

Expensive writes, noisy reads: provider-call telemetry from a production agent-memory system

Luiz Antonio Busnello

Version 1.0 · doi:10.5281/zenodo.23354363

Status (2026-10-13). Version 1.0 of a data note, not peer reviewed. It describes the dataset Busnello, L. A. (2026), *Phase-attributed provider-call telemetry from a production agent-memory system (32-day window, 2026-08-07 to 2026-09-08)*, version 1.0.0, Zenodo, 10.5281/zenodo.23130191. Rule of this file: each number is either recomputed from that dataset by `analysis.py` or quoted from the dataset README, and `parity.py` binds every number in the text to the metric or README statement it reports and fails on any other.

Abstract

Characterization studies of agent memory find that building memory costs more than querying it, but they measure benchmark runs, not deployments. We describe a public dataset of 3,544 external model-provider calls recorded over 32 days by the shared store of nox-mem, a self-hosted memory service consulted by six agents. Each call is labelled with a phase: construction, query or maintenance. We read the same table twice. The first reading is about cost. Construction is 3.16% of calls and 55.7% of estimated cost, and one construction call costs about 55 times as much as one query call, or about 25 times once the rows that match a monitoring signature are removed. Construction's share of cost stays between 55.7% and 58.7% whether or not the rows that match a monitoring signature are removed. The second is about the instrument. 62.8% of query calls match a monitoring or probe signature, so the call counts cannot be read as agent demand. The log also has a 13-day stretch with no recorded construction or maintenance call, and it contains no production knowledge-graph extraction. The whole window cost an estimated USD 0.135, which is USD 0.0042 per day, so the note reports shares and ratios rather than absolute cost. Both readings are reported next to the dataset's 16 documented defects, and the code that recomputes the numbers is deposited with the note.

Specifications

Subject	Computer science: information retrieval; systems for LLM agents
Type of data	Table (CSV), derived tables, verification script
Data collection	An in-process recorder wrote one row per provider call from every code path that reaches a model provider, with the gaps described in D7 and D9. Rows were exported from a hash-verified copy of the store.
Data format	Raw (13 columns) and derived (per-phase and per-day summaries, signature annotations)
Window	[2026-08-07T00:00Z, 2026-09-08T00:00Z), UTC, 32 days
Data location	Self-hosted virtual server; one host migration on 2026-08-23
Accessibility	Zenodo, https://doi.org/10.5281/zenodo.23130191 , CC BY 4.0
Related work	Omri et al., arXiv:2606.06448 (phase decomposition); nox-mem technical report [2]

Value of the data

- It is a record of phase-attributed provider calls from a deployed agent-memory system, rather than from a benchmark harness. Characterization studies of these systems measure controlled workloads [1];

this table shows how one deployment spends its calls over a month, idle stretches and monitoring traffic included.

- Phase labels adapt the construction / retrieval split of [1], which allows a coarse comparison, and the table adds a maintenance phase (compaction and consolidation) that the benchmark decomposition does not have.
- Token counts are stored next to the cost estimate, so a reader with other prices can recompute cost without collecting anything again. For embedding calls the counts are themselves estimates (D4).
- The dataset documents its own contamination. Signature annotations for monitoring and probe traffic are shipped in a separate file, so the raw table stays untouched, and readers can rerun every result with or without them.

1. Data description

The deposit holds the raw table (`provider_telemetry_2026-08-07_2026-09-08.csv`, 3,544 rows, 13 columns), one row of signature flags per call (`row_annotations.csv`), per-phase and per-day summaries, a verification script, and SHA-256 checksums. The dataset README documents the schema and the 16 known defects (D1 to D16). The verification script recomputes 108 values stated in the README and exits non-zero on any mismatch.

Each row is one call to an external model provider. It records when the call completed, the model, the calling code path (`caller`, 8 values), the phase, the input and output tokens, an estimated cost, the latency and a success flag. There is no free text: no prompts, responses, query text or document content. Two models occur: an embedding model and a small LLM, both from one provider. Token counts are estimated as `ceil(characters / 4)` for 96.6% of rows (all embedding calls), because the embedding endpoint did not reliably report usage (D4). Cost is tokens times public list prices captured at deployment, not an invoice (D5).

2. Methods

System. nox-mem stores agent memory in SQLite with full-text and vector indexes and a typed knowledge graph, and answers queries with hybrid lexical and dense retrieval [2]. During the window, six agents consulted its shared store. Each agent also has its own store, and those stores had no telemetry table when the operator checked on 2026-09-07; this was not re-measured for the deposit (D13).

Instrument. One recording module is called from every code path that reaches a provider. It writes one row per call and swallows its own errors, so telemetry cannot break ingestion or search; the price is that a dropped row leaves no trace (D9). The table had no rows before the recorder was deployed on 2026-08-07.

Phase assignment. Phase is derived from the call site. `embed.embedQuery` and `search-expansion.expand` are `query`. Document embedding, session distillation, digest building and knowledge-graph extraction are `construction`. Compaction and consolidation are `maintenance`. This adapts [1], where construction and retrieval are two of three phases. The third, generation, is not observable here, because the agents' own LLM turns do not pass through the memory service (D12). Phase is a property of the code path, not of the workload that triggered it: a monitor that issues a search produces a `query` row (D3).

Export. The rows were exported from a byte-preserved copy of the store, checked against its own SHA-256 manifest, without reading the live database. A second preserved copy, taken a day earlier, agrees byte for byte on every row it holds (3,278 rows).

Annotations. Three rules flag rows by signature: the post-deploy smoke test (4 rows), a monitoring canary (1,515 rows), and a five-hour night window on 2026-08-10 (588 rows). In that window, 474 rows arrive about every 30 s, which matches a latency probe scheduled for that night whose execution is not confirmed, and 114 form a burst at 02:00–02:02 UTC. A flag records a match on signature and timing, not a verified source. Below, *raw* means all 3,544 rows and *unflagged* means the 1,437 rows that no rule flags; the unflagged set is a filter on the deposited table, not a separate file.

3. First reading: what each phase costs

Shares. Table 1 gives the per-phase totals. Construction is a small minority of calls and the largest share of cost. Removing the flagged rows changes call counts a lot and cost shares little, because almost every flagged row is a cheap query embedding: the flagged rows are 2,107 calls and 5.4% of estimated cost.

Table 1: Calls and estimated cost by phase.

phase	raw calls	raw % calls	raw % cost	unflagged calls	unflagged % calls	unflagged % cost
construction	112	3.16	55.7	110	7.65	58.7
query	3,351	94.6	30.5	1,246	86.7	26.7
maintenance	81	2.29	13.8	81	5.64	14.6
total	3,544		USD 0.135	1,437		USD 0.128

Per call. A construction call costs USD 0.000673 on average and a query call USD 0.0000123, a ratio of about 55. A construction call reads 5,500 input tokens on average and a query call 82, a ratio of about 67. The two ratios differ because the phases use different models and prices: 3,341 of the 3,351 query calls are embeddings, while construction also includes LLM completions, which are billed for output tokens. The query average also depends on which rows are counted. The flagged query calls cost USD 0.0000034 on average and the unflagged ones USD 0.0000274; without the flagged rows, a construction call costs USD 0.000683 and the ratio is about 25.

Concentration. Cost is concentrated in a few large writes. The three most expensive calls, all digest builds, account for 29.7% of all estimated cost. Without them, construction’s share falls from 55.7% to 37.0%. Half of all cost sits in 31 calls, which is 0.87% of the series. A month-long average therefore depends on whether a few large construction jobs happen to fall inside the window. Figure 1 shows the same concentration in time: on 3 of the 19 days with construction, its cost exceeds 10 milli-USD, and those 3 days hold 60.3% of construction cost.

Maintenance. The decomposition in [1] has no maintenance phase. Here maintenance means compaction and consolidation of stored records, not pruning. It is 2.29% of calls and 13.8% of cost, and it ran on 11 of the 32 days.

Construction cost per query. One way to price writing against reading is to divide the construction cost in a period by the number of queries in that period. Table 2 does this for seven-day blocks of the window. The ratio depends on both of its terms. Block 4 lies entirely inside the 13-day stretch with no recorded construction (Section 4), so its ratio is zero although it has queries. In block 2 the construction cost is the same in both columns, yet the ratio is 32.9 or 61.2 μ USD per query depending on whether flagged queries are counted. A per-query construction cost from this table depends on which queries are counted as much as on what construction costs.

Table 2: Construction cost per query call, by block (micro-USD per query). Blocks 1–4 are seven days; block 5 is the last four days.

block	raw queries	raw μ USD/query	unflagged queries	unflagged μ USD/query
1 (Aug 7–13)	1,156	25.7	253	116.5
2 (Aug 14–20)	740	32.9	398	61.2
3 (Aug 21–27)	682	23.6	355	45.3
4 (Aug 28–Sep 3)	360	0.0	19	0.0
5 (Sep 4–7)	413	12.5	221	23.4

Scale. The window cost an estimated USD 0.135 in total, for one shared store serving six agents: USD 0.0042 per day over the 32 days, of which the first covers only 13.3 hours. With an absolute cost this small, the note reports shares and ratios, which are the quantities a comparison with another system would need. Whether they hold in other deployments is not tested here.

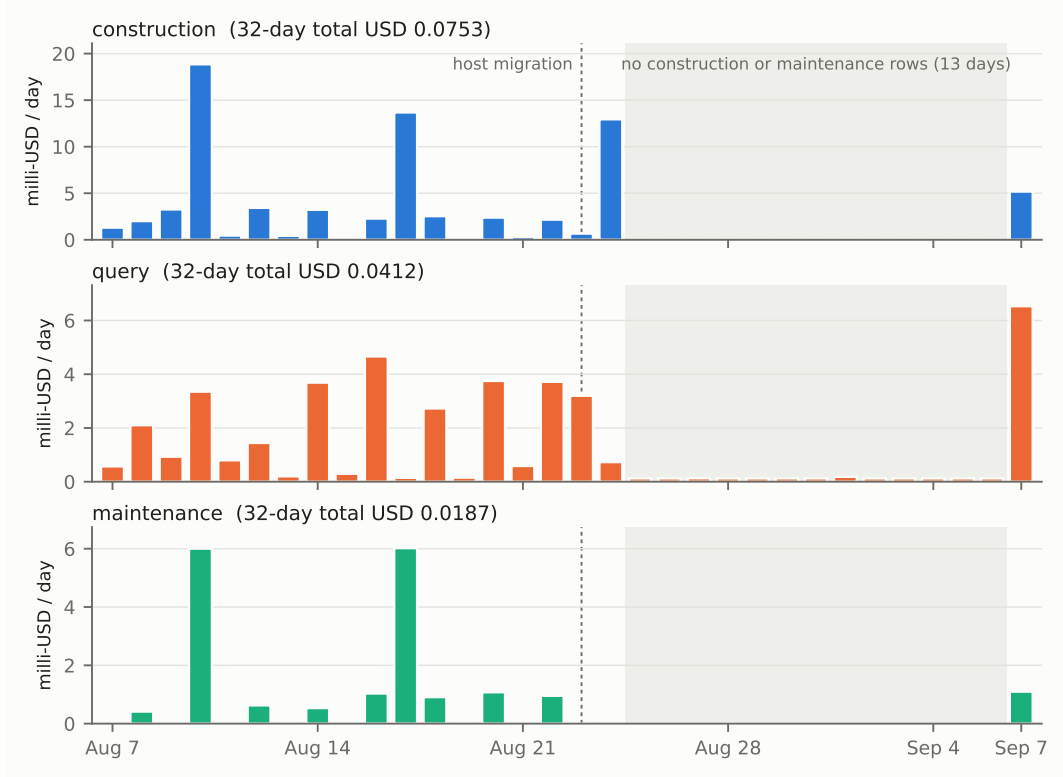


Figure 1: Estimated cost per UTC day, by phase, in milli-USD; each panel has its own scale. Shaded: 2026-08-25 to 2026-09-06, the 13 days with no construction or maintenance rows (D11). Dashed line: host migration on 2026-08-23 (D10). Drawn by `figure.py` from `daily_phase_summary.csv`; days with no row for a phase are drawn as zero.

These numbers agree in direction with [1], where the structure-augmented and agentic systems (their Paradigms III and IV) shift most end-to-end energy into construction. They do not replicate that result: the workload, the cost measure (list price here, energy and wall time there) and the systems all differ.

4. Second reading: what the log contains

Section 3 treats the table as a cost record. Read as a record of the instrument, it limits what Section 3 can support.

Most query calls match a monitoring signature. Table 3 breaks down the 3,351 query rows. 2,105 of them, or 62.8%, carry a flag; the canary signature alone accounts for 1,515, or 45.2%. That leaves 1,246 unflagged query rows, 37.2% of the phase, and these may not all be agent demand either. 485 of them fall between 02:00 and 02:59 UTC on 7 dates whose same hour also has maintenance rows, a pattern consistent with a scheduled job; their source was not verified, so they are not flagged (D2). Query volume and “queries per day” in this table cannot be read as agent demand.

Table 3: Composition of the query phase.

component	rows	% of query rows
canary signature	1,515	45.2
2026-08-10 night window	588	17.5
smoke test	2	0.060
unflagged	1,246	37.2
total	3,351	100

Thirteen days with no recorded writes. From 2026-08-25 to 2026-09-06, 13 days, there is no construction or maintenance row at all; the only rows are queries, and 96.3% of them match the canary signature (D11). The cause was not measured for this deposit, and the recorder can lose rows (D9), so the table shows that no construction or maintenance call was recorded, not that none was made. A reader averaging construction over the window is averaging over that stretch. Construction occurs on 19 of the 32 days and maintenance on 11.

A construction path that is scheduled but absent. Project records schedule LLM knowledge-graph extraction weekly, yet the table holds exactly one `kg-llm.extract` row, and it is the smoke test (D8). The dataset therefore contains no production cost for one of the system’s construction paths. The extraction call site also records only successful calls, so a failure would not leave a row either (D7). The table cannot tell apart a job that ran without provider calls, a job that failed before writing, and a job that did not run.

What a missing row means. The recorder drops errors by design (D9). The canary gives a coarse bound: it fires every 30 minutes, and no gap between consecutive rows exceeds 32 minutes except two, one after deployment and one during the host migration on 2026-08-23 (D10). This bounds outages of the whole recorder. It does not bound single dropped rows.

Failures and fallbacks. 32 calls failed, all query embeddings, on two dates (D6). The construction and maintenance modules have fallback chains to other providers, including a local model, but every row names the same provider (D14). The data are consistent with no fallback being triggered. That was not verified independently.

The two readings meet at one fact: the flagged rows are almost all cheap query embeddings. Removing them moves construction’s cost share only from 55.7% to 58.7%, but cuts the query call count from 3,351 to 1,246 and the per-call cost ratio from about 55 to about 25. Shares of total cost are stable under the filter; anything divided by a call count is not.

5. Limitations

The dataset covers one system, one shared store, one provider and 32 days. Costs are list-price estimates on mostly estimated token counts (D4, D5). Phase is assigned by call site, not by the process that issued the call (D3), and generation is not observed (D12). Annotation flags are signature matches, not verified provenance (D2). The full list of 16 defects, each with its size in rows where it can be measured, is in the dataset README.

Ethics statement

The data are operational telemetry from a system run by the author. They contain no prompts, responses, query text or document content. Their text columns hold only values from closed vocabularies (provider, model, operation, calling code path, phase and token source), which the dataset’s verification script checks.

Data availability

The dataset is at <https://doi.org/10.5281/zenodo.23130191> under CC BY 4.0. The code for this note (`analysis.py`, `parity.py`, `figure.py` and the PDF build) is deposited with this note as `code-v1.0.zip`

and is also in the public repository <https://github.com/totobusnello/memoria-nox>, in `datanote-telemetry/python3 analysis.py --fetch <dir>` downloads the dataset, checks its checksums and recomputes the numbers the note reports.

Declaration of competing interest

The author develops nox-mem.

References

- [1] Y. Omri, Z. Gan, Z. Broveak, R. Geens, Z. He, A. Pentland, M. Verhelst, T. Weissman, T. Tambe. *Agent Memory: Characterization and System Implications of Stateful Long-Horizon Workloads*. arXiv:2606.06448, 2026.
- [2] L. A. Busnello. *nox-mem: Pain-Weighted Hybrid Memory for LLM Agents*. Technical report, version 1.0.6. Zenodo, <https://doi.org/10.5281/zenodo.23147633>, 2026. The dataset README cites version 1.0.2 (10.5281/zenodo.23041503), the version current when the dataset was deposited.