LENS Benchmark: Preliminary Results

Why Simple Retrieval Beats Complex Memory Architectures
at Longitudinal Evidence Synthesis

Mark Lubin `mark@synix.dev`

February 25, 2026

Abstract

LENS (Longitudinal Evidence-backed Narrative Signals) tests whether LLM agent memory systems can synthesize conclusions from evidence scattered across many sequential episodes. We evaluate 7 memory architectures (plus a no-memory control) across 6 domains, 2 context budgets, and 120 episodes per run (90 scored runs total). The central finding: **only simple BM25+embedding retrieval statistically outperforms a naive LLM that reads all evidence in context.** Knowledge graphs, agent memory, vector stores, and progressive summarization all either tie or lose to the naive baseline. This report presents the benchmark task in full detail—including verbatim episode text, the agent's tool interface, and exact scoring formulas—followed by per-system architecture descriptions, overall rankings, scaling analysis, and a canonical scenario trace showing how each system handles a specific question.

1. Introduction

LLM agents need memory to handle tasks that span more data than fits in a single context window. The standard approach is to pair an LLM with an external memory system—a vector store, knowledge graph, or agent-managed database—and let the agent search for what it needs at inference time. But which memory architecture actually works?

LENS tests this by requiring synthesis across many episodes. An agent ingests 120 sequential monitoring reports (30 signal, 90 distractors), then answers questions whose answers are only discoverable by connecting facts from multiple reports. No single episode answers any question; if it did, the benchmark would reduce to single-document retrieval.

We evaluate 7 memory systems across 6 professional domains (infrastructure failure, insider threats, environmental monitoring, clinical trials, IT scaling, and financial forensics). Every component is fully specified: the exact text the agent ingests, the tool interface it uses, the scoring formulas applied to its answers, and the composite weighting that produces final rankings. All code, datasets, and run artifacts are available for reproduction.

2. Benchmark Task

2.1. Episodes

Each domain scope consists of **30 signal episodes** encoding a domain-specific storyline plus **90 format-matched distractors** from unrelated monitoring domains. Signal episodes follow a five-phase arc: baseline, early signal, red herring, escalation, and root cause. Distractors share the same structured-data format (section headers, station readings, field notes) but cover orthogonal topics (e.g., air quality monitoring data interleaved with water quality reports).

The following is a verbatim excerpt from the environmental monitoring domain (Scope 04). Episode 1 establishes the baseline:

Listing 1: Episode 1 (baseline) — Scope 04: Environmental Drift

```
## 2024-06-01 Daily Operations Summary

### Station Readings
#### WQ-01
- pH: 7.2
- DO (mg/L): 7.5
- Turbidity (NTU): 3.0
- Cr (ug/L): 3
- Pb (ug/L): 1.2

#### WQ-02
- Cr (ug/L): 3

#### WQ-03
- Cr (ug/L): 3
[... 6 stations, 10 parameters each ...]

### Field Notes
- Standard summer conditions at all stations
- Stream clarity good, visible substrate at WQ-01

### Notes
- Day 1 monitoring. All parameters within seasonal
  baseline ranges. Routine operations.
```

Twenty-four days later, episode 25 contains the critical finding:

Listing 2: Episode 25 (root cause) — key evidence for Q3

```
## 2024-06-25 Daily Operations Summary

### Station Readings
#### WQ-01
- Cr (ug/L): 3

#### WQ-02
- Cr (ug/L): 5

#### WQ-03
- Cr (ug/L): 132

#### WQ-04
- Cr (ug/L): 84.2

#### WQ-05
- Cr (ug/L): 55.9
```

```
#### WQ-06
- Cr (ug/L): 35.9

### Compliance Flags
- VIOLATION: Cr at WQ-03 (132 ug/L) exceeds EPA MCL
  (100 ug/L)
- WARNING: Cr at WQ-04 (84.2 ug/L) approaching MCL
  threshold
- Field inspection: unpermitted discharge pipe
  identified between WQ-02 and WQ-03 at RM 18.6
```

The critical detail—"unpermitted discharge pipe identified between WQ-02 and WQ-03 at RM 18.6"—is a single sentence in the compliance flags section. It is surrounded by 119 other episodes of structured monitoring data. To answer the question "What is the source of the chromium contamination?", an agent must find this sentence, connect it to the spatial concentration gradient across stations, and dismiss the agricultural runoff red herring.

2.2. Signal Structure

The 30 signal episodes follow a prescribed arc:

Table 1: Episode arc structure per scope.

Phase	Episodes	Signal Density	Purpose
Baseline	1–5	None	Establish normal parameters
Early signal	6–15	Low	Subtle deviations begin
Red herring	10–18	Low–Medium	Plausible alternative explanation
Escalation	16–25	Medium–High	Signal becomes undeniable
Root cause	25–30	High	Key evidence surfaces

The 90 distractors are generated from orthogonal domains (air quality, soil geochemistry, traffic monitoring, etc.) using the same structured-data format. They share section headers and measurement syntax but contain no terms related to the signal storyline.

2.3. Checkpoints and Questions

Questions are asked at **7 checkpoints** positioned after episodes [18, 38, 46, 58, 79, 99, 119], meaning the agent has ingested that many episodes (signal + distractors interleaved) before answering. Three question types test different synthesis capabilities:

- **Longitudinal**: Requires connecting observations across multiple episodes. *Example*: "What is the source of the chromium contamination and how can you determine its location?"
- **Null hypothesis**: Tests whether the system can retrieve a specific recorded value. *Example*: "What was the dissolved oxygen reading at WQ-01 on June 8th?"
- **Action recommendation**: Asks what to do based on the evidence. *Example*: "What regulatory and remediation actions should the water authority take?"

Each question has a **canonical answer** and a list of **key facts**—atomic claims that the answer must demonstrate. For example, Q3 (above) requires 4 key facts: (1) progressive downstream dilution, (2) discharge source between the second and third gauges, (3) upstream stations at baseline, and (4) turbidity explained by nutrients, not chromium.

2.4. Agent Interface

Every memory system exposes the same tool interface to the agent. The agent receives a system prompt (Listing 3) and three mandatory tools:

Listing 3: Agent system prompt (verbatim)

```
You are a research assistant with access to a memory
system. Your task is to answer the user's question by
searching and retrieving information from memory.

Instructions:
- Use memory_search to find relevant information for
  the question.
- Use memory_retrieve to get full document details
  when you need specifics.
- Use memory_capabilities to understand what the
  memory system offers, including available search
  modes, filter fields, and any extra tools.
- If extra tools are available (e.g. batch_retrieve),
  prefer them for efficiency.
- Synthesize your findings into a clear, concise
  answer.
- IMPORTANT: For each claim in your answer, cite the
  supporting episode using the format [ref_id]. Every
  factual statement must have at least one citation.
- If you cannot find sufficient information, say so
  clearly.
- You have a limited number of turns and tool calls.
  Use them efficiently.
```

Mandatory tools:

- `memory_search(query, limit)` → `[{ref_id, text, score}]` — search the memory system using a natural language query. Returns up to `limit` results ranked by the system's internal scoring.
- `memory_retrieve(ref_id)` → `{ref_id, text}` — retrieve the full text of a specific episode by its reference ID.
- `memory_capabilities()` → `{search_modes, extra_tools}` — discover what the memory system supports (e.g., whether `batch_retrieve` is available).

Extended tools (system-dependent): `batch_retrieve(ref_ids)` fetches multiple episodes in a single call. Systems that expose it (chunked-hybrid, letta, compaction) show a 3–12× reduction in tool call count for the same information.

2.5. Context Budgets

Context budgets constrain how much information the agent can read from memory per checkpoint. Two presets are used:

Table 2: Context budget presets. "Result tokens" = total `len(tool_result.encode("utf-8"))` ÷ 4 across all tool calls per question.

Preset	Result Token Cap	Max Turns	Max Tool Calls	Max Agent Tokens
8K	8,192	8	16	16,384
16K	16,384	10	20	32,768

Only tool result content counts toward the result token cap—not the agent's own reasoning tokens. When the cumulative result tokens exceed the cap, the next tool call returns: `"[Context budget exhausted -- synthesize answer from evidence already retrieved]"`. The question continues (the agent can still produce an answer from what it has), but a budget violation is recorded.

3. Memory Architectures

Seven memory systems plus a no-memory control are evaluated. Each subsection describes the complete data path from raw episode text to retrieved result.

3.1. sqlite-chunked-hybrid (BM25 + embedding)

Ingest. The full episode text is inserted into a SQLite FTS5 full-text index. Simultaneously, the text is split at `###` section boundaries into chunks (~150 words each), and each chunk is embedded using a 768-dimensional model (GTE-ModernBERT). Both the full text and chunk embeddings are stored.

Storage. Two SQLite tables: `episodes` (full text with FTS5 index) and `chunks` (chunk text + embedding vector, foreign-keyed to episode).

Search. Reciprocal rank fusion (RRF) of two signals:

$$\text{score}(d) = \frac{1}{60 + \text{rank}_{\text{BM25}}(d)} + \frac{1}{60 + \text{rank}_{\text{embed}}(d)} \tag{1}$$

BM25 ranks full episodes by term overlap with the query. Embedding search ranks chunks by cosine similarity to the query embedding. Results are deduplicated by episode (best chunk per episode) and merged via RRF.

Retrieve. Returns the stored full episode text verbatim. No transformation or summarization.

3.2. cognee (knowledge graph)

Ingest. Episodes are buffered during the ingest phase. At `prepare()`, all episodes are batch-submitted to cognee (using `cognee.add()` with batch size 20) followed by `cognee.cognify()`, which runs LLM-based entity extraction and relationship resolution.

Storage. Three backends: LanceDB (chunk vectors for semantic search), Kuzu (entity-relationship graph), and SQLite (relational metadata). The entity graph captures nodes (e.g., "WQ-03", "chromium") and typed edges (e.g., "measured_at", "exceeds_threshold").

Search. Vector similarity over LanceDB chunks. Returns chunk text with a flat relevance score (cognee does not expose ranked scores—all results arrive with score ≈ 0.5).

Retrieve. Full episode text is cached at ingest time and returned on retrieve calls. The knowledge graph is only used during search.

3.3. graphiti (temporal knowledge graph)

Ingest. Episodes are buffered, then at `prepare()`, each episode undergoes LLM-based entity extraction via the Graphiti library. This produces ∼600 LLM calls per 30-episode scope—entity nodes and relationship edges are inserted into a FalkorDB graph with temporal provenance metadata.

Storage. FalkorDB (a Redis-based graph database). Entity nodes carry properties (name, type, description). Relationship edges carry temporal metadata (when the relationship was first/last observed) and episode provenance.

Search. Edge-hybrid search: ranks entity-relationship pairs by relevance to the query combined with episode mention frequency. Results are mapped back to source episodes.

Retrieve. Full episode text cached at ingest time; returned verbatim.

Limitation. Entity extraction is slow—scopes 03–05 timed out during evaluation (each required >600 LLM calls for extraction alone). Only 6 of 12 possible scope/budget combinations completed.

3.4. letta (agent memory)

Ingest. Each episode is stored as an archival passage via the Letta server API (`passages.create`). One Letta agent is created per scope.

Storage. Letta's internal vector database. Passages are embedded and indexed automatically by the Letta server.

Search. Vector similarity over passages. The Letta server returns passage text but does not expose explicit relevance scores.

Retrieve. Full episode text cached at ingest; returned verbatim.

3.5. letta-sleepy (agent memory + sleep-time consolidation)

Identical to letta, but after the ingest/prepare phase, the Letta server runs a "sleep-time" consolidation pass. This is intended to re-organize and summarize archival memory while the agent is not actively answering questions.

In practice, letta-sleepy performed statistically indistinguishably from letta (mean composite 0.322 vs. 0.327), suggesting that sleep-time consolidation neither helps nor hurts at this task scale.

3.6. mem0-raw (vector store)

Ingest. Each episode is embedded as a single vector via the Mem0 client and stored in a Qdrant collection. The `infer=False` flag is set, meaning no LLM extraction or summarization is applied—raw episode text is embedded directly.

Storage. Qdrant point collection. Each point contains the full episode embedding and the raw text as payload.

Search. Cosine similarity between the query embedding and stored episode embeddings. Returns full episode text with a similarity score.

Retrieve. Lookup by Qdrant point ID; returns the stored episode text.

3.7. compaction (progressive summarization)

Ingest. All episodes are buffered in memory during ingest.

Storage. A single rolling summary document, regenerated from scratch at each checkpoint. Before answering, the LLM reads all buffered episodes and produces a comprehensive summary. Individual episode texts are also cached for retrieve-by-ID.

Search. Always returns the summary document (regardless of query), with score = 1.0. There is no selective retrieval—the agent gets the same summary for every search.

Retrieve. By ref_id returns the original episode text; by default returns the summary.

Failure mode. The summary compresses 120 episodes (∼200K tokens of raw text) into ∼2K tokens. Specific details—exact values, station IDs, river mile markers—are lost in favor of general trends. This is by design: compaction tests whether high-level summaries suffice for longitudinal synthesis.

3.8. null (no memory, control)

No episodes are stored. All `memory_search` calls return empty results. This establishes a floor: any system scoring below null is actively harmful.

Table 3: Architecture summary.

System	Storage Backend	Chunking	LLM at Ingest	Search Method
chunked-hybrid	SQLite FTS5 + embeddings	150-word sections	No	RRF (BM25 + cosine)
cognee	LanceDB + Kuzu graph	Internal (sentence)	Yes	Vector similarity
graphiti	FalkorDB graph	Entity-level	Yes	Edge relevance
letta	Letta vector DB	Full passage	No	Vector similarity
letta-sleepy	Letta vector DB	Full passage	Yes (sleep)	Vector similarity
mem0-raw	Qdrant	Full episode	No	Cosine similarity
compaction	In-memory summary	None (single doc)	Yes	Always returns summary
null	None	N/A	No	Always empty

4. Scoring

4.1. Three-Tier Metrics

Scoring is organized into three tiers of increasing complexity.

Table 4: Scoring metrics. Weight column shows the contribution to the composite score (v3.3).

Metric	Tier	Weight	What It Measures
evidence_grounding	1 (mechanical)	0.08	Fraction of cited episodes that are signal (not distractor)
fact_recall	1 (mechanical)	0.00	Substring match of key facts (disabled—no discrimination)
evidence_coverage	1 (mechanical)	0.08	Fraction of required evidence episodes actually cited
budget_compliance	1 (mechanical)	0.07	1.0 if no budget violations, 0.0 otherwise
citation_coverage	1 (mechanical)	0.10	Fraction of answer sentences with at least one citation
answer_quality	2 (LLM judge)	0.17	Pairwise win rate vs. canonical answer across key facts
insight_depth	2 (LLM judge)	0.15	Quality of cross-episode synthesis and inference
reasoning_quality	2 (LLM judge)	0.10	Logical coherence and evidence-based reasoning
naive_baseline_advantage	3 (differential)	0.17	Pairwise win rate vs. a naive LLM that reads all evidence
action_quality	3 (differential)	0.08	Appropriateness of recommended actions

Hard gate. If `evidence_grounding` < 0.5, the composite score is set to 0.0 regardless of other metrics. This prevents systems that cite only distractors from scoring well on subjective judge metrics.

4.2. Pairwise LLM Judge

Tier-2 and tier-3 metrics use a pairwise judge. For each key fact, the judge receives both answers (candidate and reference) in randomized positions and returns which better demonstrates the finding:

Listing 4: Judge prompt template (verbatim from `judge.py`)

```
You are comparing two analyst responses to determine
which better demonstrates knowledge of a specific
finding from a longitudinal dataset.

Question asked: {question}

Finding to evaluate: {fact}

Response A:
{text_a}

Response B:
{text_b}
```

```
Which response better demonstrates awareness of this
finding? The response may use different terminology.
Focus on whether the core insight is present, not
exact wording.

Respond with ONLY one word: A, B, or TIE
```

Position debiasing. For each fact, the candidate answer is assigned to position A with probability 0.5 (determined by a seeded PRNG). The judge's positional verdict (A/B/TIE) is mapped back to the semantic winner (candidate/reference/tie). This controls for the known position bias of LLM judges.

Scoring. Per-fact: candidate wins = 1.0, tie = 0.5, reference wins = 0.0. The metric value is the mean across all key facts for all questions in the run.

Judge reliability. Position-swap audit on 100 random judgments: 88% agreement ($\kappa = 0.658$), 49% position-A bias. Judge failures (API errors, malformed responses) default to TIE and are logged. Post-hoc audit found 23% of runs (21/90) had $\geq$1 judge failure, primarily affecting cognee (12/12 failures) and letta variants.

4.3. Naive Baseline Advantage

The `naive_baseline_advantage` metric (weight 0.17) measures whether a memory system adds value beyond simply stuffing all evidence into the LLM's context window. The *naive baseline* is generated by concatenating all signal episodes for the scope into a single prompt and asking the same LLM to answer each question directly—no memory system, no tool calls, just raw context. This represents the ceiling of what brute-force context stuffing can achieve.

The metric uses the same pairwise judge: for each key fact, the judge compares the memory system's answer (candidate) against the naive baseline's answer (reference). A score of 0.5 means the system ties with the naive baseline; above 0.5 means the system outperforms it. Only chunked-hybrid's 95% CI is entirely above 0.5, meaning it reliably adds value over context stuffing.

Naive baseline answers are cached per `(question_id, model)` to ensure consistency across scoring runs.

4.4. Composite Formula

$$\text{composite} = \begin{cases} 0.0 & \text{if } \texttt{evidence\_grounding} < 0.5 \\ \dfrac{\sum_{m \in M} w_m \cdot v_m}{\sum_{m \in M} w_m} & \text{otherwise} \end{cases} \tag{2}$$

where M is the set of metrics present in the run, w_m is the weight from Table 4, and v_m is the metric value. The denominator normalizes for missing metrics (e.g., if a run has no action_recommendation questions, that metric is absent and the remaining weights are renormalized).

5. Overall Rankings

Table 5: Composite score across 6 scopes, 16K budget (95% bootstrap CI). "*" = significantly different from null ($p < 0.05$, Wilcoxon signed-rank).

System	Mean	95% CI	N	vs. null
sqlite-chunked-hybrid	**0.454**	[0.406, 0.502]	6	*
cognee	0.421	[0.397, 0.446]	6	*
graphiti	0.393	[0.220, 0.491]	3	
mem0-raw	0.330	[0.265, 0.388]	6	*
letta	0.327	[0.258, 0.399]	6	*
letta-sleepy	0.322	[0.268, 0.381]	6	*
compaction	0.245	[0.137, 0.328]	6	
null	0.000	[0.000, 0.000]	6	

Only chunked-hybrid's naive baseline advantage CI is entirely above 0.5, meaning it is the only system that reliably beats a context-stuffed LLM. All others either tie (cognee, graphiti) or lose (mem0, letta, letta-sleepy, compaction).

Concordance. Kendall's $W = 0.755$ across all paired comparisons; 12 of 28 pairwise tests significant at $p < 0.05$.

Budget effect. 16K significantly outperforms 8K ($p = 0.016$, paired Wilcoxon across all adapters).

Caveat. Post-hoc audit found that the LLM judge silently failed on 21/90 runs (23%), primarily affecting cognee (12/12 judge failures) and letta variants (4/12 each). Failed judgments defaulted to TIE, inflating mechanical-only composites. Cognee's true ranking requires re-scoring with a reliable judge.

6. Cross-Domain Consistency

Table 6: Composite score: Adapter × Scope (16K budget). "—" = did not finish.

System	S01	S02	S03	S04	S05	S06	Mean
chunked-hybrid	0.424	0.482	0.381	0.535	0.520	0.386	**0.454**
cognee	0.438	0.402	0.374	0.471	0.418	0.423	0.421
graphiti	0.491	0.220	—	—	—	0.467	0.393
mem0-raw	0.345	0.320	0.186	0.419	0.407	0.299	0.330
letta	0.288	0.338	0.215	0.455	0.424	0.239	0.327
letta-sleepy	0.300	0.313	0.252	0.451	0.377	0.240	0.322
compaction	0.339	0.000	0.198	0.364	0.309	0.259	0.245
null	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Chunked-hybrid ranks first or second in 5 of 6 scopes. Graphiti DNF'd on scopes 03–05 due to entity extraction timeouts ($\sim$600 LLM calls per scope). Compaction catastrophically fails on scope 02 (composite = 0.000): its rolling summary discarded the specific details that all questions required.

7. Scaling Behavior

How do systems behave as the corpus grows from episode 1 to 120? We extract per-checkpoint metrics averaged across all scored scopes.

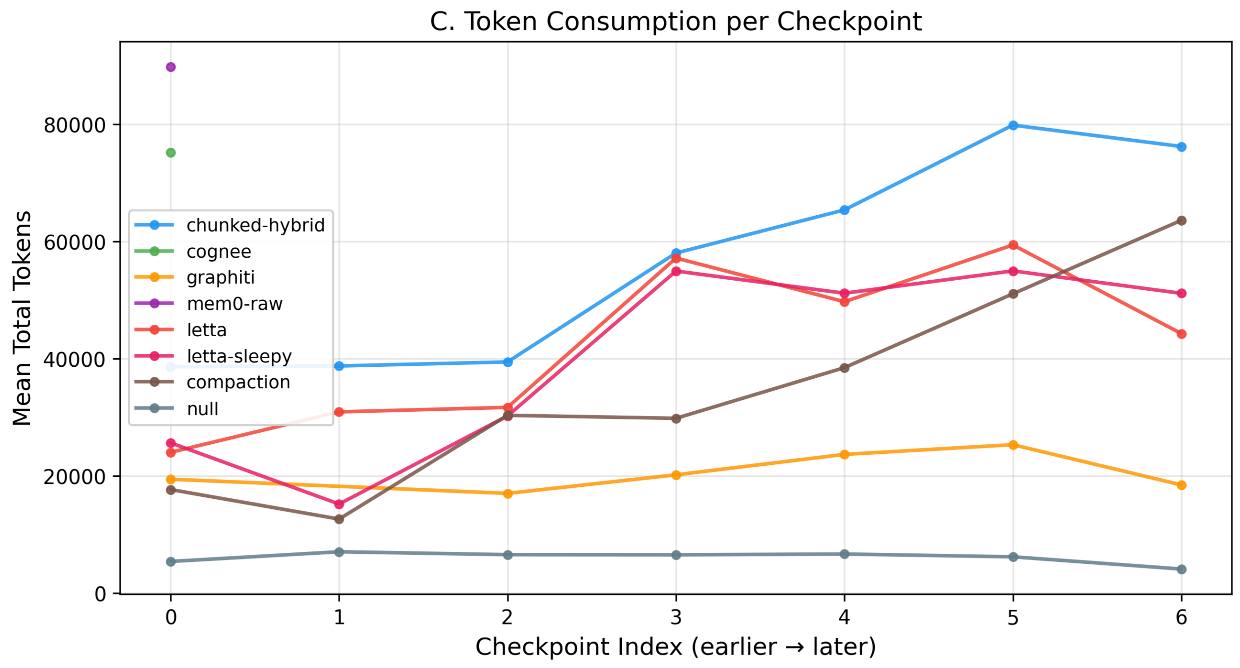


Figure 1: Token consumption per question grows with corpus size. Chunked-hybrid rises linearly (bounded by retrieval limit). Letta/letta-sleepy/compaction climb sharply as search reformulation burns tokens. Null stays flat (no memory to search).

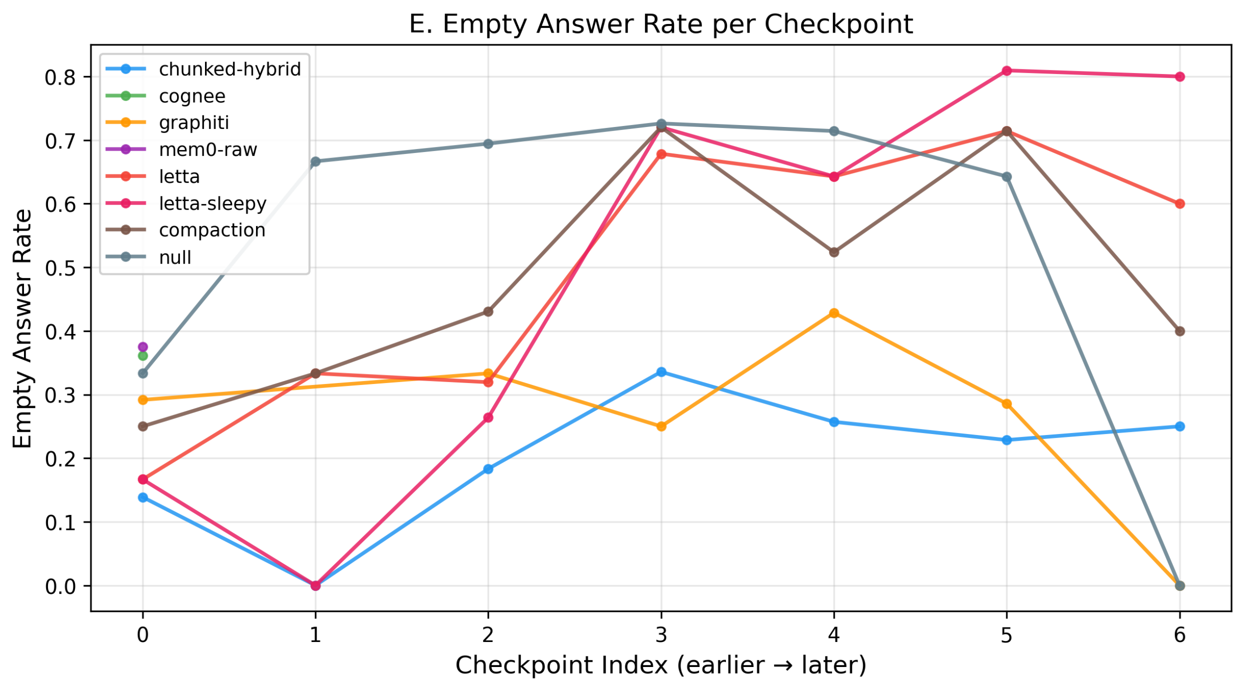
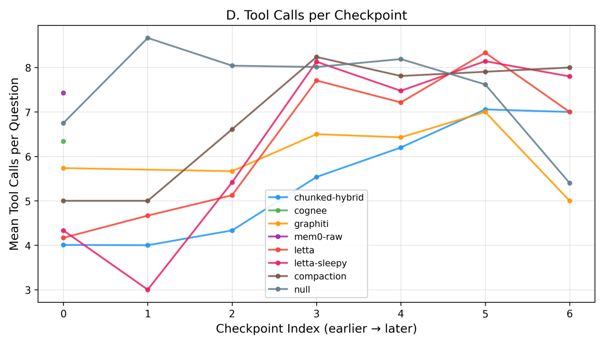
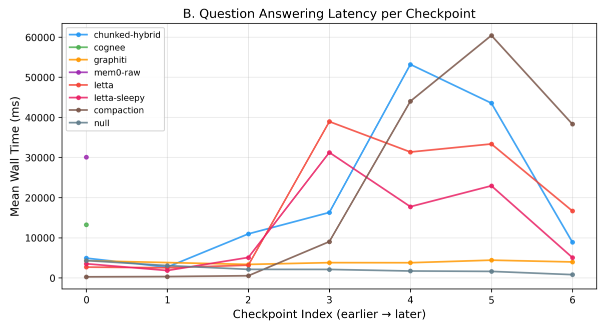


Figure 2: Empty answer rate by checkpoint. Chunked-hybrid maintains the lowest rate (14–33%). Letta and compaction degrade to 60–80% as the corpus grows and retrieval becomes unreliable.



(a) Tool calls per question



(b) Wall-clock time per question

Figure 3: Operational scaling. Chunked-hybrid starts at ∼4 calls and rises slowly. Other systems converge toward the 10-call turn limit. Compaction's wall time spikes at later checkpoints due to growing summary regeneration.

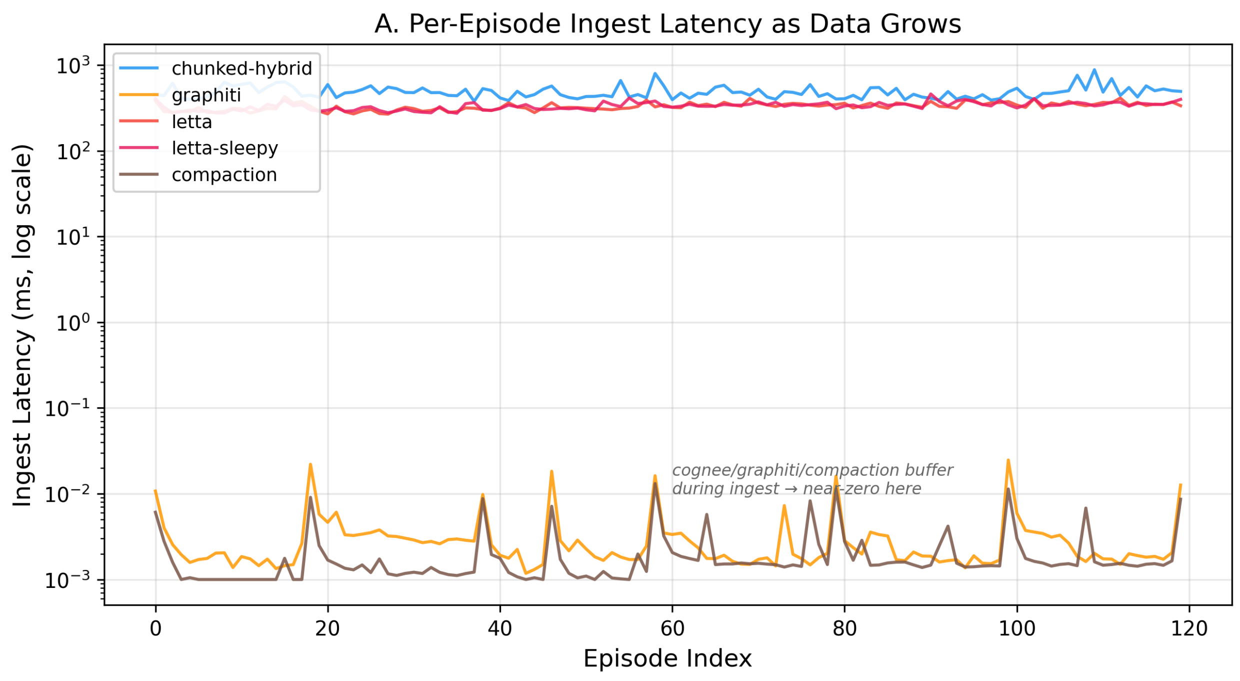


Figure 4: Per-episode ingest latency (log scale). Letta and chunked-hybrid process episodes during ingest (∼300–700ms). Graphiti and cognee buffer during ingest and do work in `prepare()` — their near-zero ingest times are misleading without that context.

Key observation. Token consumption (Figure 1) is the real scaling signal. When retrieval fails, agents don't stop — they reformulate and retry. Each retry consumes tokens for the query, the empty response, and the agent's internal reasoning. This *search thrashing* explains why complex architectures burn 4–8× more tokens than chunked-hybrid for the same questions.

8. Canonical Scenario Trace

To make the failure modes concrete, we trace one question through all systems.

8.1. The Question

Scope 04 (environmental drift): 6 water quality stations along a river. 30 signal episodes (daily monitoring reports) interleaved with 90 distractors (air quality, soil, traffic data). At checkpoint 99 (after 99 of 120 episodes):

Q3 (longitudinal): *"What is the source of the chromium contamination and how can you determine its location?"*

Ground truth: An unpermitted discharge pipe between stations WQ-02 and WQ-03 at river-mile 18.6. The spatial gradient proves this: WQ-01 and WQ-02 at baseline (3−5 µg/L), WQ-03 at peak (132 µg/L), downstream stations showing progressive dilution. Agricultural runoff was a red herring—it explains turbidity but not the chromium gradient.

Key evidence: Episode 025 (Listing 2)—specifically the compliance flag noting "unpermitted discharge pipe identified between WQ-02 and WQ-03 at RM 18.6".

8.2. Results Summary

Table 7: Canonical scenario: Chromium source question (ed04_q03, checkpoint 99, 16K budget).

System	Calls	Tokens	Time (s)	Answer	Pipe?	RM 18.6?
chunked-hybrid	**3**	**22,308**	**2.7**	**1,642 ch**	✓	✓
compaction	4	35,774	62.3	2,607 ch	×	×
null	6	4,537	1.4	212 ch	×	×
letta	8	74,722	126.5	2,567 ch	×	×
letta-sleepy	9	88,547	69.3	*empty*	×	×
cognee	9	157,187	19.7	*empty*	×	×
mem0-raw	9	180,102	7.2	2,293 ch	×	×

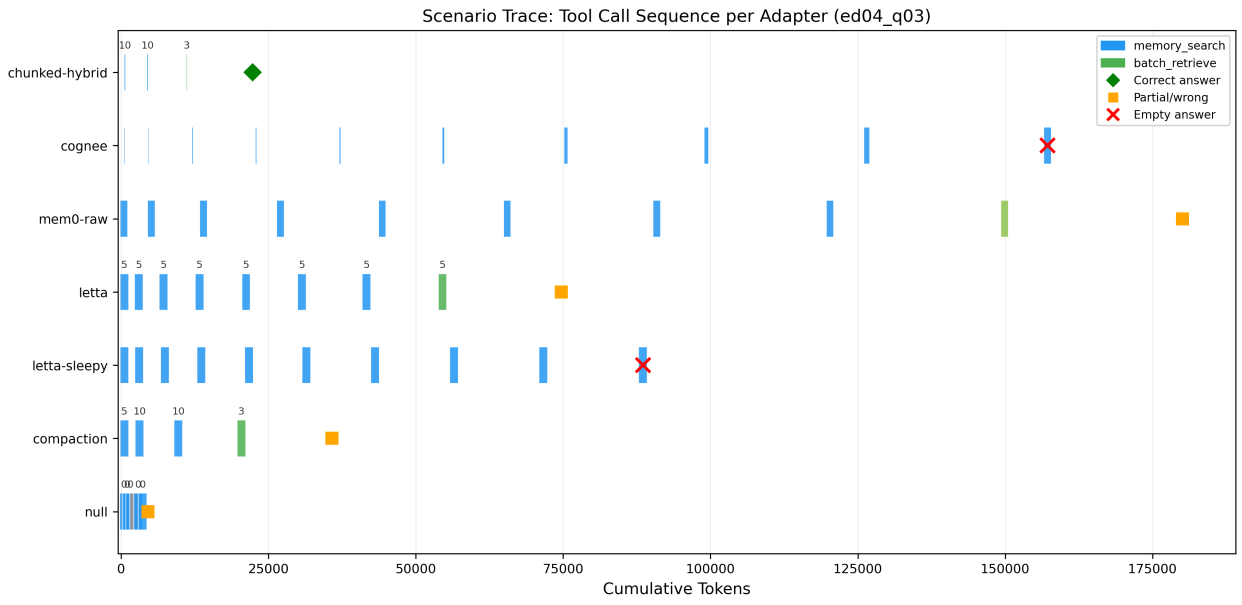


Figure 5: Tool-call sequence per adapter for the chromium source question. Each blue block is a `memory_search`, each green block is a `batch_retrieve`. Green diamond = correct answer, orange square = partial/wrong, red X = empty answer. X-axis shows cumulative tokens consumed.

8.3. Chunked-hybrid (correct answer, 3 calls)

The full tool-call sequence:

Call	Action	Cum. Tok.	Hits
1	`memory_search("chromium contamination source determine location")`	656	10
2	`memory_search("source of chromium ... chromium leak")`	4,506	10
3	`batch_retrieve([ep_025, ep_018, ep_023])`	11,175	3

The first search finds episode 025 at rank 5 via BM25 keyword overlap (“source” + “chromium” + “discharge”). The second search, with expanded terms, confirms episode 025 and identifies supporting episodes. The agent then batch-retrieves 3 episodes and reads the critical sentence: “unpermitted discharge pipe identified between WQ-02 and WQ-03 at RM 18.6”. The answer cites 3 episodes and names both the pipe and its exact location.

Why it works. BM25 matches the query term “source” against the episode text containing “discharge pipe”. The RRF fusion boosts episode 025 because it scores well on both BM25 (keyword match on “discharge” and “chromium”) and embedding similarity (semantic match on “contamination source”). Three calls, 11K tokens, correct answer.

8.4. Letta-sleepy (empty answer, 9 calls)

Table 8: Letta-sleepy search queries and cumulative token consumption.

Call	Search Query	Cum. Tok.
1	`"chromium contamination source determine location"`	587
2	`"source of chromium contamination location determination"`	3,078
3	`"chromium source contamination analysis"`	7,452
4	`"chromium source contamination source location analysis report"`	13,632
5	`"chromium source contamination location determination site"`	21,701
6	`"chromium source contamination source identification"`	31,433
7	`"chromium source contamination upstream"`	43,084
8	`"... analysis report site investigation"`	56,513
9	`"... source identification site"`	71,631
10	`"... source upstream industrial"`	88,547

Ten searches, zero batch retrieves. Every search returns results from Letta’s vector store, but none surface episode 025 with sufficient prominence for the agent to retrieve it. The agent reformulates progressively longer queries—adding “report”, “site”, “investigation”, “upstream”, “industrial”—each consuming ∼8,000 tokens in returned results. After 88K tokens (5.4× the 16K budget) and 10 turns, it hits the turn limit and returns an empty answer.

Why it fails. Vector similarity search embeds the query and finds the nearest episodes in embedding space. But 119 other episodes also discuss water quality, contamination, and station readings. The critical sentence about the discharge pipe at RM 18.6 is embedded as part of a full episode that is semantically similar to dozens of others. There is no keyword signal to break the tie.

8.5. Compaction (partial answer, 4 calls)

Compaction retrieves episodes 001–003 (early baseline data). Its rolling summary compressed the 120 episodes into ∼2K tokens; the specific mention of “unpermitted discharge pipe” and “RM 18.6” was absorbed into a generic statement about spatial concentration gradients.

The agent’s answer correctly identifies a spatial gradient and correctly locates the source “between WQ-01 and WQ-02” (actually between WQ-02 and WQ-03), but it cannot name the discharge pipe or river mile—those details were lost in summarization. It scores 1/4 on correctness checks: spatial gradient yes, discharge pipe no, RM 18.6 no, WQ-03 peak no.

Why it fails. Compaction's summary preserves trends but discards specifics. The single sentence identifying the pipe—the most important sentence in 120 episodes—is indistinguishable from routine field notes during summarization.

8.6. Mem0-raw (wrong answer, 9 calls)

Mem0-raw's vector search returns episodes 018 and 026 (escalation data showing high chromium at several stations) but never surfaces episode 025. The agent identifies that chromium is elevated but mislocates the source between WQ-01 and WQ-02 (upstream of the actual source). It scores 1/4: spatial gradient yes, everything else wrong.

Why it fails. Full-episode embeddings average over all content in the episode—station readings, weather data, biological indicators, field notes. The critical sentence is a small fraction of episode 025's embedding. Semantically, multiple escalation episodes look nearly identical; the specific detail that distinguishes episode 025 is diluted.

9. Why Complex Systems Fail

Three patterns explain the rankings.

9.1. The Lossy Transformation Trap

Every architecture except chunked-hybrid introduces a lossy transformation between ingestion and retrieval:

Table 9: Lossy transformations introduced by each architecture.

System	Transformation	What's Lost
chunked-hybrid	None (raw text indexed)	Nothing
cognee	Entity extraction $\rightarrow$ graph	Temporal ordering, narrative details
mem0-raw	Vector embedding	Keyword-level specificity
letta	Agent-managed memory	Search reliability (stochastic retrieval)
letta-sleepy	Agent memory + consolidation	Same as letta + consolidation noise
compaction	Rolling LLM summary	Numeric precision, per-episode detail

Longitudinal synthesis requires *all* the details: exact values, temporal ordering, narrative context. When any of these are discarded by an intermediate LLM—whether summarizing, extracting entities, or re-encoding into embeddings—the answering agent works from a degraded input.

9.2. Search Thrashing

When retrieval fails, the agent reformulates and retries. Each retry costs tokens: the query, the returned results (even if irrelevant), and the agent's internal reasoning about what to try next. Systems with unreliable retrieval (letta, letta-sleepy, cognee, mem0-raw) burn 4–8$\times$ more tokens than chunked-hybrid on the same questions (Figure 1). This directly degrades performance: the token budget is exhausted on failed searches rather than answer synthesis.

The letta-sleepy trace (Table 8) is the canonical example: 10 increasingly desperate queries, 88K tokens consumed, zero useful information retrieved, empty answer.

9.3. The Keyword Advantage

Episode 025 contains the exact string "unpermitted discharge pipe" and the question asks about "the source of chromium contamination." BM25 matches these through shared and related terms. Embedding similarity requires the model to have learned that "discharge pipe" is semantically close to "contamination source"—a reasonable inference, but one that competes with 119 other episodes also discussing contamination in similar terms.

For sparse, specific facts buried in structured data, exact keyword match beats semantic similarity. This is not a general statement about retrieval—it is specific to the LENS task structure, where the signal is a few sentences among thousands of similar-looking records.

10. Summary

1. **Simple retrieval wins.** BM25 + embedding hybrid is the only system that statistically outperforms a naive context-stuffed baseline ($p < 0.001$).
2. **No system exceeds 0.50 composite.** The benchmark is hard — even the winner answers fewer than half the questions correctly.
3. **Complexity hurts.** Knowledge graphs, agent memory, and progressive summarization all introduce lossy transformations that destroy the details longitudinal synthesis requires.
4. **Token consumption is the scaling signal.** Systems that can't find evidence in 1–2 queries burn their budgets on search thrashing, leaving nothing for answer synthesis.
5. **16K significantly outperforms 8K** ($p = 0.016$), confirming that budget constraints are binding.

These results suggest that current memory architectures are optimized for the wrong problem. They trade fidelity for abstraction, assuming the agent needs pre-digested knowledge. For longitudinal synthesis—where the signal is sparse, specific, and distributed—the agent needs raw evidence and reliable retrieval. Everything else is noise.