

When does shared memory help AI coding agents, and what does it cost?

Measured results from about 890 graded agent runs with and without Big Agentic Brain (BAB), an open-source shared memory for agents.

100% → 100%

Success on 12 easy bugs solvable from the code alone, with and without memory. There, memory only added cost (1.50× per run, Sonnet).

12% → 79%

Success on 8 tasks that need team knowledge: no memory vs default BAB (Sonnet, n = 24 runs each, p = 0.031).

100%

Success with a one-line search-first nudge or with automatic retrieval before the first turn.

8,598 → 648

Tokens added to every request: full toolset vs a lean 3-tool setup (75% success at 1.18× baseline cost).

Summary

AI coding agents begin every session with an empty context. Team knowledge that never made it into the repository, such as a vendor's undocumented header change, a bank-holiday rule or an ID check-digit scheme, has to be rediscovered each time or is simply unavailable. We tested whether a shared, self-hosted memory changes that, and what it costs, using Big Agentic Brain (BAB) as the memory layer and Anthropic's Claude Sonnet 5.5 and Haiku 5.5 as agents.

888

graded agent runs, all on 19
Oct 2026

20

bug-fix tasks (12 core + 8
team-knowledge)

≈ \$11.43

total measured API spend
(\$2.545 + ~\$8.88)

2

models: Sonnet 5.5
(primary), Haiku 5.5
(secondary)

What we found

- 1. If the code already contains the answer, memory is overhead.** On 12 short bug tasks, every condition solved every task (108/108 Sonnet runs, 180/180 Haiku runs). Adding BAB raised Sonnet's cost per run 1.50× (95% CI 1.36–1.69). Most of that comes from a fixed 8,598 tokens of tool definitions and instructions sent with every request. Agents rarely used the memory anyway: only 22% of Sonnet runs searched.
- 2. If the answer is team knowledge, memory decides the outcome.** On 8 tasks whose fix depends on what a colleague learned elsewhere, Sonnet without memory succeeded in 12% of runs (3/24). With default BAB it succeeded in 79% (19/24). The paired difference was +0.67 (CI +0.33 to +0.92, $p = 0.031$).
- 3. Integration design matters as much as memory.** A one-line "search first" nudge, or injecting the top 3 search results before the first turn, took success to 100% (24/24 each). A lean 3-tool setup cut the per-request overhead by about 92% and reached 75% success at 1.18× baseline cost, against 1.51× for the default.
- 4. Stale memory is a real hazard.** With only outdated facts in memory, success fell back to the no-memory baseline (12%), and agents flagged the stale entry in only 4% of runs. When current and stale facts were both present, success matched the default (79%).
- 5. Retrieval degrades with scale, mostly for vague queries.** Overall recall@5 fell from 0.93 at 20 distractor memories to 0.81 at 1,000. Paraphrased queries dropped to 0.64. Following BAB's own suggested `kind` filter returned nothing at any scale (recall 0), because that filter excludes the `learning` memories that agents write.

Read these results with their limits in mind

All tasks are synthetic and were written by us. Samples are small (8 team-knowledge families, 3 trials each). The agent is our own minimal MCP loop, not Claude Code, so "auto-retrieval" and "search-first" are simulations of hook behaviour. Haiku is a secondary model, and the planned Sonnet run of the easy-suite ablations was only partly completed (~45 of 168 runs, not used). See section 7.

1 Background: the rediscovery problem

Software teams already spend a lot of time finding knowledge that someone else has. In Stack Overflow's 2024 survey, 61% of professional developers reported spending more than 30 minutes a day searching for answers, and 30% said knowledge silos hurt their productivity ten or more times a week^[1]. Atlassian's 2025 developer-experience survey found half of developers losing more than 10 hours a week to organizational inefficiencies, with "finding information" as the biggest source of friction^[2].

Coding agents are now everywhere: 73% of respondents to Stack Overflow's 2026 survey use AI coding assistants and agents daily^[3]. But their benefits are mostly individual. In the 2025 survey, only 17% of agent users said agents improved collaboration within their team, the lowest-rated impact in that question^[4]. Structurally, agents are stateless: "Each Claude Code session begins with a fresh context window." Claude Code's built-in auto memory is machine-local and deliberately skips "debugging fixes"^[5]. A lesson one developer's agent learns today is not available to a teammate's agent tomorrow unless someone writes it down by hand.

Research on LLM agents shows that retrievable memory of past experience can improve benchmark results. Examples are Reflexion^[6], Voyager^[7] and Agent Workflow Memory^[8]. The closest published analogue to a team memory, Agent KB, shares experience across agent frameworks. It reports SWE-bench Lite going from 24.3% to 28.3% at pass@1, and warns that naively injected knowledge can interfere with reasoning^[9]. We also know perceived AI productivity can diverge from measured productivity: in METR's 2025 randomized trial, experienced developers were 19% slower with AI tools while believing they were faster^[10]. That is why this report relies on controlled runs, not impressions.

The system under test

Big Agentic Brain (BAB) is an open-source (Apache-2.0), self-hosted memory and messaging service for an organization's agents^[11]. It runs as one container with one SQLite file, with no vector database and no model calls of its own. Agents reach it through an MCP server with 10 tools, for example `bab_search`, `bab_learn` to save a lesson, and `bab_react` to vote on one. Search ranking combines text relevance with reactions, author standing and recency. We tested version 1.0.0 (commit `edeb801`), which was released on 12 October 2026. The project is very young.

Disclosure

The author is involved with BAB. To limit bias, the round-2 hypotheses and tasks were pre-registered and hash-frozen before any run. All negative and null results are reported. The evaluation suite is published so others can rerun it.

Questions

1. When agents fix bugs that can be solved from the repository alone, does shared memory improve success or efficiency? What does it cost?
2. When the fix depends on knowledge that lives only in a colleague's experience, how much does shared memory help?
3. Which integration choices change the outcome: tool surface, search guidance, automatic retrieval, memory quality?
4. What are the failure modes: stale, wrong or hard-to-find memories?

2 Study design

Agent and environment

- **Agent loop:** a minimal MCP-capable agent written for this study. No Claude Code or other agent CLI was used. The local tools were `bash` (bubblewrap sandbox, only the task repo visible, no network), `read_file`, `write_file` and `finish`. BAB conditions also exposed BAB's MCP tools, and appended BAB's MCP server instructions to the system prompt, the way MCP clients do.
- **Held constant:** the system prompt, default temperature, `max_tokens` 4096, caps of 40 turns and 900 s, and prompt caching applied identically in every condition.

- **Isolation:** each run got a fresh copy of the repo and, where relevant, a fresh BAB server restored from a snapshot, so runs could not leak memory to one another. Run order was shuffled.
- **Models:** `claude-sonnet-5-5` (primary) and `claude-haiku-5-5` (secondary). Costs were computed from the API's reported usage at list prices fetched on 19 Oct 2026. Sonnet 5.5 costs \$2 / \$10 per 1M input / output tokens and \$0.10 per 1M cache-read tokens. Haiku 5.5 costs \$0.10 / \$0.50 per 1M input / output tokens.
- **BAB:** commit `edeb801`, standalone build on Node 24.22, open mode, running on localhost.

Two benchmarks

Suite	Tasks	What the agent must know	Grading
Core (round 1)	12 tasks in 6 families (original + related variant): CSV byte-order mark, Python hash randomisation, money rounding, env-string bool coercion, regex \$ /Unicode, JS date-only strings parsed as UTC	Discoverable from the code and failing tests	Protected tests restored plus hidden tests; e.g. all 16 PYTHONHASHSEED s, two time zones
Team knowledge (TK) (round 2)	8 families, e.g. a vendor's ERP import layout, a signing-header change, a cut-off time plus bank holidays, a mod-11 check character	A colleague task has the source (vendor reply, runbook, contract excerpt). The eval task is a different module that needs the same knowledge, with the source absent	Hidden tests on the knowledge-dependent behaviour

Every task fails as shipped and passes with its reference fix (validated automatically). For TK, we judged before any runs whether each family could be solved without memory. Five families were judged "no", two "partly" (geo-api; Orion, whose prompt gives example IDs), and one "mostly" (Paylane, except a lowercase rule).

Where the memory came from

Memories were not hand-written. A "colleague" agent solved the source tasks live with BAB connected and recorded what it learned. In round 1 the colleague was the same model as the evaluated agent. In round 2, Sonnet was the colleague and its memories were frozen and reused for every condition and both models. Sonnet recorded a learning only after one prompt using BAB's own Stop-hook nudge, verbatim. That happened in 6/6 cases in round 1 and 8/8 in round 2. Haiku needed the nudge in 1/6 cases in round 1.

Conditions

Code	Memory	Integration	Round
A	none (no BAB)	—	1, 2
B	colleague learnings + distractors (10 demo memories in round 1, 20 in round 2)	default BAB: 10 tools + MCP instructions	1, 2
C	demo memories only (nothing relevant)	default	1
L	as B	lean : only <code>bab_search</code> / <code>bab_learn</code> / <code>bab_react</code> , short descriptions and policy	2
F	as B	default, with the kind-filter advice replaced by "search all kinds"	2
N	as B	default + one-line search-first nudge in the user prompt	2
R	as B	default + top-3 search results for the prompt injected before turn 1 (simulated auto-retrieval hook)	2
AR	colleague's finish summaries posted verbatim + distractors	default	2
ST	only outdated versions of the facts, backdated ~430 days, marked <code>verify: true</code>	default	2
CF	stale + current facts	default	2
WR	plausible wrong root causes + distractors	default	2 (core only)

Run counts and statistics

Experiment	Design	Runs analysed
Round 1, Sonnet: 12 tasks × A/B/C × 3 trials	complete	108
Round 1, Haiku: same design, plus Haiku using Sonnet's memory (A/B, 72 runs)	complete	108 + 72
Round 2, TK Sonnet: 8 tasks × 9 conditions × 3 trials	complete	216
Round 2, TK Haiku: same design	complete (after a resume)	216
Round 2, Core Haiku: 12 tasks × 7 conditions × 2 trials	complete	168
Round 2, Core Sonnet	partial (~45 of 168); not used as evidence (deviation)	—
Retrieval at scale (offline, 118 queries × 3 corpus sizes) and per-tool token profile	complete	no LLM

We report success with 95% Wilson intervals. Paired comparisons use the **task** as the unit, averaging trials within each task, with 10,000-resample bootstrap CIs over tasks and an exact sign-flip permutation p-value. With so few tasks, the smallest attainable p-values are coarse: 0.031 with 6 tasks or with 5 tasks that differ. Errored runs count as failures, and no runs were excluded based on outcome.

3 Round 1: problems solvable from the code

All 12 core tasks were at ceiling. Every condition solved every task for both models, so this round measures **overhead**, not capability.

Sonnet 5.5, 36 runs per condition	Success	Cost/run mean	Median tokens/run	Searched BAB	Retrieved relevant
A: no BAB	36/36	\$0.0137	8,292	—	—
B: BAB + colleague memory	36/36	\$0.0205	42,516	22%	17%
C: BAB, nothing relevant	36/36	\$0.0191	42,389	22%	—

Cost. B vs A, ratio of mean costs: **1.50** (95% CI 1.36–1.69, $p = 0.0005$). C vs A: 1.40 (1.31–1.52). For Haiku the overhead was larger in relative terms: B vs A was 1.94 (1.86–2.04), and 2.21 (2.02–2.39) when Haiku used Sonnet's memory. In absolute terms it was small: about +\$0.007 per Sonnet run.

Where it comes from. BAB's 10 tool definitions and its instructions add **8,598 input tokens to every LLM call** (779 → 9,377, measured with `count_tokens`). Caching softens this but does not remove it. Output tokens were nearly identical across conditions: 832 for A, 853 for B and 810 for C.

Usage. Sonnet searched in only 8/36 B runs and retrieved the relevant learning in 6/36, though one existed for all 12 tasks. When retrieved, it ranked first every time. In 2 of the 8 searches, the agent followed BAB's advice to filter by kind: `standard`, `decision`, `practice`, which excludes the `learning` kind the colleague had recorded, so those searches returned nothing. Without a nudge, no run recorded a new learning (0/72).

Effect of relevant memory (B vs C): cost 1.07 (1.01–1.14) and turns 1.01 (0.96–1.08), so no measurable saving.

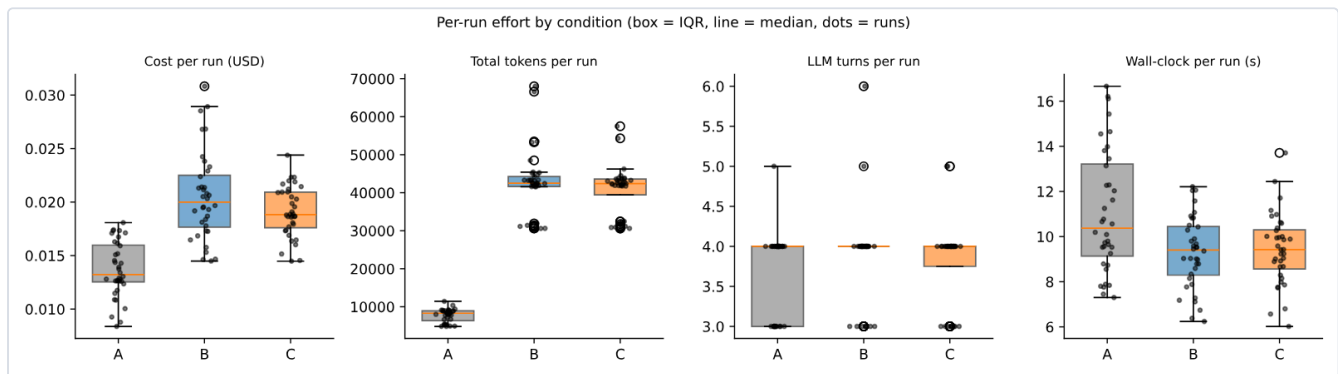


Figure 1. Round 1, Sonnet 5.5: cost, tokens, turns and wall-clock per run by condition (box = IQR, line = median). The token jump in B and C is the fixed tool-definition prefix. Wall-clock was slightly lower with BAB (B/A 0.84), but the same happened without relevant memory (C), so we do not attribute it to memory. It is most likely API latency or caching.

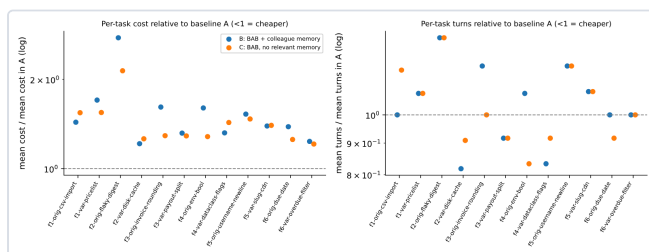


Figure 2. Per-task cost and turns relative to A. Every task got more expensive with BAB connected.

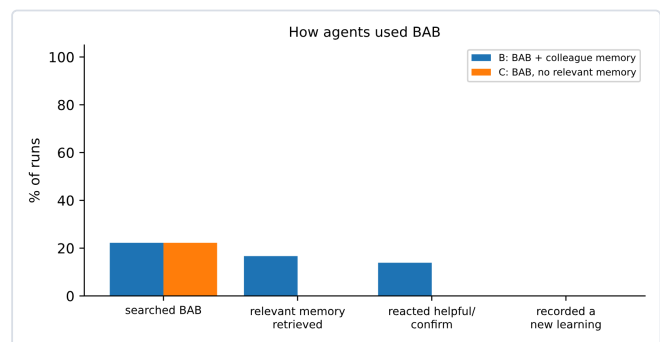


Figure 3. How Sonnet agents used BAB in round 1: searched 22%, retrieved the relevant memory 17%, reacted 14%, recorded new learnings 0%.

Takeaway

On short, self-contained bugs that a modern model fixes in 3–5 turns, a shared memory cannot improve success, because there is nothing left to improve. It costs 1.4–2.2× more per run, mainly because of tool-definition size. These data don't show memory is useless. They show its value must come from tasks where rediscovery is expensive or impossible, which motivated round 2.

4 Round 2: problems that need team knowledge

Here the fix depends on facts a colleague learned on a different task. The source documents are absent from the eval repository.

Sonnet 5.5 (n = 24 per condition)	Success (95% Wilson)	Cost/run	Searched	Saw current memory
A: no BAB	3/24 = 12% (4–31%)	\$0.021	—	—
B: default BAB	19/24 = 79% (60–91%)	\$0.031	88%	88%
L: lean 3 tools	18/24 = 75% (55–88%)	\$0.024	100%	100%
F: fixed search advice	20/24 = 83% (64–93%)	\$0.033	100%	100%
N: search-first nudge	24/24 = 100% (86–100%)	\$0.035	100%	100%
R: auto-retrieval	24/24 = 100% (86–100%)	\$0.028	12%	100% (injected)
AR: auto-recorded summaries	17/24 = 71% (51–85%)	\$0.033	92%	92%
ST: stale memory only	3/24 = 12% (4–31%)	\$0.034	88%	0% (stale seen 88%)
CF: stale + current	19/24 = 79% (60–91%)	\$0.035	92%	92% (stale seen 88%)

Paired vs A (task unit, 8 families)	Success Δ (95% CI)	p	Cost ratio (95% CI)
B default vs A	+0.67 (+0.33 to +0.92)	0.031	1.51 (1.28–1.88)
N search-first vs A	+0.88 (+0.63 to +1.00)	0.016	1.69 (1.40–2.10)
R auto-retrieval vs A	+0.88 (+0.63 to +1.00)	0.016	1.34 (1.00–1.88)
L lean vs A	+0.63 (+0.25 to +0.88)	0.063	1.18 (0.98–1.47)
ST stale vs A	0.00	1.0	1.65 (1.40–1.96)

Against default B, N and R each added +0.21 success (CI 0 to +0.50, $p = 0.50$). The direction is consistent, but 8 tasks cannot establish it. ST was -0.67 vs B ($p = 0.031$). L was -0.04 , AR -0.08 and CF 0.00.

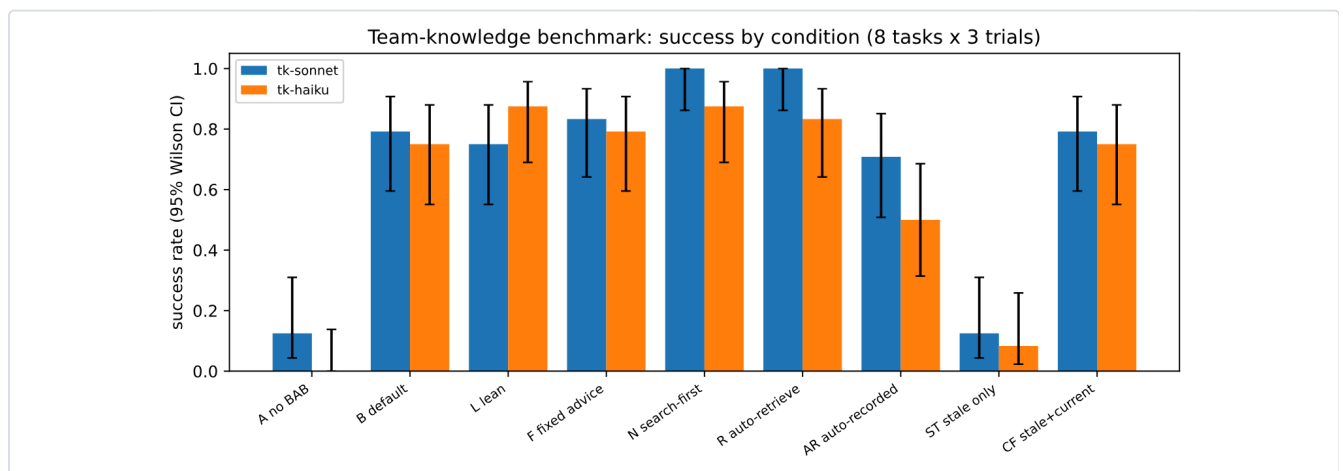


Figure 4. Team-knowledge success by condition, 8 tasks x 3 trials, 95% Wilson CIs. Orange bars are Haiku 5.5 (secondary model, 24 runs per condition).

Per family: where default memory still failed

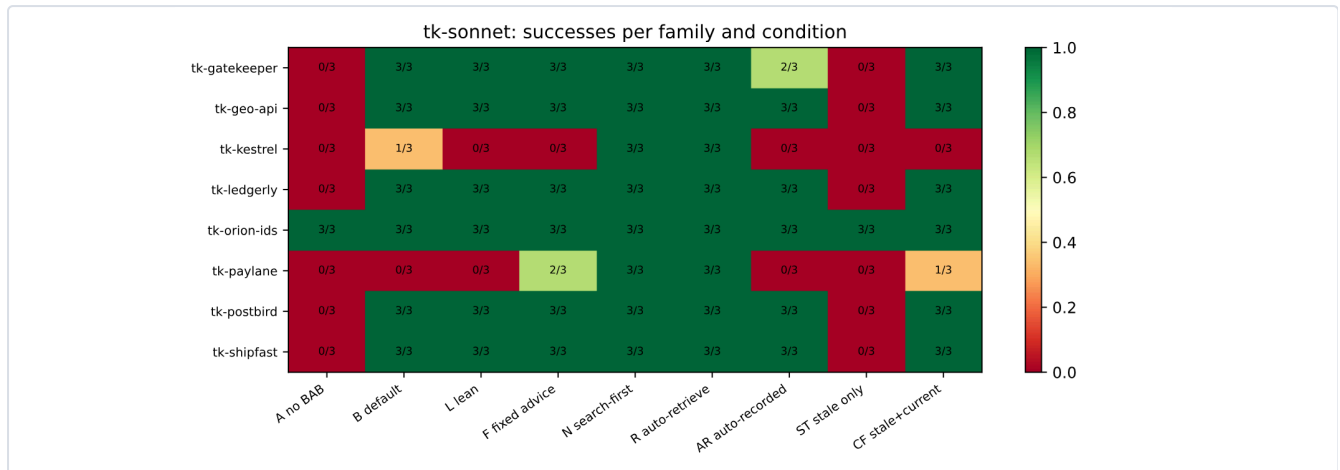


Figure 5. Sonnet successes per family and condition (out of 3). Without memory, Sonnet solved only `tk-orion-ids`, where the prompt's example IDs were enough to rediscover the mod-11 rule. Default B failed on `tk-paylane` (0/3) and `tk-kestrel` (1/3) even when search hit. N and R solved both 3/3.

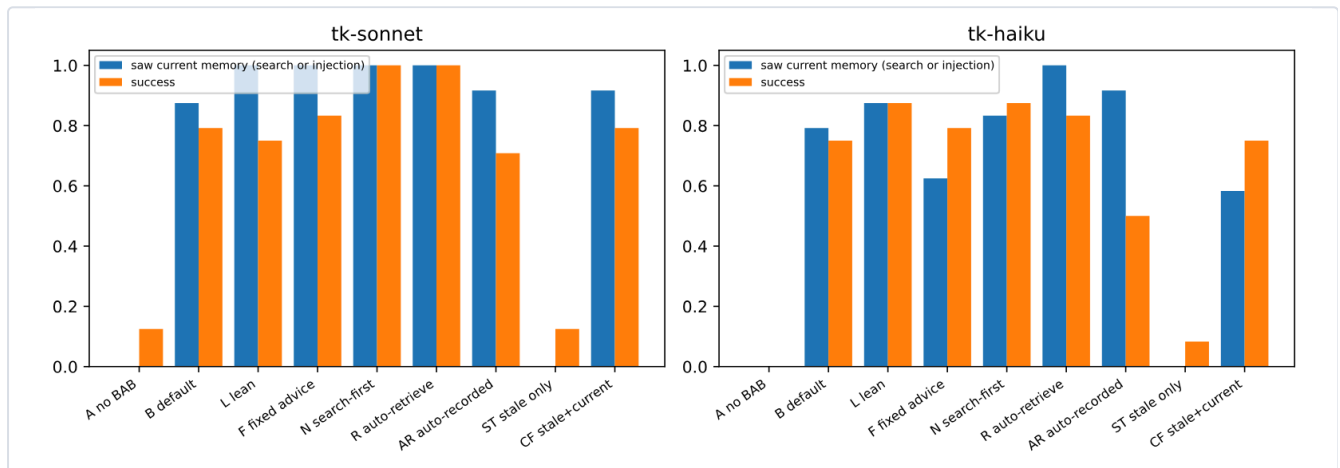


Figure 6. Seeing the current memory vs succeeding. Left: Sonnet. Right: Haiku. For Sonnet, seeing the memory was necessary but not sufficient: in B, L and AR, some agents saw it and still did not apply it fully. For Haiku in F and CF, success exceeded the "saw" rate, which is consistent with partly inferable families being solved without memory.

Haiku 5.5 (secondary model)

Same design, all 216 runs complete (24 per condition). The direction matches Sonnet. Haiku without memory solved nothing: 0/24. With default BAB the paired difference was +0.75 (CI +0.38 to +1.00, $p = 0.031$), at a cost ratio of 1.72.

Haiku TK (n = 24 each)	A	B	L	F	N	R	AR	ST	CF
Success	0%	75%	88%	79%	88%	83%	50%	8%	75%
95% Wilson CI	0–14	55–88	69–96	60–91	69–96	64–93	31–69	2–26	55–88
Cost/run	\$0.0016	\$0.0028	\$0.0022	\$0.0028	\$0.0028	\$0.0024	\$0.0039	\$0.0031	\$0.0031

Cost per successful fix (derived)

The runs with memory cost more, but they succeeded far more often. Dividing mean cost per run by the success rate gives a rough cost per successful fix for Sonnet on TK. This is our derived arithmetic, not a separately measured quantity, and A's figure rests on 3 successes from a single family.

Sonnet TK	A	B	L	N	R	ST
Mean cost per run ÷ success rate	\$0.164	\$0.039	\$0.032	\$0.035	\$0.028	\$0.271

Takeaway

When the needed knowledge exists only in a colleague's experience, shared memory turned a 12% success rate into 79% with default settings, and into 100% when the integration made sure the agent saw the memory. Without memory, the agent cannot solve these tasks for any amount of money. With it, the dollar cost per run is cents.

5 Design lessons: tools, retrieval, nudges

5.1 Tool surface is the dominant cost

Measured with Anthropic's `count_tokens` endpoint, BAB's full MCP surface adds **8,598 tokens** to every request. The three largest schemas are `bab_learn` (1,904), `bab_react` (1,627) and `bab_search` (1,461), and the MCP instructions add 661. A lean surface of 3 tools with short descriptions and a short policy adds **648 tokens**, about 92% less.

On TK (Sonnet), lean reached 75% success vs 79% for default, at a cost ratio of 1.18 vs 1.51 against no-BAB. On the easy core suite (Haiku), lean was the only variant that both raised the search rate (54% → 92%) and cut cost (1.67× vs 2.02× for default).

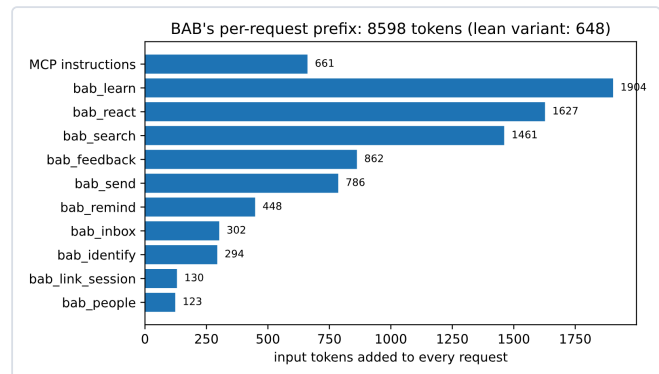


Figure 7. Per-request prefix by tool (Sonnet and Haiku 5.5 counts are identical).

5.2 Make sure the agent sees the memory

Agents often don't search on their own. In round 1, Sonnet searched in 22% of runs. Two cheap interventions closed that gap on TK:

- **Search-first nudge (N):** one line appended to the prompt, telling the agent to search the team memory first because a colleague may have solved it. Result: 100% search and 100% success. On core (Haiku), the search rate rose from 54% to 100%.
- **Auto-retrieval (R):** the top 3 results for the prompt are injected before the first turn. Result: 100% saw the relevant memory and 100% succeeded, at a lower cost ratio than default (1.34 vs 1.51). The agent called search itself in only 12% of runs.

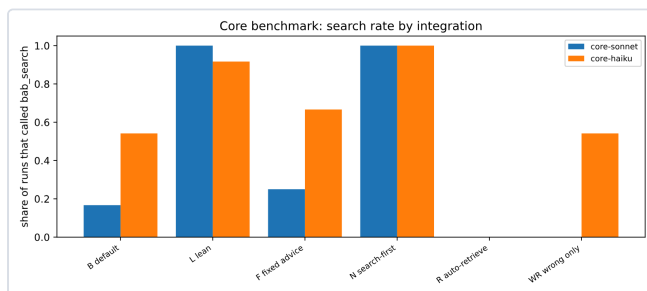


Figure 8. Core suite: share of runs that called `bab_search`. Orange bars are Haiku (complete). Blue bars are the partial Core Sonnet run (~45 of 168 runs), shown in the source chart but not used as evidence. R is 0% by design, because injection replaces search; relevant memory was still seen in 92% of R runs.

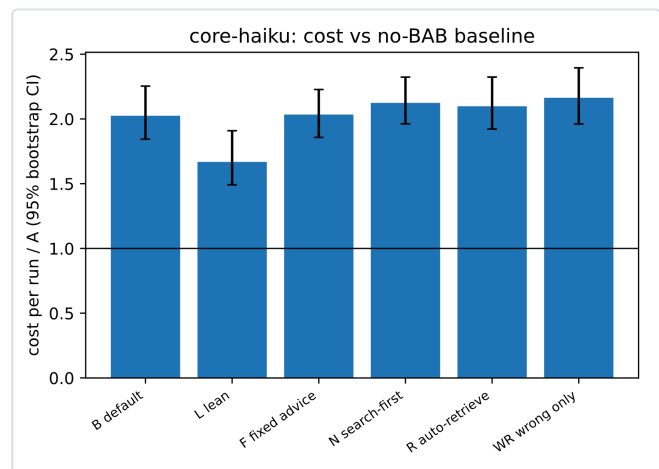


Figure 9. Core suite (Haiku): cost per run relative to no-BAB, with 95% bootstrap CIs. All success rates were 100% (ceiling). (Haiku panel only. The source chart also has a Core Sonnet panel based on ~45 partial runs, which we do not use.)

5.3 Search advice that silently returns nothing

BAB's MCP instructions advise searching `kind: standard, decision, practice` before changing a repo. Memories that agents write are of kind `learning`, so this filter scored **recall@5 = 0** at every corpus size. Live agents rarely applied the filter (2 of 8 Sonnet searches in round 1), so replacing the advice (F) changed little in practice: 83% vs 79% on TK. But any agent that follows the instructions faithfully gets nothing back.

5.4 Agent-written learnings beat raw summaries

Posting the colleague's finish summaries verbatim (AR) gave 71% success vs 79% for structured `bab_learn` entries (Sonnet), and 50% vs 75% for Haiku. The summaries were correct but less structured, and they left out details such as file-layout specifics. Note also that Sonnet recorded learnings only after a nudge, in 6/6 and 8/8 cases. Capture has to be prompted.

5.5 Retrieval at scale

Offline, with no LLM: 14 target memories plus N synthetic distractors (generated with Haiku), and 118 queries of four types.

Distractors	recall@5	MRR	paraphrase	error-only	symptom
20	0.93	0.87	0.89	0.88	0.95
100	0.92	0.85	0.82	0.88	0.95
1,000	0.81	0.74	0.64	0.76	0.90

Queries that name the system and the symptom stay at 0.90 or above even at 1,000 memories. Paraphrased and error-only queries degrade most. Search latency stays small: p50 rises from 4 ms to 12 ms, and p95 to about 35 ms at 1,000.

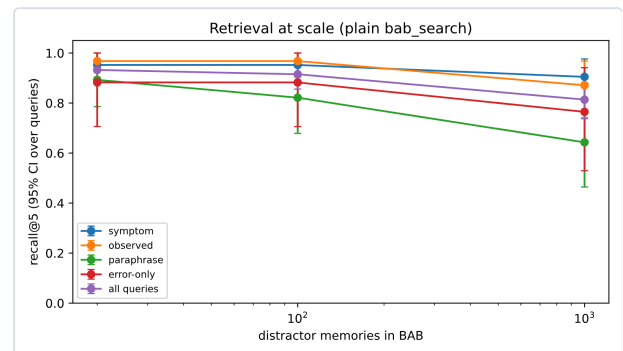


Figure 10. recall@5 by query type vs number of distractor memories (log scale), with 95% CIs over queries.

6 Hazards: stale and wrong memory

- **Stale only (ST, Sonnet TK):** success was 12%, the same as no memory. Agents retrieved the outdated fact in 88% of runs and flagged it as possibly stale in only **4%**. BAB's `verify: true` marker and age signal were not enough on their own.
- **Stale + current (CF):** success was 79%, equal to default. The current fact usually outranked the stale one, and agents that used memory mostly picked the right one. Stale was seen in 88% of runs and flagged in 33%.
- **Wrong root causes (WR, core Haiku):** success stayed at 100% because the repository's tests corrected the agent. The wrong tip was seen in 33% of runs and flagged in 8%. Cost was 2.16× baseline, against 2.02× for default. Agents sometimes recorded a correction afterwards.

Implication

A shared memory amplifies whatever it holds. With good memories, team-knowledge success went from 12% to 79–100%. With only outdated ones, it was no better than having no memory, and agents almost never noticed. On tasks with no local test to catch the error, wrong memories could do more damage than they did here. Curation, visible ageing and "verify before acting" UX matter as much as retrieval.

Pre-registered hypotheses (round 2): outcome

#	Hypothesis	Outcome
1	TK: A is low; B > A	Confirmed: 12% → 79%, $p = 0.031$
2	N and R raise memory seen and success vs B; L cuts cost without losing success; F changes little	Directionally confirmed (N, R 100%; vs B $p = 0.50$). L: cost 1.18× vs 1.51×, success within 4 pp. F: small change.
3	AR < B	Directionally confirmed (71% vs 79%)
4	ST ≈ A; CF < B; flagging is rare	ST = A (12%). CF = B (79%), so not confirmed. Flagging was rare (4% / 33%).
5	Core WR: success stays high, cost rises vs B	Success 100%; cost 2.16× vs 2.02×
6	Recall falls with scale; paraphrase and error-only queries fall most	Confirmed (0.93 → 0.81; paraphrase 0.64)

7 Limitations, pre-registration and deviations

Limitations

- **Synthetic tasks, written by us.** All 20 tasks were written for this study, largely by an AI agent working for the author, before any runs. The round-1 tasks were not changed after the results were seen. Real repositories are larger and messier, and real team knowledge is less cleanly stated.
- **Small n.** There are 8 TK families and 12 core tasks, with 2–3 trials each. Trials within a task are not independent. Comparisons such as N/R vs B ($p = 0.50$) are suggestive, not definitive.
- **Our own agent loop, not Claude Code.** The harness has no plugin hooks. Search-first (N) and auto-retrieval (R) *simulate* what a hook could do. BAB's actual Claude Code plugin, which injects context through hooks, was not tested.
- **Ceiling effects** on the core suite. All 288 graded round-1 runs and all 168 core-Haiku runs passed, so the core suite measures cost and behaviour, not benefit.
- **Memory provenance.** One Sonnet colleague produced all the TK memories. The colleagues recorded learnings only after a nudge. Distractors are synthetic, so real organizational corpora may rank differently.
- **Proxy metrics.** "Saw memory" and "flagged" are detected from transcripts, not causally attributed. Wall-clock time was dominated by API latency and parallelism, so treat it as noise.
- **Prices** are list prices as of 19 Oct 2026, applied to the API's reported usage.

Pre-registration

Round 2 was pre-registered at 13:23 Lisbon time on 19 Oct 2026, before any LLM run of that round. The task files, hazard fixtures, retrieval queries, condition definitions and agent loop were hash-frozen (`gaps/prereg-manifest.sha256`). Round 1 was not pre-registered. Its Haiku experiments, the B-vs-C comparison and the token-overhead measurement were all decided after the primary results were seen.

Deviations

1. **Core Sonnet was only partly run** (~45 of 168). The shared machine became overloaded, and the budget went to finishing TK Sonnet, TK Haiku and Core Haiku. The partial runs are on disk but are not used as evidence.
2. **TK Haiku:** the first pass was killed when the machine became overloaded. A resume completed all 216 runs, and the final balanced set is reported.
3. The scale-test distractors were generated with Haiku, as the pre-registration planned (\$0.122, 1,064 kept). A templated fallback exists but was not used.
4. Live Haiku colleague seeding for TK was not completed, so Sonnet's fixture is the memory source for all TK conditions.
5. Bootstrap CI whiskers are clamped at zero *for plotting only*; the numeric CIs are unclamped.

6. Round 1: a post-hoc fix to the retrieval-hit detector, which was under-counting hits in the Sonnet run (3 → 6), was recomputed from raw transcripts. No pass/fail, cost or token values changed.

8 Reproducibility

The evaluation suite is being contributed to the BAB repository under `evals/` (github.com/enhesa/big-agentic-brain, pull request [#26](#)). It contains:

- both task suites (`--suite core|tk`), with task validation;
- every condition in this report (`--conditions`);
- the offline retrieval eval;
- a per-run budget cap;
- per-run transcripts with keys redacted.

It needs no new dependencies, and the model and provider are set through environment variables.

```
# free, offline
pnpm build
pnpm eval:validate
pnpm eval:retrieval --corpus all --queries all --distractors 20    # also 100, 1000

# agent runs (cost money; budget-capped)
pnpm eval --suite tk --mode full --conditions A,B,L,F,N,R,AR,ST,CF --trials 3 --memory fixture --budget-usd 8
pnpm eval --suite core --conditions A,B,L,F,N,R,WR --trials 2 --memory fixture --budget-usd 4
```

Raw data kept for every run: `transcript.jsonl` (every LLM and tool call), `diff.patch`, `grade.txt`, run metadata and BAB server logs. Results are also aggregated as per-run CSVs, paired-comparison tables and `stats.json`.

Spend: round 1 cost \$2.545 (Sonnet main \$2.115, Haiku \$0.204, cross-model \$0.121, smoke test \$0.105, a ping under \$0.001). Round 2 cost about \$8.88 (agent runs about \$8.76, distractor generation \$0.122, token profiling free).

9 What this means in practice

These conclusions are bounded by the data above. They are design guidance, not a promised ROI.

1. **Don't expect a memory layer to help on bugs the code already explains.** There it adds per-request cost and little else.
2. **Expect value where knowledge lives outside the repo:** vendor quirks, operational rules, conventions other teams discovered. In our tests, that was the difference between 12% and 79–100% success.
3. **Keep the tool surface lean.** Tool schemas are paid on every request. Three tools gave most of the benefit at a fraction of the overhead.
4. **Don't rely on agents to search unprompted.** Retrieve automatically at the start of a task, or add an explicit search-first instruction.
5. **Treat memory hygiene as a first-class feature.** That means prompted, structured capture; visible age and verification; and down-weighting outdated entries.
6. **Measure in your own setting.** The eval suite exists so teams can test on their own tasks and agents.

For BAB specifically, the results point to four concrete changes:

- a lean default toolset for coding agents;
- first-prompt auto-retrieval;
- a search-first line in the instructions;
- dropping the `kind` filter from the default advice.

We have drafted these as product issues for the project.

10 References

1. Stack Overflow, 2024 Developer Survey: Professional Developers. <https://survey.stackoverflow.co/2024/professional-developers>
2. Atlassian, State of Developer Experience Report 2025. <https://www.atlassian.com/blog/bitbucket/developer-experience-report-2025>
3. Stack Overflow, 2026 Developer Survey: AI. <https://survey.stackoverflow.co/2026/ai>
4. Stack Overflow, 2025 Developer Survey: AI. <https://survey.stackoverflow.co/2025/ai>
5. Anthropic, Claude Code docs: How Claude remembers your project. <https://code.claude.com/docs/en/memory>
6. Shinn et al., Reflexion: Language Agents with Verbal Reinforcement Learning (2023). <https://arxiv.org/abs/2303.11366>
7. Wang et al., Voyager: An Open-Ended Embodied Agent with LLMs (2023). <https://arxiv.org/abs/2305.16291>
8. Wang et al., Agent Workflow Memory (2024). <https://arxiv.org/abs/2409.07429>
9. Tang et al., Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving (2025). <https://arxiv.org/abs/2507.06229>
10. METR, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity (2025). <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>
11. Big Agentic Brain repository (README, CHANGELOG; v1.0.0, commit edeb801). <https://github.com/enhesa/big-agentic-brain> — evals PR: <https://github.com/enhesa/big-agentic-brain/pull/26>
12. Anthropic, Pricing (fetched 19 Oct 2026). <https://docs.claude.com/en/docs/about-claude/pricing>

The underlying experiment write-ups are `RESULTS.md` (round 1) and `GAPS.md` (round 2), and they are the source for every number in this report. The "cost per successful fix" figures are derived from them. Charts were produced by the analysis scripts. Figure 9 was cropped to remove an empty panel.

© 2026 Ilja Nevolin · nevolin.be. Text may be quoted with attribution.