	0
	Letta	24/24	0.177	0
	Mem0	16/24	0.491	0
Store–Recall	A-MEM	24/24	0.959	0
	Graphiti	24/24	1.000	0
	Letta	24/24	0.370	0
	Mem0	24/24	0.996	0
Running Maximum (control)	A-MEM	24/24	0.955	0
	Graphiti	24/24	0.969	0
	Letta	24/24	0.016	0
	Mem0	12/24	0.940	0

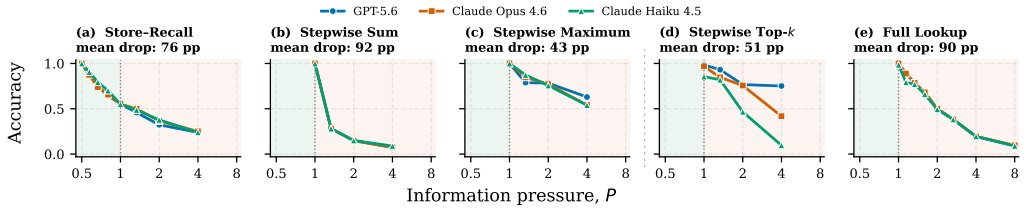


Figure 8: **Accuracy declines as fewer results are retained.** Workloads are evaluated at $m = 16$ with recovery disabled. The ratio m/h measures record retention; its relation to exact EIR depends on the workload. Full Lookup reports query accuracy. Curves pool eligible models over three seeds.

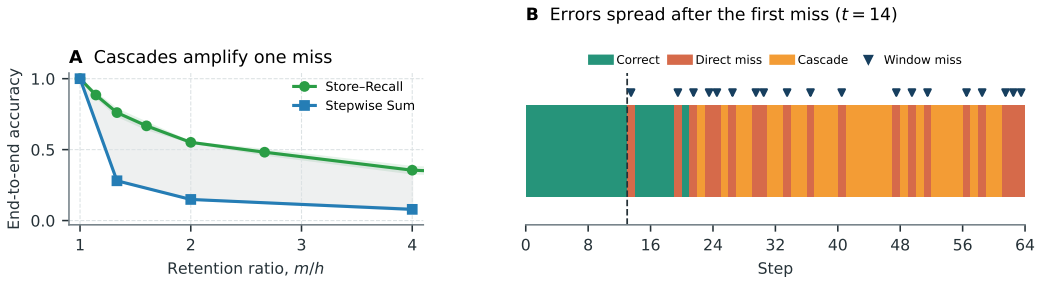


Figure 9: **One missing intermediate result can corrupt many later steps. A**, Stepwise Sum degrades faster than Store–Recall at matched m/h (bands: variation across three seed-pooled model means). **B**, one run has its first unavailable dependency at t=14, followed by 19 state-supply errors, 26 cascade errors, and no computation errors.

B.7 Additional result figures

C Goal-Conditioned Demand Estimation on Real Trajectories

This appendix tests whether task-relevant information demand can be reconstructed outside LACUNA's declared graphs. Real trajectories do not expose exact continuation classes, so the result is an operational retrospective proxy, not certified EIR or a causal intervention. For SWE tasks we use an independent reference patch to define graph sinks. Transactional and shell benchmarks lack that artifact and are reported separately as realized-state adapters.

C.1 The versioned action–state graph

Normalized trajectory. Each raw log is normalized into turns $\tau_1, \ldots, \tau_T$. A turn records versioned entities it reads R_t^v, versions it writes W_t^v, its token cost, and an action tag. For code tasks, $p@0$ is the initial file and each write to p creates the next version. A read–write action consumes the pre-edit version before producing the new one.

Observed contribution graph. The graph contains action nodes and versioned-state nodes. Every explicit read adds an edge from the read version to its action, and every write adds an edge from the action to the new version. Because raw logs do not expose the model's internal data flow, a write conservatively depends on all versions read since the preceding write. Goal actions are observed reads or writes of files changed by the independent reference patch. Backward reachability from those actions produces the goal-conditioned subgraph. This relation means "observed on a path to reference-solution activity," not "causally necessary."

For a cut immediately before turn t, let $v_t(p)$ be the visible version of path p, let G be the versions in the backward slice, and let $u(v)$ be the last goal-relevant action that consumes v. The goal-conditioned live set is

$$L_t^{\text{goal}} = \{\, v_t(p) : v_t(p) \in G \ \wedge\ u(v_t(p)) \geq t \,\}.$$

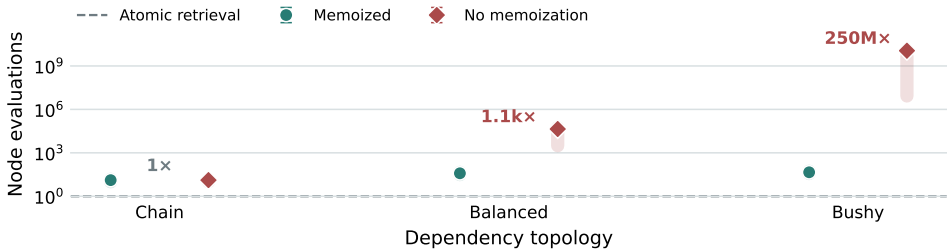


Figure 10: **Reusing computed results avoids repeated recovery work.** Reconstruction costs with and without reuse of shared ancestors, across 99 cones per graph type. Labels give mean cost ratios; the dashed line marks one-read retrieval. Pale bars show interquartile ranges, and error bars show bootstrap 95% confidence intervals.

Algorithm 2 Goal-conditioned versioned liveness

Require: Turns $\tau_{1:T}$ with reads R_t^v and writes W_t^v; reference files $F^\star$; entity costs $c(v)$
Ensure: L_t^{goal} and $\widehat{\text{EIR}}_{\text{goal}}(t)$

```
assign file versions in tool-call order (reads before writes)
add version-to-action read edges and action-to-version write edges
add each write's conservative dependencies since the preceding write
S ← {t : τ_t reads or writes a file in F*}
G ← version nodes backward-reachable from S
u(v) ← last action in the slice that consumes v
for t = 1 to T do
    L_t^goal ← {v_t(p) : v_t(p) ∈ G ∧ u(v_t(p)) ≥ t}
    EIR_goal(t) ← Σ_{v ∈ L_t^goal} c(v)
end for
return the per-cut demand sequence and its peak
```

The retrospective demand proxy is

$$\widehat{\text{EIR}}_{\text{goal}}(t) = \sum_{v \in L_t^{\text{goal}}} c(v), \qquad \widehat{\text{EIR}}_{\text{goal}}^{\max} = \max_t \widehat{\text{EIR}}_{\text{goal}}(t),$$

where $c(v)$ is an estimated token cost for file state. A run with no observed reference-file access has no graph sink and receives zero observed goal-conditioned demand; this is a coverage failure, not evidence that the task intrinsically requires no information.

Validation and interpretation. On 36 LACUNA cases augmented with a wrong-path future read, the graph retains all known required files and excludes the distractor in every case. This tests graph mechanics under known dependencies. On real traces, missing internal reasoning edges and alternative correct solutions remain unobserved, so the proxy should be interpreted as one conservative observed slice rather than a minimal requirement.

C.2 Software-agent trajectories: Verified, CoderForge, and OpenHands

We apply the retrospective demand proxy to 4,988 real agent trajectories from SWE-bench Verified (Jimenez et al., 2024) produced by nine frontier models (`gemini-3-pro-preview`, `claude-opus-4.6`, `claude-opus-4.5`, `gemini-3-flash-preview`, `minimax-m2.5`, `glm-5`, `gpt-5.2`, `claude-haiku-4.5`, `gpt-5-mini`; resolve rates 56–100%). Unlike LACUNA, these trajectories expose no declared task graph and no closed-form EIR; we apply the backward-liveness analysis of §C to produce the operational proxy $\widehat{\text{EIR}}_{\text{live}}(t) = \text{cost}(L_t^{\text{norm}})$ where L_t^{norm} is the reference-solution set—the files edited by the gold patch. It anchors one certified solution and is not assumed necessary for every possible correct continuation.

Table 15: Domain anchors for retrospective demand. SWE uses the hybrid graph; benchmarks without a reference solution retain separate realized-state adapters and are not pooled with SWE token demand.

Benchmark	State entity	Goal or anchor
SWE-bench / SWE-rebench	versioned file	independent reference-patch file
τ-Bench	typed database entity	reference task state
Terminal-Bench 2.0	versioned shell path	realized trace only

Model contrast and pressure regime. Across nine models, mean context occupancy ranges from 1% (`gpt-5-mini`, short trajectories, 56% resolve) to 16% (`gemini-3-flash-preview`, long exhaustive searches, 76% resolve). These naive occupancy estimates are low, but they do not establish $P < 1$: neither exact real-task EIR nor effective accessible capacity is certified, and supply can be limited when an observation is first returned even if the accumulated history fits the model window. The wide capability spread (56%–100% resolve) creates a richer failure landscape than a two-model study: models fail for qualitatively different reasons—some quit too early, some search too broadly— providing a stringent test of whether $\widehat{\mathrm{EIR}}_{\mathrm{live}}$ captures task difficulty independently of model behavior.

Observed scaffold-level supply limits. We audit raw exports for scaffold-specific clipping markers rather than infer availability from occupancy. In the publicly available OpenHands trajectory logs, a truncated tool response contains the literal marker `<response clipped>`. We treat this marker as evidence that the complete tool output is unavailable in the recorded interaction, and therefore as a proxy for tool-derived information supplied to the agent at that step. It is not a reconstruction of the agent's full context: the exports omit per-call prompt snapshots and do not reveal any hidden scaffold state or later conversation condensation. All 5,000 mini-SWE-agent configurations cap each command observation at 10,000 characters, retaining the first and last 5,000. This path fires 598 times in 265 trajectories; 135 events in 116 trajectories target a gold-patch file. All such events occur in `gemini-3-pro`, which resolves all 500 tasks, so this cohort cannot identify an effect on failure. In CoderForge, 244 file-view responses are explicitly clipped across 203/500 trajectories; 116 events in 115 trajectories target a gold-patch file. Per-call prompt size is monotone in all 500 runs, providing evidence against silent history condensation in this scaffold, though eight runs record context-window overflow errors. In OpenHands, 24,270 views are clipped across 20,839/67,074 trajectories; 12,452 events in 12,114 trajectories concern files the agent later modifies. Modified-file clipping is a behavioral anchor, not proof that omitted lines were normatively required. No OpenHands-derived export contains explicit conversation-summary markers, so absence of history condensation is not established. Resolve rates differ across clipping groups, but task difficulty, exploration breadth, and model behavior confound those observational contrasts. The audit establishes real output-level supply limits, not that clipping caused failures.

We further restrict the gold-patch cohorts to source views whose requested range demonstrably covers an old line deleted or replaced by the gold patch, before the first detected write to that file. We then search later covering views, also before that write, for the exact normalized old-line text. Pure additions are excluded because their new lines did not exist to be viewed. In Verified, 87 changed old-line instances across 30 runs are re-exposed after a clipped view, while 29 instances across nine runs are not re-exposed before the write or run end. All occur in `gemini-3-pro`, which resolves every task. CoderForge contains 91 re-exposed instances across 30 runs and 34 non-re-exposed instances across six runs. It has only one run per task, so no same-task outcome contrast is available. This refinement verifies omission of specific gold-changed old-line text from retained tool output; it still does not establish that the line was necessary for the agent's decision or that its omission caused failure.

SWE-rebench provides an independent gold patch for all 6,306 OpenHands tasks and all 67,074 runs. Applying the same mechanical test finds 15,404 re-exposed gold-changed old-line instances across 2,916 runs and 7,493 instances not re-exposed before the first detected write across 1,239 runs. Restricting to exclusive trajectory groups without ambiguous line evidence, raw task success is 48.5% among 2,405 recovered runs and 35.8% among 1,187 never-recovered runs, a -12.7 percentage-point difference (95% CI $[-16.1, -9.3]$). We retain all 3,592 runs in a logistic mixed

model with a random intercept for each of 676 tasks and fixed effects for total base-source lines across gold-fix files, number of gold-fix files, and gold-patch additions plus deletions. Ten tasks with unavailable base-file sizes use median imputation with a missingness indicator. Model is constant because all OpenHands runs use `Qwen3-Coder-480B`. The random-intercept association is null (OR 0.89, 95% credible interval $[0.67, 1.19]$). A correlated-random-effects sensitivity that additionally controls each task's proportion of never-recovered runs is also null (OR 0.94, 95% credible interval $[0.71, 1.26]$). To avoid relying only on random-effects assumptions, a separate model estimates task success from runs outside both clipping groups and again finds no association (OR 1.27, 95% CI $[0.82, 1.98]$, $p = .29$). Leave-one-model-out estimation is unavailable because the cohort contains one model. Recovery is itself post-clipping agent behavior, so even the adjusted estimates are descriptive. The raw gap does not persist after adjustment for task mix.

Figure 1A gives one trajectory-level example using the literal marker in the public log rather than a simulated context policy.

Demand and downstream behavior. Within the same 3,592 OpenHands clipping-classifier runs, we test whether hybrid peak demand is associated with failure. Dividing by the common nominal 128k window only shifts the log scale, so coefficients are per demand doubling. The association is small after controlling trajectory tokens (OR 1.022, 95% CI $[1.002, 1.041]$, $p = .027$) and null after adding task difficulty estimated from disjoint runs (OR 1.016, $[0.992, 1.041]$, $p = .191$). Hybrid demand uses the completed trajectory and is not a clean pre-run causal exposure.

For mediation, the primary pre-run exposure is total source lines in the gold-patch files, with median imputation and a missingness indicator for 56 runs. It predicts failure after trajectory-length adjustment (OR 1.30 per doubling, $[1.10, 1.55]$, $p = .003$), but not after independent task-difficulty adjustment (OR 1.03, $[0.82, 1.28]$, $p = .826$). Demand weakly predicts failure to recover an omitted gold line (OR 1.19, $[0.99, 1.42]$, $p = .059$), while non-recovery does not independently predict failure in the joint model (OR 1.41, $[0.92, 2.17]$, $p = .115$). Task-cluster bootstrap standardization gives an indirect risk difference of 0.19 percentage points from the 25th to 75th demand percentile (95% interval $[-0.05, 0.57]$; 200 resamples). An exploratory rework proxy combining repeated commands, edit-application failures, repeated edits, and submission after failed tests also shows no mediated path (rework-to-failure OR 1.00, $[0.79, 1.26]$, $p = .984$). Thus neither adjusted hybrid demand nor the available downstream-behavior measures support the proposed failure or mediation mechanism in this selected clipping cohort.

Figure 1B in the introduction summarizes the length-adjusted relationship; the task-difficulty-adjusted model reported here is the inferential result.

Known-dependency validation. Before applying the graph to real traces, we evaluate 36 synthetic LACUNA instances crossing aggregate, extremum, and arbitrary-access contracts; $m \in \{2, 4, 8, 16\}$; and value widths of 1–3 bits. Each case includes an injected wrong-path future read. The graph retains every known required file and excludes the distractor in 36/36 cases.

Estimating retrospective demand from completed runs. We apply the goal-conditioned graph to every eligible run: 4,988 Verified, 500 CoderForge, and 67,074 OpenHands/Qwen3 runs (72,562 total). Nodes represent actions and versioned files. Explicit reads feed an action, writes produce new versions, and a write depends conservatively on versions read since the preceding write. We backward-slice from every observed read or write of a reference-patch file and compute liveness only over versions in that slice. This binary contribution relation avoids pseudo-precise utility weights. Reachability does not establish causal necessity.

Table 16 reports the results. Mean peak demand is 9.0k tokens on Verified, 73.1k on CoderForge, and 4.5k on OpenHands; medians are 4.6k, 53.6k, and 1.2k. Gold-file sink coverage is 92.0%, 99.0%, and 57.4%, respectively. A zero value can therefore mean that the recorded run never accessed a reference-patch file, not that the task required no information.

Exploratory access behavior. We measure token distance from each read to the next read, overwrite, or trajectory end. The inferential model replaces separate bins with $x = \log_2(\text{distance}/500)$, using bin midpoints over the stable 0–32k range. A task-fixed-effects Poisson model uses reread count as the outcome and log token exposure as an offset; errors are clustered by run. It includes run

Table 16: **Goal-conditioned demand calculated from completed SWE runs.** The graph backward-slices from observed accesses to reference-patch files. Goal access is the fraction of runs with at least one such sink.

Cohort	Runs	Goal access	Mean peak demand	Failure AUC
Verified	4,988	92.0%	9,001	.541
CoderForge	500	99.0%	73,135	.440
OpenHands/Qwen3	67,074	57.4%	4,460	.527

length, model indicators, failure, gold-needed status, and all lower-order interactions. The sparse 32–64k needed-file bin contains one reread and is excluded. The model contains 31,779 cells from 4,987 runs and all 500 task fixed effects; one run has no usable file-version exposure.

The all-file failed-by-log-distance term is null (RR 1.018, 95% CI 0.992–1.046, p=0.174). The pre-specified failed×needed× $\log_2$(distance) term is negative (RR 0.951 per doubling, 0.911–0.992, p=0.020; Table 17). Thus, relative to resolved runs and to their behavior on other files, failed runs' reread rate for gold-needed files falls by an additional 4.9% per distance doubling. Needed files are generally reread more often, so this is a relative attenuation rather than an absence of retrieval.

The survival formulation is primary because a file that is never revisited still contributes waiting-time exposure until run end; it does not condition on eventual return. We nevertheless run three gap sensitivities. The original end-censored maximum gives OR 1.44, but can be mechanically inflated when failed runs continue after useful work ends and is not treated as confirmatory. A closed-gap analysis removes that tail but conditions on eventual reread or edit; after task/model effects and run-length control its needed-file OR is 0.86. In a joint closed-gap placebo model, needed-file OR is 0.87 and other-file OR is 0.81, showing that this selected subset is not solution-specific.

The fair-clock sensitivity avoids both extremes by retaining nonreturns but stopping every clock at the run's final edit of any file. This fixes the earlier overly strict implementation, which required a gold-needed file itself to be edited and retained only 270 runs and 12 discordant tasks. The corrected metric covers 4,574 runs and 261 discordant tasks. Raw medians are 2,326 tokens (resolved) and 3,738 (failed). Conditional logistic regression with task and model effects and $\log_2(1 + \text{run tokens})$ gives OR 1.18 per gap doubling; a 1,000-sample task-stratified run bootstrap gives [0.985, 1.43] (all fits successful). Omitting run length gives OR 1.10, so adjustment does not hide a large mediated effect here. The fair-clock direction agrees with the survival model but is not independently significant. Neither analysis distinguishes a model that retained the file, a model that lost track of it, and an agent that followed a different reasoning path. They characterize recorded access behavior, not context-caused failure.

Conservative information-use anomaly audit. We additionally search the 72,562 SWE traces for four exact log signatures: tool-reported edit-application failure, identical command and output with no intervening input-file write, exact edit repetition after a failed-test candidate, and submission after an explicit failing-test result. Rules exclude test-runner and environment failures and were manually reviewed on stratified samples. Only 25 edit-application failures remain (25/25 reviewed events are genuine tool failures), but they do not identify why the edit was stale. Exact command/output repetition is common (77,164 events; 75/75 reviewed are exact), yet the sample includes deliberate repro reruns and patch inspection, so it is not a misuse label. The pipeline finds 110 repeated-edit candidates, but the saved event representation lacks enough intervening test evidence to validate them strictly. Only three submit-after-failed-test events remain after excluding infrastructure failures; two are confirmed and one has truncated output. These log-only signatures therefore do not provide an adequately powered, high-specificity context-failure test.

C.3 Cross-domain adapters: transactional entities and shell files

τ-Bench entities. τ-Bench (Yao et al., 2024) contains persistent-state tool interactions but no patch files. We normalize 318 frontier-model trajectories (226 retail and 92 airline) and treat typed database identifiers—users, orders, products, and items—as state entities. Median peak later-used demand is four entities for retail and two for airline; the 99th percentiles are eight and three. An

Table 17: **Recorded return rates differ by outcome and file relevance.** The primary rate ratio comes from a task-fixed-effects Poisson model with log token exposure as offset, run-clustered errors, run-length and model controls. The fair-clock sensitivity ends at each run's final edit of any file; its interval is a task-stratified run bootstrap.

Test	Estimate	95% CI	p
Failed $\times \log_2$ distance	RR 1.02	[0.99, 1.05]	0.174
Failed $\times$ needed $\times \log_2$ distance	RR 0.951	[0.911, 0.992]	0.020
Fair-clock needed-file gap, per doubling	OR 1.18	[0.985, 1.43]	—

entity-counting task-side proxy has failure AUC 0.555 overall, but it is a task-complexity association rather than exact EIR or evidence of context-caused failure.

Terminal-Bench shell files. Terminal-Bench 2.0 (Samiei et al., 2026) provides neither a normative patch nor a declared dependency graph, but its trial directories retain nested command/observation logs. Parsing shell commands into versioned file reads and writes yields 166 analyzable trajectories from 178 trials; 12 have fewer than three tool turns. Median peak later-used state is 22.5 files (11.5k estimated tokens), the 90th percentile is 131 files (64.5k tokens), and the 99th percentile is 455 files (226.5k tokens). Approximate path extraction and the absence of a reference solution make this a realized shell-trace audit, not exact EIR or normative demand.

Together these adapters show that hindsight demand can be reconstructed when a trace exposes identifiable reads and writes. Their native units remain separate from software-file token costs.

C.4 Mechanistic evidence across RQ1–RQ3

Evidence from execution logs. We inspect recorded replies and tool actions at the first missing-input step and subsequent steps. In the Stepwise Sum example with $m = 16$ and $h = 4$, the first missing input occurs at step 7, which needs A_1 and A_5; only A_5 remains in context. These traces show how agents proceed when an input is missing and how the resulting mistakes spread.

Continuing without acknowledging the missing input. A representative `Gpt56Reasoning` reply is simply `ANSWER 7 3090`, although the correct answer is `6392`. The recorded reply contains no recovery action or acknowledgment that A_1 is missing. This establishes continued execution without visible recovery, rather than what the model internally recognized.

Recognizing the loss but continuing anyway. The Claude traces make the missing input explicit. A representative `ClaudeHaiku45` reply lists retained answers including $A_4 = 5258$ and $A_5 = 1243$, then submits `3090`. `ClaudeOpus46` states that A_1 is not resident before continuing. In these examples, acknowledging the missing input does not lead to a correct recovery; the agent proceeds using an invalid substitute.

Propagation through earlier answers. Once an incorrect result is retained, later steps may compute correctly from incorrect inputs. We identify this pattern by applying the declared operation to the agent's own retained answers: if it reproduces the submitted answer using an incorrect earlier result, the step is classified as a cascade error. This accounts for errors at steps with no new missing input and supports the propagation analysis in Section 4.1. The Stepwise Sum–Store–Recall comparison in Figure 9 is consistent with this mechanism, although the cross-workload difference is not itself a causal estimate.

Stopping before recovery completes. In the recovery probes, `Gpt41Mini` can follow the correct dependencies but stop before reaching the retained results needed to reconstruct the target. Figure 11 contrasts completed traversal with early termination in the original Stepwise Sum subset. This differs from substitution: the agent begins a valid recovery but does not finish it.

The traces distinguish three observable behaviors: continuing without acknowledging the loss, acknowledging it without recovering the input, and stopping during recovery. They explain how miss-

ing information can lead to persistent errors or incomplete execution, complementing the controlled restoration tests.

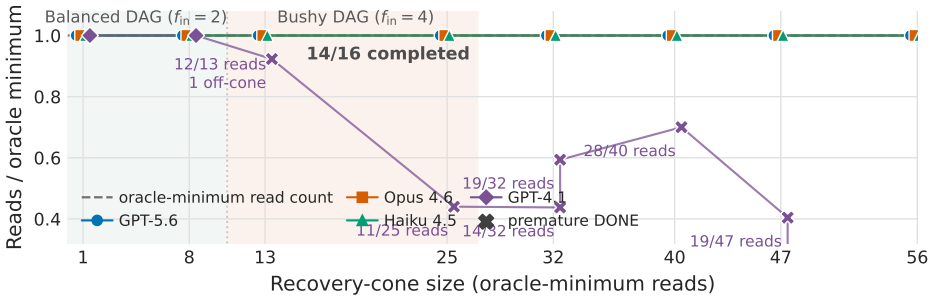


Figure 11: **When recovery succeeds it is read-optimal; the observed failures stop too soon.** Read overhead is reads divided by the oracle minimum (1.0 is optimal). All 14 completed probes equal 1.0; crosses mark the two GPT-4.1 probes that issue DONE before reading the full recovery cone. All 16 probes in the original Stepwise Sum subset are shown.

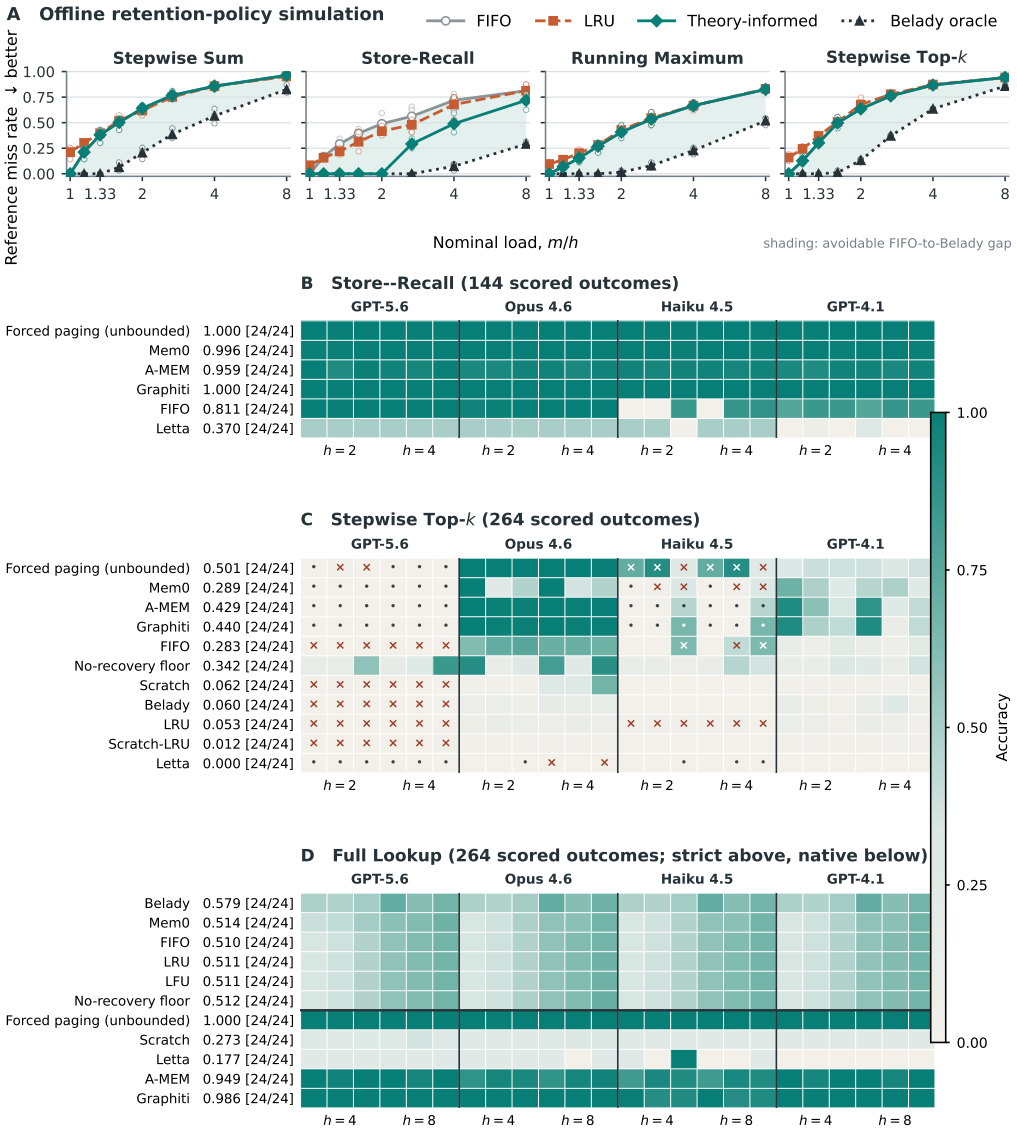


Figure 12: **Retention policy and system execution both affect performance. A:** Simulated policy miss rates; Belady uses future requests. **B–C:** Architecture comparisons on Store–Recall and Stepwise Top-k. Crosses mark turn-limit failures. **D:** Full Lookup comparisons, separating equal-capacity controls from native systems. Row labels report means and scored counts.