Why Git Is the Memory Solution for the Agentic Development Lifecycle

Git-bound, routed memory for coding agents: structure, episode, and synthesis under a token budget

Frank Guo[1]

[1]Rekal — rekal.dev · guocongmit@gmail.com · github.com/rekal-dev/rekal-cli

Abstract. Coding agents now produce a growing share of a team's code, while the reasoning behind each change — the alternatives weighed, the constraints discovered, the approaches rejected — is trapped in assistant transcripts that vanish with the session. Memory for this setting, the agentic development lifecycle (ADLC), is usually posed as one retrieval problem and built as machinery: tiered stores, memory graphs, compiled wikis, model-judged admission. We argue memory should instead be **git-bound** — built into the repository's version control, inheriting the guarantees the machinery struggles to construct: ground truth from commits, freshness from rebuild, verification from the merge, containment from review. On this ledger we solve two problems separately, then combine them. **Seed supply** is closed as an eight-corpus retrieval study under a pre-registered ship discipline: five imported ranking mechanisms rejected, two kept, and a best configuration of ≈0.31 pooled MRR — ≈60× the raw-transcript grep floor, ≈15× an honest parsed-turn floor. **Answer assembly** is where ranking stops helping: single-shot retrieval scores only 0.07–0.20 answer-sufficiency on real developer questions, and ungated episode injection measurably degrades good answers. A **router** dispatches breadth to a git-anchored structural map, pointed lookups to confidence-gated episodes, and rationale to **decision synthesis**, which reconstructs why-arcs no single session contains (0.83 sufficiency on a young ≈50k-LOC production system). Routed, the system answers at **382–980 tokens per question** — three orders of magnitude below the recorded history. Because ground truth is mined from commit–session links rather than annotated, every result is replicable on any user's own history at zero labeling cost. The remaining constraint is capture. Code, benchmark, and paper source: github.com/rekal-dev/rekal-cli.

1 The question an agent actually asks

Code has a ledger; intent does not. Version control records every line a team ships, but the reasoning that produced those lines — explored designs, rejected alternatives, the correction a reviewer shouted mid-session — lives in AI-assistant transcripts that expire with the terminal window. An agent that cannot remember what its own team already tried will confidently re-propose it.

We call this setting the **agentic development lifecycle** (ADLC): git still runs the code's lifecycle, but a growing share of the intent behind each change is produced in agent sessions git never sees. The title's claim is the paper's thesis, argued and then measured: the memory solution for the ADLC is git itself — **bound**, not bolted on. Bound, because the lifecycle's own primitives supply what memory machinery cannot (ground truth, freshness, verification, containment; §3). And routed, because once the ledger exists, answering from it is not one retrieval problem (§7).

Start from what memory **is** for an agent. The context window is the scarce resource, so memory is not a store — it is **context assembly under a token budget**: for each question, deliver the few thousand recorded tokens that change the answer, out of millions [1]. The dominant framing then takes one more step that we refuse: it treats "each question" as one retrieval problem, embeds the corpus, ranks sessions, and scores success by Mean Reciprocal Rank against a single gold session.

Watch what developers actually ask, and the framing breaks into three kinds:

- **Breadth** ("how is the data-processing pipeline architected end-to-end?"). The answer is spread across many sessions and the code tree; no single episode contains it. Episodic recall floors here by construction.
- **Pointed** ("which session implemented the validation layer, and how?"). The case retrieval is built for — a specific episode answers.
- **Rationale** ("why was the batch execution engine chosen instead of the managed inference service?"). The answer is a **decision that evolved** across sessions and was never written down in one place. Single-shot retrieval returns one fragment of an arc.

These kinds need different mechanisms, and a system that ships only the middle one fails most of the population. The paper therefore solves **two problems, separately, then combines them**. Problem one is **seed supply**: can the right prior sessions be found at all? We close it as a retrieval study — honest floors, a disciplined mechanism sweep, two validated levers (§5). Problem two is **answer assembly**: given seeds, can the question actually be answered within a budget? We close it with three modes and a router (§7–8). The combination — one engine, one skill-layer router

— outperforms every single-mechanism alternative on coverage of the question kinds, at a fraction of the token cost of the strongest single mode.

Contributions. (1) The **git-bound** position: a memory engine built into version control whose guarantees — annotation, staleness, self-confirmation, contamination — are inherited by construction (§3), and a self-labeling benchmark that construction makes possible (§6). (2) A **closed seed stage**: an eight-corpus retrieval study under a strict incumbent-versus-candidate discipline [2] that establishes honest grep floors, rejects five imported ranking mechanisms with confidence intervals, validates two (per-corpus tuning; a facet term ported from SPM [3]), and derives testable rules for when each helps (§5). (3) The **routed three-mode architecture** — structural map, gated episodes, decision synthesis, shipped as one binary plus three agent skills — and an **answer-sufficiency** evaluation showing each mode wins a different question kind, ungated injection harms, and synthesis reconstructs decision rationale that no retrieval variant reaches (§7–8). (4) The token-economics reading: full-kind coverage at 382–980 tokens per routed question (§8.3).

2 A worked example

On Monday, an engineer and an agent rework webhook delivery. Mid-session the engineer interrupts: **"don't retry on a fixed delay — it stampedes the downstream on recovery."** The agent switches to exponential backoff with jitter; the change merges. The post-commit hook captures the session — turns, tool calls, the interruption tagged `human_steering` — scrubs secrets, and appends it to the repo's ledger with a link to the commit SHA. Nobody writes documentation.

On Friday, three different questions arrive, and the router sends each to a different place. **"Should webhook retries use a fixed delay?"** is pointed: episodic recall returns the Monday session with the steering turn as the top snippet, confidence clears the gate, and a bounded drill reads a five-turn neighborhood — about 1,500 tokens. **"How does delivery work end-to-end?"** is breadth: the structural map — a condensed subsystem map generated from the repository at its current SHA — answers from structure; no episode is fetched. **"Why exponential backoff instead of a delivery queue?"** is rationale: a decision-scoped gather collects the decision-relevant turns across sessions (the steering turn, the alternatives discussed before it, the constraint that forced the change) and one synthesis call reconstructs the arc, citing each claim's turn and commit. Same ledger, three mechanisms, each bounded to what the question warrants.

3 Git-bound: the inherited guarantees

Prior memory systems — tiered stores [4], memory graphs [5], compiled wikis [6], guarded by model-judged admission [7] and evaluated on annotated benchmarks [8] — import four problems they then cannot discharge. In the software-engineering setting each has an answer that is a git primitive every team already runs, **by construction**:

1. **Annotation** — where does supervision come from without human labels? → The **commit**: a post-commit hook links sessions to the verified change they produced; ground truth is mined, not annotated (§6).
2. **Staleness** — how does derived memory stay current? → **Rebuild and diff**: the index is disposable and regenerated from the append-only ledger; compiled structure (including §7′s map) is a function of the tree at a SHA, refreshed by diffing, its drift surfaced as a reviewable `git diff`.
3. **Self-confirmation** — who verifies what enters shared memory, if not the model judging its own homework? → The **merge**: only checkpoints whose commit landed on the default branch are exportable; review + CI
   1. merge are the external verifier.
4. **Contamination** — how does memory cross a scope boundary? → **Code review** is the sole audited egress; cross-repo memory is index-only and structurally unpushable [9], [10].

We take annotation as given by the self-labeling method below and focus the empirical work on the three-mode architecture and its routing. The long-form defense of the four guarantees is in the superseded substrate report (repository history); the design survives here as the platform the modes stand on.

4 The ledger and the engine

Rekal is one binary embedding its database engine, embedding models, and compression dictionary; git is the only wire. A post-commit hook parses the active assistant session(s) — adapters cover four agent CLIs — deduplicates turns, scrubs secrets and anonymizes paths **before any byte is stored** [10], and appends to an append-only ledger (`data.db`, committed via a per-author orphan branch under the merged-only gate). A derived index (`index.db`) is rebuilt locally. The ledger records what was **said** — conversation turns with high-signal roles preserved (`human_steering` corrections, out-of-band compaction `summary` distillations) — and what was **touched**: tool names, file paths, command output, and the commit SHA. Crucially, it does not store diff content: code content is reconstructed from the commit SHA on demand, so intent lives in the ledger and content lives in git (thin on the wire; the division of labor §9 defends).

Recall scores sessions by a weighted hybrid over the parsed turns — BM25 full-text, latent-semantic, and neural cosine — with per-role boosts (steering, summary), a subagent down-weight, and an opt-in **facet** term (§5.3). All weights apply at query time; no reindex to change them. Ship defaults: layer mix 0.35/0.10/0.55, steering boost 1.3, summary boost 1.15, subagent 0.7, max-norm, facet 0.3 (the held-out tuned value; the layer fails soft on corpora without

facet material, and an explicit 0 restores the byte-identical pre-facet engine).

5 Problem one: seed supply, closed

This section reports the retrieval program to its end and states what it does and does not buy. Protocol throughout: self-labeled gold (§6), 10% dev / 90% held-out test, and the incumbent-versus-candidate discipline of [2] — a candidate configuration ships only if it beats the incumbent on the held-out split under a paired bootstrap CI excluding zero.

Corpora. Eight real working corpora spanning four workload classes, anonymized: **Corpus A** (a documentation/knowledge base of ≈4,000 markdown documents; ≈1,400 captured sessions; prose-heavy, diverse tooling; retrieval n_test=212) and **Corpus B** (a production software system of ≈50k lines of code; n_test=456–521 by split) are the two with statistically clean splits; six smaller corpora (code, docs/architecture, notes, build-scripts, two mixed) are reported directionally and flagged where tuning would overfit.

5.1 The floors: parsing is most of the miracle

On Corpus A, term-frequency grep over the **raw** transcript JSONL scores 0.005 pooled MRR. Even at shipped defaults the hybrid over the parsed ledger beats it 37× (0.187; 57× on nDCG@10, non-overlapping bootstrap CIs); under the **best** held-out configuration (§5.2, ≈0.31 — derived from the tuned baseline plus the facet marginal) the gap is **≈60 ×**. Raw transcripts are adversarial retrieval material (tool dumps, base64, duplicated sidechains); parsing the history into attributed turns is what makes it searchable at all. That result alone, however, is a floor against weak material — so we also report the stronger **parsed-turn grep** floor (term-frequency rank over the same turns the index sees) on all eight corpora. The hybrid dominates it everywhere: 0.021 → 0.225 (≈11×; **≈15×** at the best configuration) on Corpus A, 0.028 → 0.159 (≈6×) on Corpus B, and on every small corpus besides (floors 0.08–0.19 vs hybrids 0.21–0.58). Ranking still earns its place on parsed turns; grep does not close the gap once parsing is granted.

5.2 The mechanism study, and the best configuration

Seed stage (pooled MRR, held-out)	**Corpus A**	**Corpus B**
grep, raw transcript JSONL (floor)	0.005	—
grep, parsed turns (honest floor)	0.021	0.028
BM25-only	0.200	0.148
hybrid, shipped default	0.225	0.159
best held-out config: tuned hybrid + facet	**≈0.31**	**≈0.24**
Mechanism sweep (paired bootstrap CI; detail in Appendix A)		
rejected: RRF · temporal decay · lexical dilution · z-norm · embedder swap	all n.s. or degradation	
kept: per-corpus weight tuning	B +0.032 [.016,.049]	
kept: SPM facet term [3] (fb=0.3 on both)	A +0.110* B +0.053*	

Table 1: Problem one closed: floors, the mechanism study under the RHO discipline [2], and the best held-out configuration — ≈15 × the honest grep floor on Corpus A. Five imported mechanisms rejected with intervals; two kept. Best-config cells are derived from the tuned baseline plus the facet marginal (exact cells in the run record). Negative results are load-bearing: they close the stage.

Three findings organize Table 1. **Hybrid beats BM25 where paraphrase opens a surface-form gap and not elsewhere**: pooled on Corpus A the two sit within each other's CIs (0.187 vs 0.171 on the primary label run), the separation is a provenance effect (T1 MRR 0.301 vs 0.248; R@5 0.537 vs 0.425), and the only corpus where the hybrid wins outright is the noisy-commit prose corpus (+0.352 [.158,.549]) — on exact-vocabulary code corpora it ties.

The facet term is the one imported mechanism that survives end-to-end. SPM [3] derives structured facets (task, data schema, tool config, output constraints) for session-start context assembly; we port the idea to a retrieval term: a deterministic per-session facet document (distinct tool paths + command prefixes + steering text; no LLM, built at index time), BM25-searched as a fourth, orthogonal hybrid term — additive and config-gated, so at the default facet_boost=0 the engine is byte-identical to the baseline. Root cause of the win: the answer to "what tools/config did session X use" lives in tool-call **metadata** that a session's conversational turns often never mention. Bolted naively onto the untuned default mix the term looks corpus-conditional (significant on A, null on B); the joint re-tune — one never ships an untuned knob — shows the marginal is significant on **both** corpora, with magnitude monotone in tool-path diversity (Table 5, Appendix A).

The embedded model is not the bottleneck. Substituting a strong hosted general-purpose embedder, under a protocol that gives it every advantage, moves the neural-only ablation slightly and the shipped hybrid not at all (n.s. on both corpora; Table 6, Appendix A): **alignment, not**

embedding quality, is the binding constraint [11], and the local-first default — embedding model inside the binary, no API — costs approximately nothing in recall.

5.3 What the mechanism study teaches

Every pure re-ranking mechanism failed; the single retrieval gain came from adding an **orthogonal evidence layer** for a **kind of question** the turn index structurally misses. Gains come from coverage of question kinds, not rank polish. The seed stage is hereby closed — retrieval is a solved-enough seed supplier with two validated levers and characterized limits — and the rest of the paper applies the same lesson above retrieval, where the layers are no longer index columns but memory **modes**.

6 Ground truth without annotation

No benchmark exists for repo-grounded intent recall: conversational suites [8] evaluate chat personas; IR suites evaluate document QA. RekalBench's defining property follows from the git binding: **the corpus labels itself.** Every checkpoint records which sessions produced which commit; a SQL miner over ledger plus git topology emits gold pairs with no human in the loop — provenance (commit → producing session), decision recall (steering turns), dead ends (never-merged branches), multi-hop (file-co-occurrence session pairs). An LLM paraphrases each label's context into a natural question under a 4-gram Jaccard ≤0.30 leakage ceiling. Supervision cost: zero. The retrieval study of §5 runs on these labels (415 pairs on Corpus A alone; T4 multi-hop supplies 92–104 pairs each in the three high-co-occurrence corpora); label noise biases scores **down**, not up, so the numbers are conservative. Because labels are mined, the entire harness is public, fully local, and runnable by anyone on their own store.

7 Problem two: answer assembly — three modes and a router

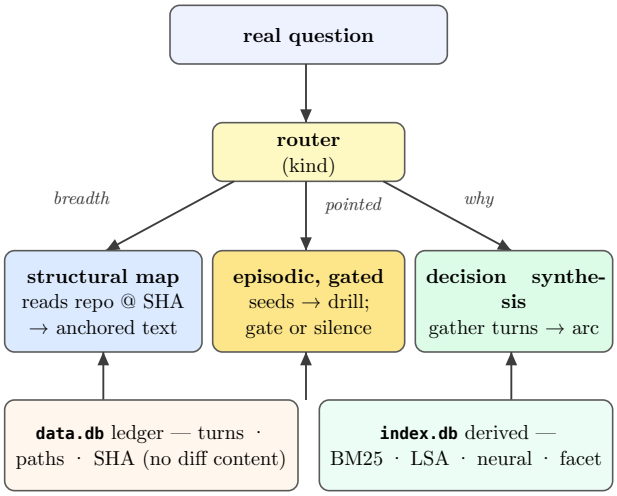

Figure 1: One git-native ledger supplies seeds and structure; a router sends each question to one of three modes by kind. Code content is reconstructed from the commit SHA on demand, so the ledger stays thin. Episodes are confidence-gated before they join the map.

7.1 Structural map (breadth)

The map is a condensed subsystem description **authored by an LLM that reads the repository** — its directory skeleton, its own README/architecture documents — not clustered from co-occurrence statistics (we tried; the statistical version grouped tool-call-id dumps and scratch files into meaningless clusters; comprehension filters what statistics cannot). Its reader is an agent, so its format is structured text with greppable path anchors rather than a diagram: subsystems cut by responsibility with purpose stated as behavior, load-bearing edges labeled with what crosses them, and the main flows — dense per token, and section-scoped so refreshes rewrite only what changed. The map is watermarked with the HEAD commit and regenerated on demand, so it is never a stale snapshot: it is a function of the tree at a SHA, refreshable by diffing only the sections whose files changed — the staleness guarantee of §3 applied to compiled structure. It answers "what exists and how it connects," and nothing about "why."

7.2 Episodic recall, confidence-gated (pointed)

For pointed questions the engine returns top-k seed sessions and the agent drills into the cheapest dense anchor (a summary turn, else a turn window). The critical design point is that episodes must be **gated**: we calibrated a hit signal from the retriever's own per-layer scores — top-1 score and top-1–top-2 gap separate hits from misses (means 0.91 vs 0.87; gap 0.046 vs 0.017) — and inject episodes only when the signal clears the bar. Ungated, low-confidence episodes **poison** a good map (§8.2).

Mode	**Overall** (95% CI)	**Broad**	**Pointed**	**Why**	**Avg tok**
Corpus A — documentation/knowledge (≈4,000 markdown docs, ≈1,400 sessions)					
structural map	0.33 (.13–.53)	0.50	0.17	0.33	201
episodic single-shot	0.20 (.00–.40)	0.33	0.17	0.00	914
routed (map + gated episodes)	0.43 (.20–.67)	0.83	0.50	0.17	382
decision synthesis	0.47 (.20–.67)	0.50	0.50	0.33	2762
Corpus B — production software system (≈50k LOC, young history)					
structural map	0.53 (.30–.77)	0.50	0.33	1.00	969
episodic single-shot	0.07 (.00–.27)	0.00	0.00	0.33	648
routed (map + gated episodes)	0.60 (.37–.80)	0.67	0.42	0.83	980
decision synthesis	**0.83 (.67–.97)**	0.92	0.92	0.50	3135

Table 2: Answer-sufficiency by mode on both corpora (n=15 real questions each, tagged broad/pointed/why; blind judge; bootstrap 95% CI on the overall). With this sample size the overall point estimates carry wide intervals and most pooled contrasts are **not** statistically separable — read the results as directional patterns across question kinds, not a ranking of modes. The routing signal is in the per-kind columns: episodic-alone floors on breadth and near-floors overall on the uniform young corpus (0.07); the map answers breadth but not why; synthesis dominates the corpus whose decisions are recent. Routing is no worse than the best single mode on every kind and strictly better on breadth, at 382–980 tokens.

7.3 Decision synthesis (rationale)

For "why" questions the agent does not retrieve one session; it **gathers every decision-relevant turn** — a direct query over the ledger for the choice, its alternatives, and reasoning markers ("because", "instead of", "constraint", "rejected") — then **synthesizes the arc**: original design → alternatives rejected → the constraint that forced the change → final rationale, with every claim carrying its turn and commit pointer (the self-confirmation guarantee applied to generated text). Where the reasoning references code, the diff is pulled from the commit SHA at synthesis time. This is the mode single-shot retrieval cannot emulate, because the answer is **distributed** and was never a single record.

7.4 The router is a skill: gated triage over workflows

The router and its three modes ship as **one** agent skill — written workflow playbooks, internally routed, driving engine primitives — so the agent loads a single policy. Its first decision is not which memory mode but **which substrate**: present-tense code facts belong to the tree (grep and read at HEAD) and are deliberately not answered from memory — a recalled episode about how X **used** to work can mislead about how it works now; past-tense intent belongs to the ledger; structure belongs to the map, which bridges the two (derived from the tree, serving memory). Grep the tree for present tense, recall the ledger for past tense — and stay silent when it is neither. Within the ledger, a **triage** step then classifies the question's kind from its shape ("how does X work end-to-end" → breadth; "which session did X" → pointed; "why X instead of Y" → rationale), a **gate** decides whether episodic evidence enters at all (the confidence signal of §7.2 — below the bar, the mode stays silent rather than injecting noise), and each mode is then a **workflow**: map → regenerate-on-diff and answer from structure; hunt → seeds, gate, drill the cheapest dense anchor; why → gather decision turns, pull diffs by SHA, synthesize with pointers. Nothing in this pipeline is a trained component: triage rules, gate thresholds, and workflows are inspectable text, versioned in git like everything else, so improving the routing policy is editing a file under review — the same maintenance story as the rest of the system. Rationale lives in turns; structure lives in the tree; the modes are **routed, not stacked**.

8 Evaluation: answer-sufficiency, not rank

We reject single-gold MRR as the headline. Our metric is **answer-sufficiency**: for a real question, assemble context under a mode, have a distinct blind judge rate whether the context is SUFFICIENT (1), PARTIAL (½), or INSUFFICIENT (0) to answer, and report the mean with bootstrap 95% CIs, plus average tokens per question. Ground truth for the corpora is self-labeled; questions are real developer questions tagged **broad / pointed / why** (n=15 per corpus; per-question judgments and generated maps are in the run directory). Corpus A is the documentation/knowledge corpus of §5 (≈4,000 markdown documents); Corpus B is a production software system of ≈50k lines of code — a layered data/ML pipeline (ingest → transform → model build) inside a ≈3,000-session repository — with uniform tooling and a **young** recorded history: the harder test, memory that has barely accumulated.

8.1 Each mode wins a different kind

Table 2 is the study's main table. We stress up front that with 15 questions per corpus, a single judge, and wide bootstrap intervals, the overall point estimates are **not** statistically separable — most CIs overlap. We therefore read the results as directional patterns across question kinds, not as a ranking of modes; the routing signal is in the per-kind columns, not the overall.

Stage added	What it adds	Sufficiency (A / B)	Tokens/question
grep, raw transcript JSONL	nothing — the floor (0.005 MRR)	≈0	unbounded scan
grep, parsed turns	weak seeds (0.021/0.028 MRR)	—	unbounded scan
best seed config: tuned hybrid + facet (≈0.31 MRR)	pointed answers; seeds for every mode	0.20 / 0.07 (single-shot)	≈0.9k; ≈1.5k with drill
+ structural map	breadth	0.33 / 0.53	201 / 969, amortized regen
+ decision synthesis	rationale, multi-hop	0.47 / **0.83**	≈2.8–3.1k
routed: map + gated episodes	**all kinds at the per-kind floor**	**0.43 / 0.60**	**382 / 980**

Table 4: Coverage at cost — the two problems combined. Seed supply makes the ledger findable (grep floors → ≈0.31 best config: ≈60× raw, ≈15× the honest floor); answer assembly converts findable into answered, kind by kind; routing buys the per-kind floor at 382–980 tokens against a recorded history of tens of thousands of turns. Synthesis buys the hardest kind at ≈3× the routed price — cost follows the question. With the question-kind distribution of real usage (mined from the ledger's own query log; instrumented, future run) the last row becomes one expected-cost figure: $\mathrm{E}[\text{tokens}] = \sum_{\text{kind}} p(\text{kind}) \cdot \text{cost}(\text{mode})$.

8.2 Gating and the rationale ablation

Arm	Result	Note
Episode gating, isolated (Corpus B, n=12)		
episodic single-shot	0.00	near-floor; uniform pipeline
structural map	0.63	reads the real layered architecture
map + episodes, **ungated**	0.29	low-confidence episodes **poison** the map
map + episodes, **gated**	0.50	gate suppresses 11/12 bad injections
The execution-engine rationale question (sufficient?)		
structural map	partial	structure, not rationale
the source code itself	no	uses the engine; no why
episodic single-shot (top-3)	no	returns one fragment
synthesis, under-gathered (4 terms)	no	too few turns gathered
synthesis, adequate gather (30 turns)	**yes**	blind judge; 2.1k tok

Table 3: Corpus B ablations. Top: gating isolated on a separate question set — ungated episodes degrade a good map and the confidence gate recovers most of the loss (recovering 0.29→0.50 of the 0.63 map baseline; only 1 of 12 retrievals cleared the gate, so gating here mostly means silence). Bottom: the mode ablation on one evolved architectural decision — only a decision-scoped gather plus synthesis answers it.

Three lessons. **Episodes must be gated, or they degrade the map**: naive context-stuffing — the integration everyone builds first — is measurably harmful, and knowing when to stay silent is part of the system. **Decision synthesis is the most distinctive mode**, and its quality is bounded by the **gather**, not the synthesizer: a weak four-term gather starved the same model into INSUFFICIENT; an adequate gather (30 decision-relevant turns, ≈2.1k tokens, one synthesis call) reconstructed the full arc for a design that began under one constraint set and drifted to "good enough." The lesson is precise: **the rationale was in memory all along; single-shot retrieval fragments it and a poor gather starves it, but a decision-scoped gather plus synthesis assembles it.** And **the young corpus is the strong case for memory**: a barely-accumulated history already answers 0.83 of real questions under synthesis — memory pays off long before it is big.

8.3 The economics: coverage at cost

Table 4 is the paper in one table: each stage added covers a question kind the previous stages missed, and the combined routed system answers at a cost three orders of magnitude under the recorded history — with cost following the **question**, not the corpus.

9 The real bottleneck is capture, not ranking

Every failure in §8 that was not a routing error was a **capture gap**: the answer had never been verbalized in the ledger. This reframes the research target. The ledger deliberately keeps only what was said plus what was touched (git SHA; no diff content) — thin to stay on the wire — and synthesis pulls code from the commit on demand. Our finding is that the division of labor is correct but incomplete on one side: intent lives in the ledger and content lives in git, so the remaining loss is reasoning that agents never say out loud. Keeping capture thin is therefore not a limitation to walk back — it preserves the git-bound answers to staleness and contamination that a fatter, content-bearing ledger would compromise — but capture completeness (verbalization rate, measured per corpus) is the binding constraint on everything above it, and the next unit of improvement per token spent lies there, not in ranking.

10 Related work

Memory-graph and tiered-store systems [4], [5], [12] and self-evolving retrieval over compiled wikis [6] target long-horizon recall but assume annotation, maintain compiled structure in software, and admit via model judgment — the four §3 problems. The data-management evaluation [11] reaches the verdict this paper takes literally: effec-

tiveness depends on aligning memory structure with the workload; the coding workload's structure is the repository, and Rekal aligns by construction. On the compression axis we side with preservation-with-attribution [13] (raw turns, harvested summaries) over write-time consolidation [4]; on the retrieval axis with active reconstruction [5] (the skills drive search–facet–zoom–drill loops); the storage is flat indexes joined at query time [14]. Against retrieval-free direct corpus interaction [15], [16], our floors sharpen RISE's argument [1] at the retrieval layer. SPM [3] contributes the structured-facet thesis our seed stage validates in a different role (retrieval term, not context assembly). The retrospective-optimization discipline [2] governs every shipped mechanism. Our departure is threefold: (i) git as the source of ground truth, freshness, and the sharing gate rather than rebuilding them; (ii) single-gold MRR replaced, as the headline, by routed answer-sufficiency; (iii) decision synthesis over the episodic trail identified as a distinct memory mode, not a retrieval variant.

11 Limitations

The sufficiency evidence is small: n=12–15 questions per corpus, one blind judge, one execution model; we report CIs and treat magnitudes as directional, and the honest summary of Table 2 is per-kind patterns, not separable pooled rankings. Answer-sufficiency is a proxy for task help, not task completion. The map is only as good as the repository's own structure and documentation; synthesis reconstructs rationale only to the extent it was verbalized (§9); the judge saw no trail — model and question sets are modest and the harness supports scaling both. The seed-stage study is one operator's corpora; because labels are mined, replication is free for any user on their own history. Specified but not run here: synthesis on the mined T4 multi-hop gold (92–104 pairs per high-co-occurrence corpus), a second judge model with agreement, the wild-question kind-distribution (and with it the expected-cost figure), and gate recalibration after the facet term enters the score mix.

12 Conclusion

Agent memory is not one retrieval problem but three modes — structure, episode, and synthesis — over a ledger whose hard guarantees come from version control. The paper's shape is deliberate: two problems solved **separately** — seed supply, closed as a retrieval study with honest floors, a mechanism graveyard, two validated levers, and rules for when each helps; answer assembly, closed with three modes behind a skill-router — and then **combined**, where the combination outperforms every single-mechanism alternative on coverage of the question kinds at a fraction of the strongest mode's token cost (Table 4). The value above ranking is in routing each question to the mode that can answer it, gating episodes on confidence so they help rather than harm, and reconstructing evolved decisions by synthesis rather than retrieval. The binding constraint is capture. One tool, three modes, git-bound — and the strongest result is the one the field has not been measuring: memory can reconstruct **why** a system became what it is. Git already runs the ADLC's code; bound and routed, it runs its memory too.

Reproducibility. Every value traces to a committed run record (per-mode sufficiency judgments, blind-judged synthesis runs, generated maps, gating ablation; retrieval matrix, mechanism sweeps, facet screens) — anonymized aggregates under `docs/research/runs/`. Corpora are anonymized (A: a documentation repository; B: a production data/ML pipeline subsystem); no session content or corpus identity leaves the operator's machine — published artifacts are aggregates, prompts, and code. The router and its three mode workflows ship as one agent skill (`rekal`); the judge is a single automated model and the questions are the authors' own corpora — the study is a within-system characterization, not a competition. Engine, skills, benchmark spec, extraction SQL, and this paper's source: github.com/rekal-dev/rekal-cli (`docs/research/`).

Appendix A: seed-stage detail

The facet term at two operating points. Evaluated by bolting it onto a fixed configuration, a mechanism can appear corpus-conditional when it is merely mis-tuned; only the joint re-tune separates "does not work here" from "was not given its operating point." What remains genuinely corpus-conditional is the **magnitude**: the marginal is monotone in tool-path diversity (the high-diversity corpus gains +0.110; the uniform-pipeline corpus +0.053, at ≈2.3× less path diversity per call) — a testable deployment rule. The term ships enabled at the tuned facet_boost of 0.3; corpora without facet material pay nothing (the facet index is guarded), and an explicit 0 restores the byte-identical pre-facet engine. We test the retrieval port only, explicitly not a reproduction of SPM's task-completion result, which would need a curated gold set.

Facet term, by operating point	**Corpus A**	**Corpus B**
screen: facet-doc BM25 vs turn recall (structural queries)	+0.184 [.05,.32] **sig**	+0.016 n.s.
end-to-end, **bolted onto** the default mix (fb=0.6)	0.179→0.294 +0.116 [.04,.20] **sig**	−0.019 n.s.
joint re-tune (fb tuned with the mix; fb=0.3 selected on both)	marginal +0.110 [.056,.173] **sig**	marginal +0.053 [.012,.107] **sig**

Table 5: The facet term at two operating points (held-out, paired bootstrap CIs). Naively bolted on, it looks corpus-conditional; jointly re-tuned — one never ships an untuned knob — the marginal is significant on both corpora. The naive null was an artifact of an untuned operating point, not an absent effect.

Embedding options. The substitution protocol gives the alternative its best shot: same held-out split and candi-

date pool; asymmetric query/document input types sent natively; the content-hash embedding cache rebuilt per model; four configurations per embedder (neural-only and hybrid, each at default and dev-tuned weights); paired per-query bootstrap.

Configuration (pooled MRR)	Corpus A	Corpus B
BM25-only	0.200	0.148
neural-only — embedded (local)	0.080	0.045
neural-only — hosted	0.093	0.055
hybrid — embedded, default mix	0.225	0.159
hybrid — embedded, dev-tuned mix	0.211	**0.182**
hybrid — hosted, default mix	0.217	0.140
hybrid — hosted, dev-tuned mix	**0.229**	0.170

Table 6: Embedding options on the held-out split. The hosted embedder (a large general-purpose API model) lifts the **neural-only** ablation slightly on both corpora, but the **hybrid** — the shipped system — is flat on A and worse on B (hosted-default 0.140 falls below BM25-only); all hybrid deltas n.s. under paired per-query bootstrap. Scope: one strong general-purpose alternative tested; a code-tuned embedder is the open falsifier and runs through the same gate for free (content-hash cache; query-time weights).

References

[1] S. Zhuang and others, "Towards Retrieving Interaction Spaces for Agentic Search." [Online]. Available: https://arxiv.org/abs/2606.06880

[2] W. Pan and others, "Evolving Agents in the Dark: Retrospective Harness Optimization via Self-Preference." [Online]. Available: https://arxiv.org/abs/2606.05922

[3] Apple Machine Learning Research, "Shared Selective Persistent Memory." [Online]. Available: https://arxiv.org/abs/2607.09493

[4] S. Yan, J. Ni, L. Zheng, and others, "AdaMem: Adaptive User-Centric Memory for Long-Horizon Dialogue Agents." [Online]. Available: https://arxiv.org/abs/2603.16496

[5] S. Ji, Y. Li, and B. Hooi, "Memory is Reconstructed, Not Retrieved: Graph Memory for LLM Agents." [Online]. Available: https://arxiv.org/abs/2606.06036

[6] H. Ming, F. Li, X. Wu, W. Que, and others, "Retrieval as Reasoning: Self-Evolving Agent-Native Retrieval via LLM-Wiki." [Online]. Available: https://arxiv.org/abs/2605.25480

[7] S. Zhu, Y. Qi, Y. Wang, and others, "Escaping the Self-Confirmation Trap: An Execute-Distill-Verify Paradigm for Agentic Experience Learning." [Online]. Available: https://arxiv.org/abs/2606.24428

[8] A. Maharana and others, "Evaluating Very Long-Term Conversational Memory of LLM Agents." [Online]. Available: https://arxiv.org/abs/2402.17753

[9] Survey authors, "A Survey on Long-Term Memory Security in LLM Agents: Attacks, Defenses, and Governance Across the Memory Lifecycle." [Online]. Available: https://arxiv.org/abs/2604.16548

[10] State-contamination authors, "State Contamination in Memory-Augmented LLM Agents." [Online]. Available: https://arxiv.org/abs/2605.16746

[11] W. Zhou and others, "Are We Ready For An Agent-Native Memory System?." [Online]. Available: https://arxiv.org/abs/2606.24775

[12] S. Wu, H. Zhu, Y. Zhang, X. Wang, and S. Yeung-Levy, "AutoMem: Automated Learning of Memory as a Cognitive Skill." [Online]. Available: https://arxiv.org/abs/2607.01224

[13] S. Zhou and J. Han, "A Simple Yet Strong Baseline for Long-Term Conversational Memory of LLM Agents." [Online]. Available: https://arxiv.org/abs/2511.17208

[14] Y. Wu, J. Li, X. Liang, and others, "SAG: SQL-Retrieval Augmented Generation with Query-Time Dynamic Hyperedges." [Online]. Available: https://arxiv.org/abs/2606.15971

[15] Z. Li, H. Zhang, and others, "Beyond Semantic Similarity: Rethinking Retrieval for Agentic Search via Direct Corpus Interaction." [Online]. Available: https://arxiv.org/abs/2605.05242

[16] GrepSeek authors, "Training Search Agents for Direct Corpus Interaction." [Online]. Available: https://arxiv.org/abs/2605.29307