RESEARCH REPORT · OCTOBER 2026 · UPDATED 21 OCT

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

Measured results from 1,104 graded agent runs and a 378-session distributed-team experiment, with and without Big Agentic Brain (BAB), an open-source shared memory for agents.

12% → 79%

Success on 8 tasks that need team knowledge: no memory vs default BAB (Sonnet, n = 24 runs each, p = 0.031). Opus: 12% → 88%.

58% → 100%

Same-agent fix rate on a repeat problem with the agent's own local notes (Sonnet). Cost per fix fell from $0.0692 to $0.0427.

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).

13–67% → 96–100%

Fix rate when a teammate's agent on another machine hits the same wall: local-only memory vs shared BAB (Sonnet, Haiku, Opus).

$0.0427 → $0.0363

Team cost per verified fix, no memory vs shared BAB, first solvers included (Sonnet). Haiku: $0.00672 → $0.00333.

8,598 → 648

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

Ilja Nevolin · nevolin.be Data collected 19 and 21 Oct 2026 · BAB commit edeb801 Disclosure: the author is involved with BAB

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, Haiku 5.5 and Opus 5.5 as agents. A third round, added on 21 October, compares no memory, each agent's own local memory, and a shared team memory for agents that work on separate machines.

1,104 + 378	**20 + 3**	**≈ $46.77**	**3**
graded agent runs (19 Oct) + distributed-team sessions (21 Oct)	bug-fix tasks (12 core + 8 team-knowledge) + 3 two-repo use cases	total API spend: ≈ $29.36 rounds 1–2 (incl. $17.93 Opus) + $17.41 round 3	models: Sonnet 5.5 (primary), Haiku 5.5, Opus 5.5

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. 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) and with default BAB in 79% (19/24); the paired difference was +0.67 (CI +0.33 to +0.92, p = 0.031). A larger model did not make memory unnecessary: Opus 5.5 went from 12% (3/24) to 88% (21/24). Haiku went from 0% to 75%.
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 Sonnet 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.
5. **Retrieval degrades with scale and with harder queries.** Recall@5 fell from 0.93 at 20 distractor memories to 0.81 at 1,000 (paraphrases 0.64). That 0.93 is the easy case: with vendor-overlapping distractors and blind-written queries, recall@1 was 0.56, and 0.13 for the blind queries alone. BAB's own suggested `kind` filter returned nothing at any scale.
6. **Local memory helps the same agent; only shared memory reaches a teammate.** In the distributed-team round, an agent's own notes took Sonnet's repeat-problem fix rate from 58% to 100% and Haiku's from 21% to 96% (median time to a verified fix 900 s → 17.5 s). When a *different* agent hit the problem, local-only memory fixed 67% (Sonnet), 13% (Haiku) and 67% (Opus); the shared BAB fixed 100%, 96% and 100%.
7. **The saving is per fix, not per session.** Memory sessions use more tokens, so mean cost per session stayed flat or rose: the pre-registered primary test failed for Sonnet on H1 (ratio 1.058, p = 0.32). Team cost per verified fix, first solvers included, was lowest with shared memory for Sonnet ($0.0363 vs $0.0427 without memory) and Haiku ($0.00333 vs $0.00672). For Opus (2 teams, exploratory), local-only was cheapest ($0.0908 vs $0.0950 shared), because shared lookups add overhead when the agent already knows the answer.

> **Read these results with their limits in mind**
> All tasks are synthetic and were written by us. Team-knowledge tasks were built so the needed fact lives only in a colleague's memory, so they show that the channel works, not how often that situation arises. Samples are small: 8 team-knowledge families with 3 trials each, and in round 3 only 4 Sonnet teams and 2 Opus teams, with one use case (geo-api) driving much of the gap. The agent is our own minimal MCP loop, not Claude Code. See sections 5, 6.12 and 9.

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 and round-3 hypotheses and tasks were pre-registered and hash-frozen before any of their runs. 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?
5. When agents work on separate machines, how much does an agent's own local memory do, and what does a shared layer add on top? (Round 3, section 6.)

2 Study design (rounds 1 and 2)

Round 3 has its own design, described in section 6.

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), `claude-haiku-5-5` (secondary) and `claude-opus-5-5` (team-knowledge suite and round 3). 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. Opus 5.5 costs $4 / $20 per 1M input / output tokens, $0.20 per 1M cache-read tokens and $5 per 1M cache-write 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 all three 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 `bab_search` / `bab_learn` / `bab_react`, 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 `verify: true`	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, TK Opus: same design, added after the Sonnet and Haiku TK runs	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), hard retrieval set, per-tool token profile	complete	no LLM
Round 3, distributed teams (section 6)	Sonnet 4 teams, Haiku 8, Opus 2	378 sessions

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.

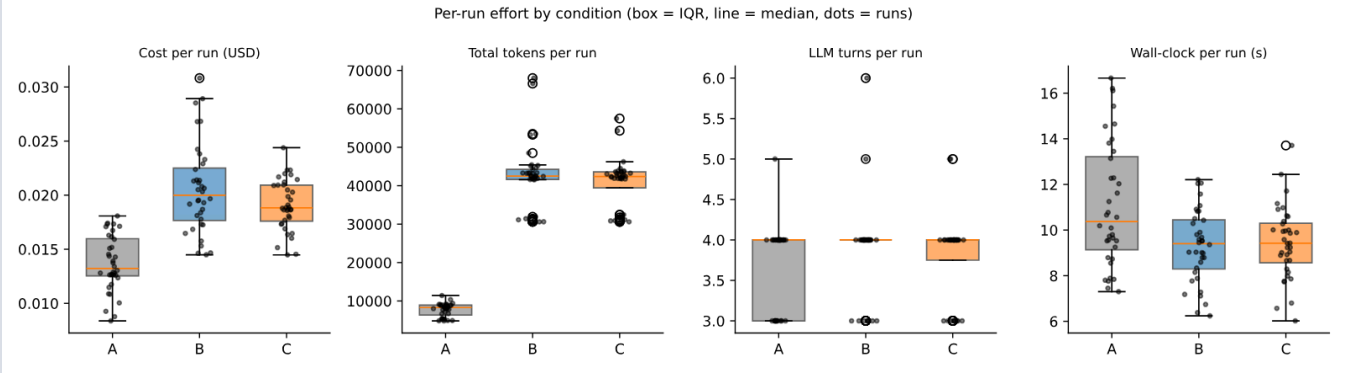

Figure 1. Round 1, Sonnet 5.5: cost, tokens, turns and wall-clock per run by condition (box = IQR, line = median). The token jump in B and C is the fixed tool-definition prefix. Wall-clock was slightly lower with BAB (B/A 0.84), but the same happened without relevant memory (C), so we do not attribute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

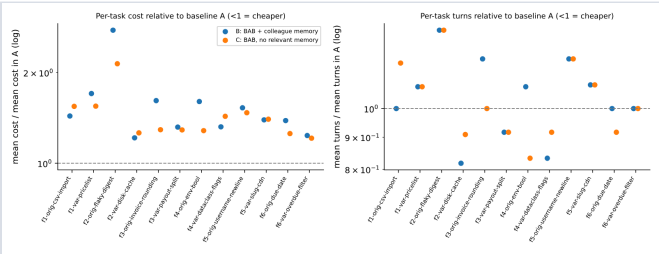

Figure 2. Per-task cost and turns relative to A. Every task got more expensive with BAB connected.

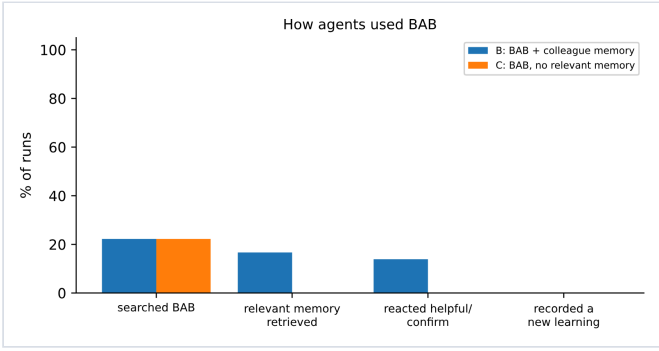

Figure 3. How Sonnet agents used BAB in round 1: searched 22%, retrieved the relevant memory 17%, reacted 14%, recorded new learnings 0%.

> **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.

4.1 Sonnet 5.5 (primary model)

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.

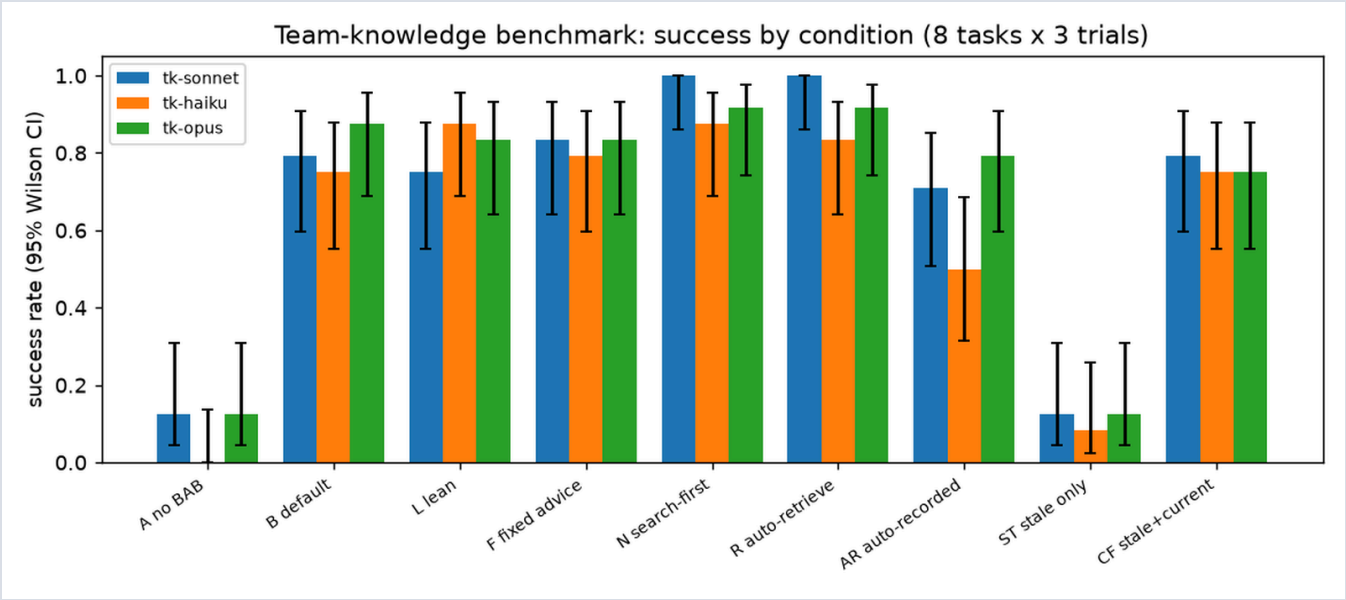

Figure 4. Team-knowledge success by condition, 8 tasks × 3 trials, 95% Wilson CIs. Blue: Sonnet 5.5. Orange: Haiku 5.5. Green: Opus 5.5 (24 runs per condition each).

4.2 Per family: where default memory still failed

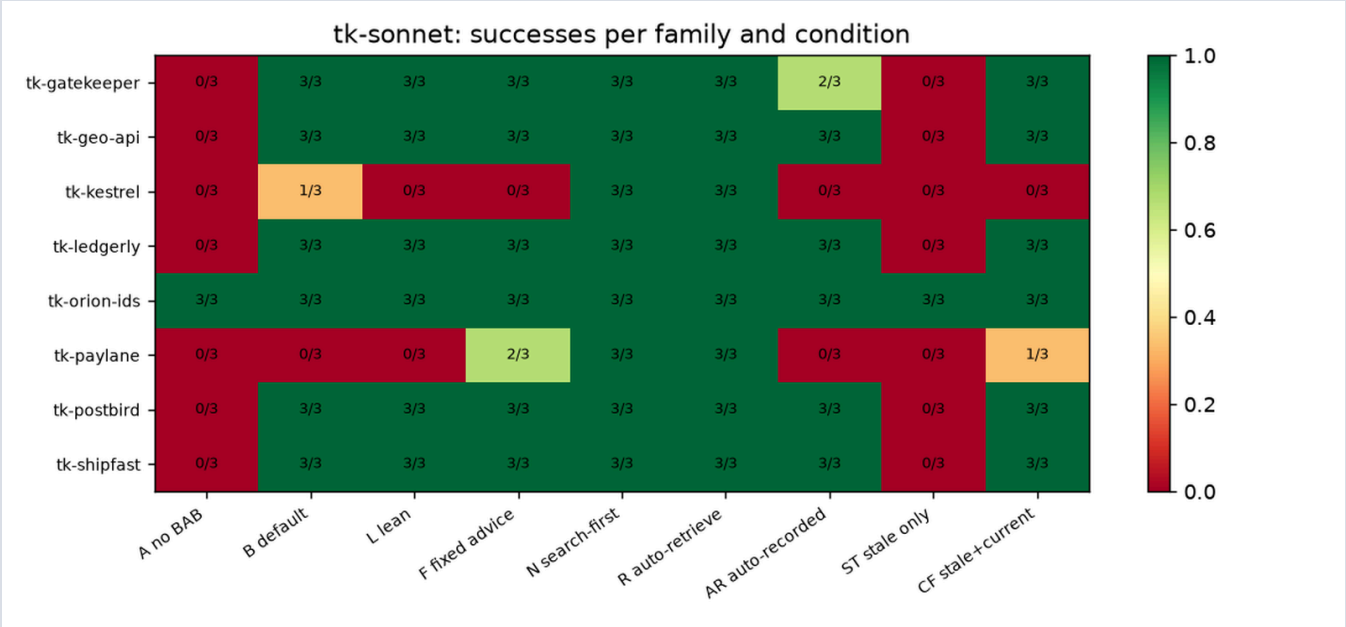

Figure 5. Sonnet successes per family and condition (out of 3). Without memory, Sonnet solved only `tk-orion-ids`, where the prompt's example IDs were enough to rediscover the mod-11 rule. Default B failed on `tk-paylane` (0/3) and `tk-kestrel` (1/3) even when search hit. N and R solved both 3/3.

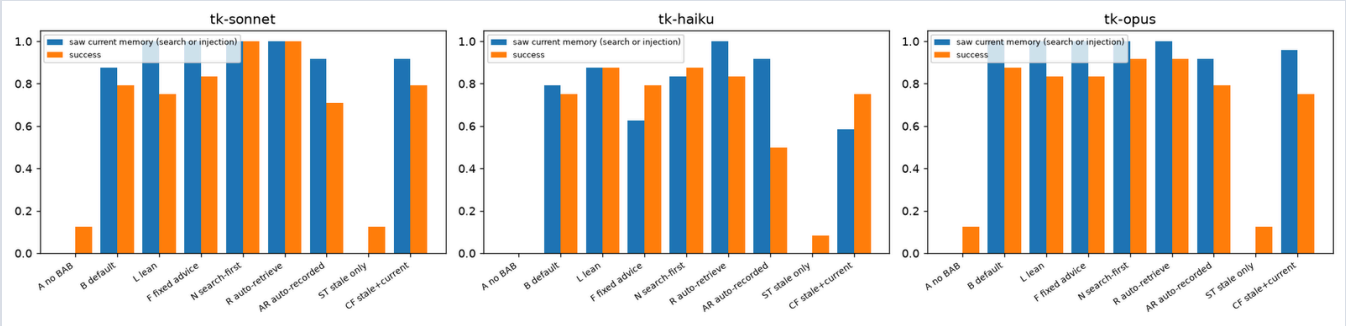

Figure 6. Seeing the current memory vs succeeding. Left: Sonnet. Middle: Haiku. Right: Opus. For Sonnet, seeing the memory was necessary but not sufficient: in B, L and AR, some agents saw it and still did not apply it fully. For Haiku in F and CF, success exceeded the "saw" rate, which is consistent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

4.3 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

4.4 Opus 5.5 (team-knowledge suite)

Same design and the same frozen memory fixture, all 216 runs complete (24 per condition), for $17.93 under a $20 cap. **A larger model did not make memory unnecessary.** Without memory Opus succeeded in 3/24 runs (12%), the same rate as Sonnet. With default BAB it succeeded in 21/24 (88%), the best default result of the three models. The paired difference was +0.75 (CI +0.38 to +1.00, p = 0.031), at a cost ratio of 1.54 (1.36–1.80). N and R reached 92%, and with only stale memory (ST) success fell back to 12%, just like Sonnet. Lean (L) was again cheaper than default with similar success. Opus costs more per run than Sonnet, roughly 2.7× in condition A.

Opus TK (n = 24 each)	A	B	L	F	N	R	AR	ST	CF
Success	12%	88%	83%	83%	92%	92%	79%	12%	75%
95% Wilson CI	4–31	69–96	64–93	64–93	74–98	74–98	60–91	4–31	55–88
Cost/run	$0.0553	$0.0853	$0.0705	$0.0856	$0.0864	$0.0831	$0.0965	$0.0942	$0.0905
Searched	—	100%	100%	100%	100%	17%	92%	96%	96%
Saw current memory	—	100%	100%	100%	100%	100%	92%	0%	96%

Compared on the same fixture: default B was 88% for Opus, 79% for Sonnet and 75% for Haiku. Auto-recorded summaries (AR) held up best on Opus (79%, against 71% for Sonnet and 50% for Haiku).

4.5 Deep dive: results grouped by memory type

Grouping the nine conditions by what the agent had in memory makes the cross-model pattern easy to see. "Shared, good" pools B, L, F, N and R (colleague learnings in BAB, integration varies, n = 120 per model). "Shared, poor" pools AR, ST and CF (raw summaries, stale facts, or stale plus current; n = 72). No condition in rounds 1–2 gave an agent its own notes from its own earlier sessions; round 3 adds that arm.

TK suite	A: no memory	Shared, good content	Shared, poor content
Sonnet 5.5	12.5% · $0.0205/run	87.5% · $0.0301/run	54.2% · $0.0337/run
Haiku 5.5	0% · $0.0016/run	82.5% · $0.0026/run	44.4% · $0.0034/run
Opus 5.5	12.5% · $0.0553/run	87.5% · $0.0822/run	55.6% · $0.0937/run

On the core suite (easy, repo-revealed bugs) every group reached 100% for Haiku, and shared memory only added cost ($0.00115 vs $0.00229 per run).

Where the 12% no-memory successes came from. All of Sonnet's 3/24 and Opus's 3/24 condition-A successes were on `tk-orion-ids`; Haiku's A was 0/24. The agents inferred the check character from the two example IDs in the prompt. Sonnet wrote: *"I inferred that rule from your two examples rather than from a spec … Please confirm the mod-11 rule against the real Orion spec."* Opus wrote: *"It is the ISBN-10 style mod-11 rule … It was the only weighting I tried that fits both examples."* **There was no leakage.** `git log` (run in 24/24 Sonnet, 24/24 Haiku and 21/24 Opus A runs) shows a single `init` commit, the sandbox binds only `/usr`, `/etc` and the workspace, hidden tests and oracle are copied in only at grading time in a separate directory, and no suspicious commands were logged. Haiku's three Orion runs failed: one dropped the checksum entirely, two found the right rule but stripped dashes too liberally.

What A failures looked like. Every A run ended by calling `finish`; none hit the turn limit. A minority gave up without a code change (Sonnet 5/21, Opus 6/21, Haiku 0/24). The rest made plausible but wrong or partial fixes, such as adding an `X-Shipfast-Timestamp` header. Most summaries said openly that they were guessing (Sonnet 18/21, Haiku 21/24, Opus 20/21).

4.6 Cost per successful fix (derived)

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

$ per success, TK	A	B	L	N	R	AR	ST	CF
Sonnet 5.5	0.164	0.039	0.032	0.035	**0.028**	0.046	0.271	0.044
Haiku 5.5	∞ (0/24)	0.0038	**0.0026**	0.0033	0.0028	0.0078	0.037	0.0042
Opus 5.5	0.442	0.097	**0.085**	0.094	0.091	0.122	0.754	0.121

> **Takeaway**
> When the needed knowledge exists only in a colleague's experience, shared memory turned a 12% success rate into 79% (Sonnet) and 88% (Opus) with default settings, and into 92–100% when the integration made sure the agent saw the memory. Model size did not substitute for the missing fact. With memory, the dollar cost per run is cents.

5 Review caveats: how to read rounds 1 and 2

Four caveats bound how far the round-2 numbers generalise. They came out of reviewing the first version of this study, and we added the hard retrieval test in response.

1. **The TK tasks were built so the fact lives only in colleague memory.** The missing fact (a vendor spec, a team convention) is absent from the eval repo by construction. So the 12% → 79–88% gap mostly shows that the memory channel works when the knowledge is there, not how often that situation arises in real work. Round 3 relaxes this: its probe repos hold a recorded-traffic file from which the rule *can* be inferred, so no-memory agents have a genuine path to a fix.
2. **Differences between BAB variants are within noise.** With 8 task families, N and R vs default B was +0.21 (CI 0 to +0.50, p = 0.50). The smallest attainable exact sign-flip p with 8 tasks is 0.016. Treat the ranking of B, L, F, N and R as suggestive; only "memory vs no memory" and "stale vs good memory" are clearly resolved.
3. **Recall 0.93 is the easy case.** The soft distractor set never mentions the invented TK vendors, so full-text search gets a unique-token hit and several queries repeat a title or error string. A harder set adds 40 distractors that share vendor and subsystem words but state different facts, plus 16 blind-written queries:

Retrieval setting	Queries	recall@1	recall@5 (95% CI)	MRR (95% CI)
TK soft (20 distractors, v2 queries); previous upper bound	50	0.94	1.00 (1.00–1.00)	0.96 (0.92–1.00)
TK hard (20 + 40 hard distractors, v2 + blind queries)	**66**	**0.56**	**0.85 (0.76–0.92)**	**0.68 (0.59–0.78)**
of which blind-written queries	**16**	**0.13**	**0.44 (0.19–0.69)**	**0.26 (0.11–0.42)**

Blind queries have a maximum stopword-free token coverage of 0.25 against their targets. The 0.93 / 0.81 recall@5 figures in section 7.5 use the general 118-query set with soft distractors.

4. **The no-memory successes are explained, not leaked.** All came from the Orion IDs family, where agents inferred a mod-11 check from the prompt's examples (section 4.5).

6 Round 3: distributed teams, local vs shared memory

Rounds 1 and 2 had no local-only memory arm and no multi-session sequences: every run started cold against one memory on one server. So they could not separate "an agent's own notes help" from "a teammate's notes help". Round 3 was designed to answer exactly that, on 21 October 2026 (11:54–14:12 Lisbon time).

6.1 Question

Several agents work as if on separate machines: separate sandboxes and workspaces, no shared filesystem and no shared local notes. They hit the same wall on related tasks. Each agent keeps its own persistent memory across its own sessions; only the shared BAB layer can carry agent 1's lesson to agent 2. We measure the **local effect** (the same agent repeats a related problem), the **shared effect** (a different agent hits the problem after a teammate solved it), and the **team view** (do the savings survive once the first solver's extra recording cost is counted?).

6.2 Use cases

Each use case has a first-exposure task (S1), where the knowledge is written down in that repo, and a related probe task in *another* repo where it is not. The probe *can* be solved without memory, but slowly: the probe repo holds a recorded-traffic file (`ops/*.jsonl`, 64–70 records, 12–20 KB) from which the rules can be inferred, though they are never stated as prose. The probes are the frozen, validated TK eval tasks plus that ops file and a one-line README pointer; every accepted or rejected record agrees with the oracle (0 mismatches), and all 19 TK tasks fail as shipped and pass with the reference fix.

Use case	Kind	S1: knowledge in repo	Probe symptom (other repo)	What memory carries
UC1 Paylane	Undocumented vendor API quirk	Fix refunds per `docs/paylane-api-v3-changes.md`	Charges off by orders of magnitude / "amount must be a string"	Amount is a minor-unit **string**, ISO 4217 exponents (JPY ×1, KWD ×1000), half-up, **lowercase** currency
UC2 Gatekeeper	Internal platform auth convention	Billing API claims per `docs/gatekeeper-migration.md`	Reports service: 401 "invalid token claims"	iss `https://gatekeeper.internal`, aud `svc:<name>` as string or list, `scp` space-separated, **120 s skew**
UC3 geo-api	Incident recurrence on an internal service	Routing client per `docs/geo-api-runbook.md`	Geocode client: key suspended after an incident	`Retry-After-Ms` in ms (default 1000), ≤ 3 retries, **503 = quota, never retry**

6.3 Conditions and sequence

	Persistent memory	Retrieval prompt	Recording prompt at end of S1
N0	none; nothing survives a session	none	none
LO	own `/memories` dir via a memory tool (view / create / str_replace / insert / delete / rename), carried across that agent's own sessions only; read-only in bash	"view /memories at the start of every task" + "view your memory directory first"	local mirror of BAB's Stop-hook nudge, only if the agent wrote nothing
SH	LO **plus** one BAB per team (lean toolset: search / learn / react; 20 standard distractors)	same protocol + "view /memories **and** search the team memory first"	BAB's own Stop-hook nudge (verbatim), only if no `bab_learn`

SH is local *plus* BAB because real teams have both, and the fair question is what the shared layer adds. Both memory arms get a matched check-first line and a matched one-shot recording nudge, so retrieval and write opportunities are symmetric. The local memory is a plain tool, so it works with any provider.

Sequence (one team × condition = one cell). A team has 3 agents, one per use case. (1) **S1:** each agent solves its first-exposure task, in seeded random order; in SH all three write to the team BAB, in LO each writes only to its own `/memories`. (2) **Freeze:** every memory dir and the team BAB are snapshotted. (3) **Probes**, each from the frozen snapshot in a fresh workspace and BAB copy: a **same** probe gives the related task to the agent that solved S1; a **cross** probe gives it to a different agent, rotating (i + 1) mod 3, whose own memory is non-empty but irrelevant. That is 9 sessions per cell (3 S1 + 3 same + 3 cross).

Prior work is identical across conditions. Cells ran in seeded random order with two in parallel, so box load and API drift hit all conditions alike.

6.4 Pre-registered hypotheses

The plan, hypotheses, metrics, analysis code, run matrix and stop rules were fixed before any paid run (freeze recorded 21 Oct 11:54:28 Lisbon time, manifest hashed). Nothing in the hypotheses or analysis changed afterwards; two amendments changed only the harness and the run matrix (section 6.12).

- **H1 (local, same agent):** on same probes, LO costs less than N0 (mean cost per session, expected ratio ≤ 0.70), and time to verified delivery is shorter.
- **H2 (shared, cross agent):** on cross probes, SH costs less than LO (expected ≤ 0.70), and is faster.
- **H2b (manipulation check):** on cross probes, LO ≈ N0 (expected ratio 0.9–1.15): local notes do not transfer.
- **H3 (team total):** team cost per delivered fix over all 9 sessions, including S1 recording overhead, is ordered SH < LO < N0.
- **H4 (no penalty):** on same probes, SH ≈ LO (ratio within 0.85–1.15).

6.5 Metrics and statistics

- **Headline metrics:** cost at list price (input, output, cache read and cache write) per session and **per delivered fix** (total ÷ fixes); **time to a verified fix** (`ttd_capped`): wall-clock seconds to a graded-passing `finish`, with non-delivery set to the 900 s cap, so a fast wrong fix is never "fast". Turns are reported as a load-free proxy.
- **Primary test** (per model; Sonnet confirmatory): two-sided **stratified permutation test** on mean log cost per probe session, labels permuted within use case. **Holm** correction across H1 and H2.
- **Effect sizes:** ratio of means and ratio of cost per delivered fix, each with a 95% **stratified bootstrap** CI (teams resampled within use case, 10,000 reps).
- **H3:** team is the unit, permutation over teams. H2b, H3, H4, all time and turn contrasts, and Haiku and Opus are uncorrected and exploratory.
- **Exclusions:** none based on outcome. Errored runs count as failures, with cost counted; smoke runs are excluded.

6.6 Runs and spend

Block	Sessions	Spend	Cap
Smoke, Haiku (v1, superseded by Amendment 1)	27	$0.0837	$0.20
Smoke, Haiku v2	27	$0.0826	
Smoke, Sonnet	18	$0.6201	$1.00
Sonnet main, lost in a box restart at 12:44 (≥ 137 sessions logged; tracker $5.275)	—	$7.50 (accounting bound)	$12.50
Sonnet rerun (4 teams × 3 conditions)	108	$3.7671	
Haiku (8 teams × 3 conditions)	216	$0.6488	$1.00
Opus, exploratory (2 teams × 3 conditions)	54	$4.7040	$8.50
Total	**378 analysed**	**$17.41 ($15.18 using the logged $5.275)**	**$30 ceiling never exceeded**

6.7 Headline: cost per fix and time to a verified fix

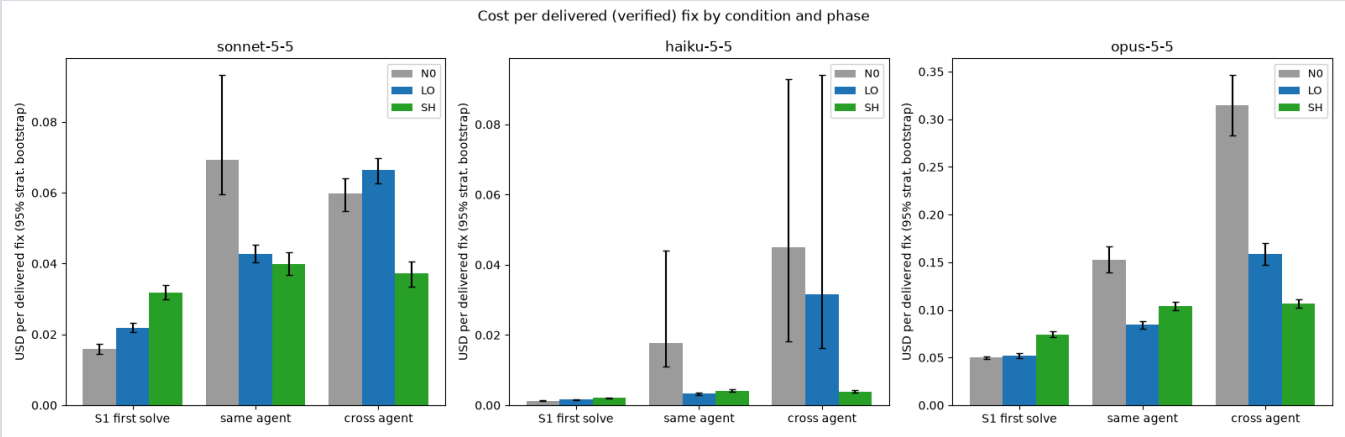

Figure 11. Cost per delivered (verified) fix by condition and phase, with 95% stratified bootstrap CIs. With shared memory (green), a teammate's cross-agent fix costs about as much as a same-agent fix; with local-only memory (blue) it stays as expensive as having no memory.

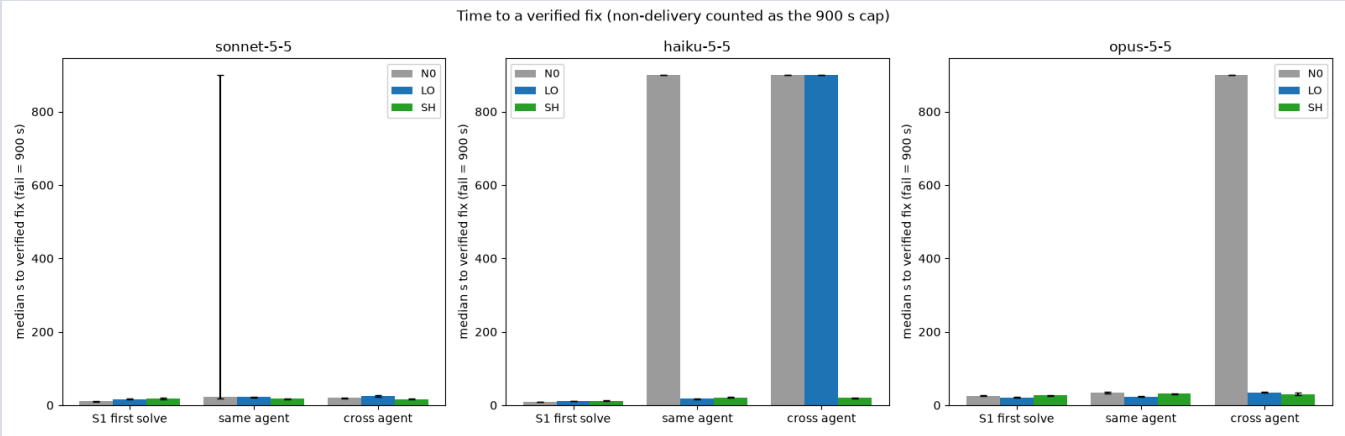

Figure 12. Median time to a verified fix, with non-delivery counted as the 900 s cap. Bars at 900 s mean most sessions in that cell never delivered. The wide Sonnet N0 same-agent interval reflects its 58% success rate.

Sonnet 5.5 (n = 12 per cell)	Phase	Success	$/session	$/fix	ttd median (s)	Tokens (median)
N0	S1	1.00	0.0159	0.0159	10.3	7.9k
N0	same	0.58	0.0404	0.0692	22.4	23.3k
N0	cross	0.67	0.0399	0.0598	20.1	24.3k
LO	S1	1.00	0.0219	0.0219	16.1	17.5k
LO	**same**	**1.00**	**0.0427**	**0.0427**	**22.6**	**29.5k**
LO	cross	0.67	0.0442	0.0663	26.2	29.7k
SH	S1	1.00	0.0319	0.0319	17.4	23.8k
SH	same	1.00	0.0398	0.0398	17.7	35.1k
SH	**cross**	**1.00**	**0.0372**	**0.0372**	**16.1**	**31.2k**

Haiku 5.5 (n = 24)	Phase	Success	$/session	$/fix	ttd (s)
N0	S1	1.00	0.00126	0.00126	8.6
N0	same	0.21	0.00368	0.0176	900
N0	cross	0.08	0.00375	0.0450	900
LO	S1	1.00	0.00160	0.00160	11.6
LO	**same**	**0.96**	**0.00309**	**0.00322**	**17.5**
LO	cross	0.13	0.00394	0.0315	900
SH	S1	1.00	0.00212	0.00212	12.4
SH	same	0.96	0.00390	0.00406	21.3
SH	**cross**	**0.96**	**0.00371**	**0.00387**	**19.8**

Opus 5.5 (n = 6, expl.)	Phase	Success	$/session	$/fix	ttd (s)
N0	S1	1.00	0.0501	0.0501	24.1
N0	same	0.67	0.1019	0.1528	35.2
N0	cross	0.33	0.1050	0.3149	900
LO	S1	1.00	0.0520	0.0520	20.6
LO	**same**	**1.00**	**0.0843**	**0.0843**	**23.7**
LO	cross	0.67	0.1058	0.1587	35.1
SH	S1	1.00	0.0744	0.0744	26.3
SH	same	1.00	0.1039	0.1039	31.7
SH	**cross**	**1.00**	**0.1067**	**0.1067**	**29.1**

$/fix = total cost ÷ fixes delivered. ttd = median time to delivery, failures counted as 900 s. Tokens = median tokens per session. Highlighted rows: the best memory arm for same and cross probes.

Same agent, repeat problem. Local notes took Sonnet from 58% to 100% and cut its cost per fix from $0.0692 to $0.0427 (ratio 0.617). Haiku went from 21% to 96%, with median time to a verified fix falling from 900 s to 17.5 s. Opus went from 67% to 100%. **Teammate, related problem.** Local-only memory did not transfer: 67% (Sonnet), 13% (Haiku), 67% (Opus). The shared BAB delivered 100%, 96% and 100%, at a cost per fix close to the same-agent case.

6.8 Team totals, first solvers included

Model	Cond.	Fixes	Team $/fix	vs N0
Sonnet	N0	27/36	0.0427	1
Sonnet	LO	32/36	0.0408	0.955
Sonnet	**SH**	**36/36**	**0.0363**	**0.849**
Haiku	N0	31/72	0.00672	1
Haiku	LO	50/72	0.00414	0.616
Haiku	**SH**	**70/72**	**0.00333**	**0.496**
Opus	N0	12/18	0.1285	1
Opus	**LO**	**16/18**	**0.0908**	**0.707**
Opus	SH	18/18	0.0950	0.740

Team totals count all 9 sessions per team (S1 + same + cross), so they include SH's seeding overhead: SH's S1 sessions cost 1.4–2× N0's, because the first solvers write to the BAB.

Shared memory had the lowest team cost per fix for Sonnet (−15% vs no memory) and Haiku (−50%), and delivered every fix for Sonnet and Opus. For Opus, with only 2 teams, local-only edged out shared ($0.0908 vs $0.0950); nothing in the Opus team comparison is significant (p ≥ 0.32).

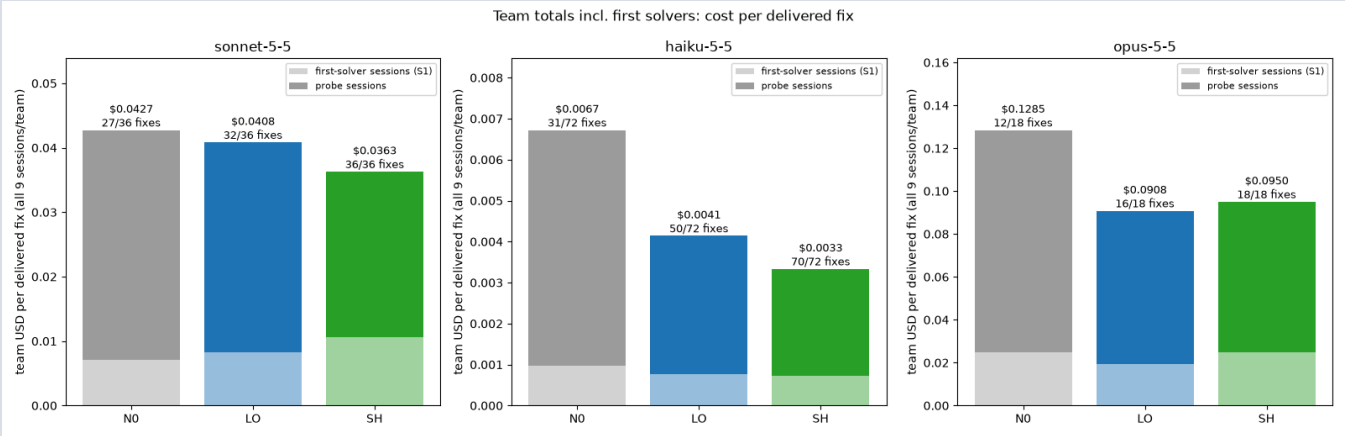

Figure 13. Team cost per delivered fix, including first-solver sessions (light segment) and probe sessions (dark segment), with fixes delivered above each bar.

6.9 Hypotheses: outcomes

#	Model	Primary ratio [95% CI], p (Holm)	Secondary	Outcome
H1 LO/N0, same	Sonnet	1.058 [0.971, 1.149], p 0.316 (0.632)	$/fix 0.617 [0.456, 0.731]; mean ttd 0.056, p 0.0095	fails on primary
	Haiku	0.840 [0.757, 0.936], p 0.014 (0.028)	$/fix 0.183; ttd 0.081, p 0.0005	supported, weaker than ≤ 0.70
	Opus	0.827 [0.749, 0.919], p 0.026 (0.052)	$/fix 0.552 [0.499, 0.613]	exploratory
H2 SH/LO, cross	Sonnet	0.841 [0.743, 0.934], p 0.0145 (0.029)	$/fix 0.561 [0.496, 0.622]; ttd 0.052, p 0.0005	supported, weaker than ≤ 0.70
	Haiku	0.943 [0.876, 1.012], p 0.372 (0.744)	success 0.96 vs 0.13; $/fix 0.123 [0.038, 0.236]; ttd 0.071, p 0.0005	n.s. per session; large per fix
	Opus	1.009 [0.925, 1.105], p 0.846	$/fix 0.672 [0.616, 0.737]	exploratory, n.s.
H2b LO/N0, cross	all	Sonnet 1.109 [1.009, 1.226], p 0.062; Haiku 1.050 [0.981, 1.129]; Opus 1.008 [0.885, 1.158]	an agent's own memory does not help a teammate	holds
H3 team $/fix	Sonnet	SH/LO 0.889, p 0.03; LO/N0 0.955, p 0.44	SH 0.0363 < LO 0.0408 < N0 0.0427	ordering holds; LO < N0 n.s.
	Haiku	SH/LO 0.805, LO/N0 0.616, both p 0.0005	8 teams	supported
	Opus	SH/LO 1.046, LO/N0 0.707; p ≥ 0.32	LO 0.0908 < SH 0.0950 < N0 0.1285	ordering fails
H4 SH/LO, same	Sonnet	0.932 [0.847, 1.029]		holds
	Haiku	1.262 [1.174, 1.357], p 0.0005	SH costs ~26% more on same probes	fails
	Opus	1.232 [1.153, 1.319], p 0.006	also slower: ttd 1.34×	fails

Primary metric: mean cost per probe session. Holm applies across H1 and H2 only; Opus Holm values are informational. Ratios, CIs and p-values are from the frozen analysis output (`deepdive/dist_contrasts.csv`, `dist_h3_team.csv`). Mean ttd ratios are dominated by the 900 s failure penalty, so read them as "how often it got fixed".

6.10 Per use case

Fixes delivered	Sonnet (n = 4)			Haiku (n = 8)			Opus (n = 2)		
	N0	LO	SH	N0	LO	SH	N0	LO	SH
Paylane, same	**4/4**	**4/4**	**4/4**	4/8	7/8	**8/8**	**2/2**	**2/2**	**2/2**
Paylane, cross	**4/4**	**4/4**	**4/4**	1/8	3/8	7/8	**2/2**	**2/2**	**2/2**
Gatekeeper, same	3/4	**4/4**	**4/4**	1/8	**8/8**	7/8	**2/2**	**2/2**	**2/2**
Gatekeeper, cross	**4/4**	**4/4**	**4/4**	1/8	0/8	**8/8**	0/2	**2/2**	**2/2**
geo-api, same	0/4	**4/4**	**4/4**	0/8	**8/8**	**8/8**	0/2	**2/2**	**2/2**
geo-api, cross	0/4	0/4	**4/4**	0/8	0/8	**8/8**	0/2	0/2	**2/2**

- **geo-api is never solved without relevant memory.** Every N0 same and cross cell, and every LO cross cell, scored 0 for all three models. The rule can be discovered from the incident log, but no model discovered it. Most of the success and $/fix gaps come from this use case.
- **Paylane and Gatekeeper** are mostly solvable cold by Sonnet and Opus. There, memory changes cost by ±10–20% rather than deciding whether the fix lands. Opus N0 cross on Gatekeeper failed 0/2.
- **Haiku** fails most cold probes on every use case, so memory has its largest relative effect there.

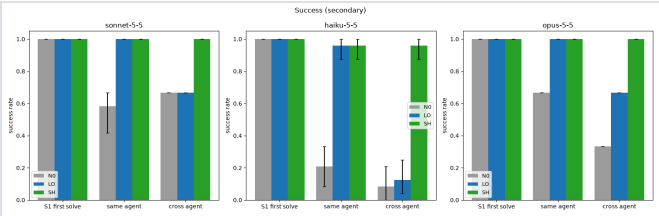

Figure 14. Success by condition and phase (secondary metric).

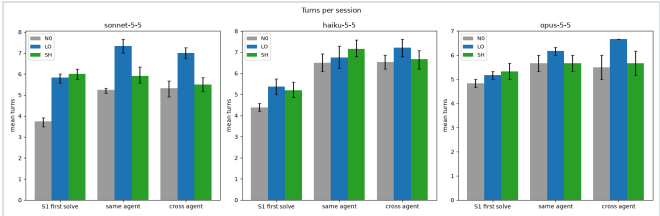

Figure 15. Mean turns per session. Memory arms take more turns (reading and writing notes) even where they deliver more fixes.

6.11 What surprised us

1. **Memory raises tokens per session but cuts cost per fix.** LO and SH sessions read memory and write notes: Sonnet LO same used 1.26× N0's median tokens, and SH about 1.5×. So per-session cost is flat or higher, and Sonnet's H1 fails even though LO delivers 100% vs 58%. The pre-registered primary metric (mean $/session) penalises the condition that actually ships more fixes.
2. **Shared memory has an overhead on same-agent work** (H4 fails for Haiku and Opus), and it makes seeding in S1 cost more. For Opus this overhead outweighs the cross-agent benefit at team level, so LO edges out SH.
3. **Haiku with SH on cross probes (96%) far exceeds Haiku N0 on any probe.** The cheapest model with a shared BAB delivered more reliably than Sonnet with no memory.
4. **A keyword-match artefact inflates "relevant memory seen" for LO cross probes** (0.67–0.83). A bash command that ran `cat /memories` alongside the README, which names the vendor, triggered the keyword match. The clean measure, `relevant_local_before`, is 0 for cross probes by construction. No hypothesis metric uses the inflated field.

6.12 Integrity, amendments and limitations

Audit: clean. An automated audit scanned 381 run-metadata files, 378 transcripts and 2,039 files in total (smoke runs still on disk included). It found no suspicious or symlink commands, no references to hidden tests, oracles or the evals tree, no probes outside the sandbox and no credentials in any transcript. Two runs were flagged for writing to `/memories` via bash; both were checked by hand. One was a regex false positive (a heredoc that wrote to the repo); the other was a write attempt that the read-only mount refused, after which the agent used the memory tool. There was no oracle access and no way around the memory freeze.

Amendment 1 (12:01, after the first Haiku smoke, before any main run): agents viewed `/memories` with the memory tool and then tried to `cat` it in bash, which failed because the directory was not mounted in the sandbox. The agent's own `/memories` is now mounted read-only; writes still go only through the tool. The smoke was rerun. **Amendment 2** (~13:05): a box restart at about 12:44 lost every per-session file of the running Sonnet main block. Its spend is counted at a $7.50 bound; Sonnet was rerun from scratch with 4 teams instead of 8 (conditions are never cut). Hypotheses, metrics and analysis were unchanged by either amendment, and the frozen source files still match their hashes.

Limitations of round 3

- **Lost Sonnet run.** About $7.50 of Sonnet sessions were lost to the restart and not analysed. The rerun has 4 teams (12 probes per cell), so Sonnet, the confirmatory model, has less power than planned.
- **Opus has 2 teams** (6 probes per cell) and is exploratory; its intervals are wide and its Holm values are informational.
- **Three synthetic use cases** with a recorded-traffic discovery path. How hard the ops files are to read sets the size of the no-memory penalty, and geo-api drives much of the gap.
- **Identical probe for same and cross.** This isolates agent identity, but real teammates' tasks differ more. One prior session per agent: no long-horizon growth, stale or conflicting local notes.
- **"Relevant memory seen" is inflated for LO cross probes** by the keyword-match artefact above; it is not used in any hypothesis.
- **Simulated machines.** Agents were isolated sandboxes on one host with a per-team BAB on localhost. Minimal agent loop: a real IDE agent that auto-loads its local memory file could make LO cheaper.
- **Bootstrap CIs on mean ttd are tight** because failures pile up at the 900 s cap; read them alongside the success rates. Wall time depends on API latency and box load.

> **Takeaway**
> An agent's own memory pays off when the same agent hits the same wall again. It does nothing for a teammate. When related problems land on different machines, a shared memory is what carries the lesson: 96–100% cross-agent fixes against 13–67% with local notes alone, and the lowest team cost per fix for Sonnet and Haiku even after paying for the first solvers to write it down. The saving is per fix, not per session, and the shared layer is worth keeping cheap when the agent already knows the answer.

7 Design lessons: tools, retrieval, nudges

7.1 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). Round 3 used the lean toolset for its shared arm.

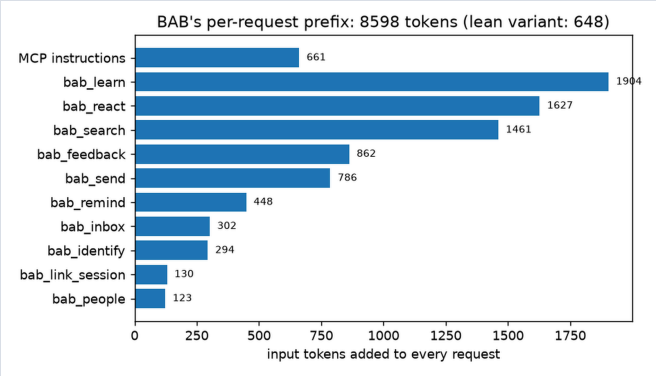

Figure 7. Per-request prefix by tool (Sonnet and Haiku 5.5 counts are identical).

7.2 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 (Sonnet). 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.

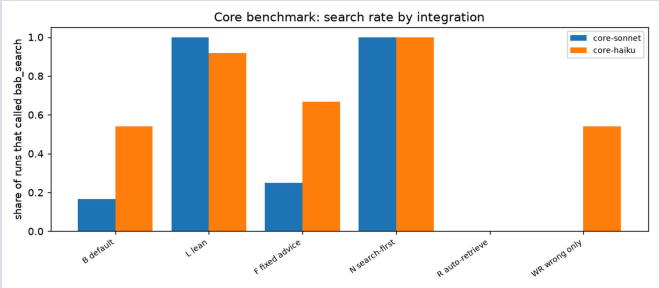

Figure 8. Core suite: share of runs that called `bab_search`. Orange bars are Haiku (complete). Blue bars are the partial Core Sonnet run (~45 of 168 runs), shown in the source chart but not used as evidence. R is 0% by design, because injection replaces search; relevant memory was still seen in 92%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

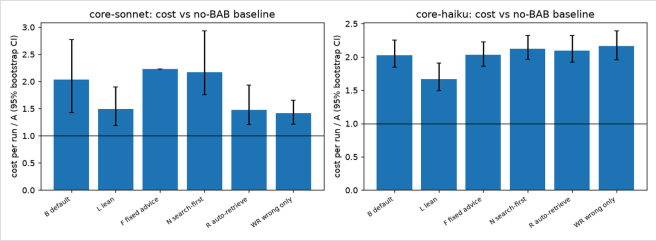

Figure 9. Core suite (Haiku): cost per run relative to no-BAB, with 95% bootstrap CIs. All success rates were 100% (ceiling). Haiku panel only; the source chart also has a Core Sonnet panel based on ~45 partial runs, which we do not use.

7.3 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.

7.4 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; Opus was less sensitive (79% vs 88%). 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.

7.5 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. These are soft distractors; on the hard set (section 5), recall@1 was 0.56 overall and 0.13 for blind-written queries.

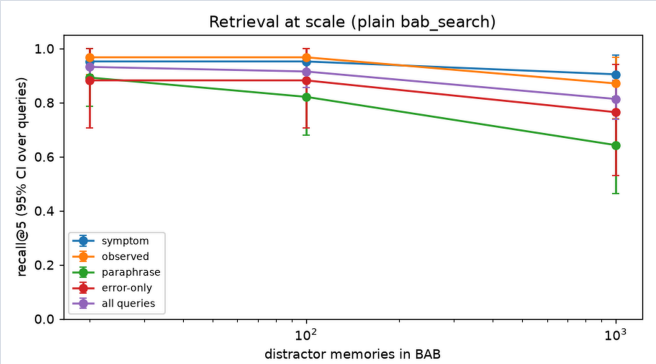

Figure 10. recall@5 by query type vs number of distractor memories (log scale), with 95% CIs over queries.

8 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. Opus also fell back to 12%.
- **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)

9 Limitations, pre-registration and deviations

Limitations (all rounds)

- **Synthetic tasks, written by us.** All 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, and in round 3 only 4 Sonnet teams and 2 Opus teams. 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; in round 3 the LO-cross "relevant memory seen" field is inflated by a keyword-match artefact. Wall-clock time was dominated by API latency and parallelism.
- **Single host.** Rounds 1 and 2 used one BAB server on localhost. Round 3 isolated agents as if on separate machines, but on one host with a per-team BAB on localhost; a real multi-machine, multi-team deployment was not measured.
- **Opus coverage.** Opus 5.5 ran on the team-knowledge suite and as an exploratory arm in round 3, not on round 1 or the core ablations.
- **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. The Opus 5.5 team-knowledge round was not in the pre-registration; it was added later the same day and reused the frozen tasks, memory fixture and conditions. Round 3 was pre-registered in `NEW-EXPERIMENT-PLAN.md` and frozen at 11:54 Lisbon time on 21 Oct 2026, before its first paid run (`deepdive/dist-prereg-FREEZE.txt`, manifest hashes).

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. TK Opus: one interrupt after 197 runs. A resume completed the last 19, and all 216 are reported.

4. 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.
5. Live Haiku colleague seeding for TK was not completed, so Sonnet's fixture is the memory source for all TK conditions.
6. Bootstrap CI whiskers are clamped at zero for plotting only; the numeric CIs are unclamped.
7. 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. Round 3: Amendment 1 (read-only `/memories` mount in the sandbox, smoke rerun) and Amendment 2 (box restart; Sonnet rerun with 4 teams instead of 8, lost spend counted at a $7.50 bound). See section 6.12.

10 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 round-1 and round-2 condition (`--conditions`); the offline retrieval eval, including the hard distractor and blind query sets; a per-run budget cap; and 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

# round 3: distributed teams (sequence runner and frozen analysis)
BAB_EVAL_MODEL=claude-haiku-5-5 node evals/seq.mjs --teams 8 --team-offset 100 --seed 202 --budget-usd 1.00 --
parallel 2 --out runs/dteam/haiku --resume
python3 evals/scripts/analyze_dist.py runs/dteam/sonnet runs/dteam/haiku runs/dteam/opus
```

The round-3 harness (local memory tool, sequence runner, discoverable probe variants, analysis) is on the local branch `feat/evals-distributed` (commits `b1f9e1b`, `6e8c99b`) and has not been pushed yet. Raw data kept for every run: `transcript.jsonl` (every LLM and tool call), `diff.patch`, `grade.txt`, run metadata and BAB server logs, aggregated as per-run CSVs, paired-comparison tables and `stats.json`; round 3 adds `results.json/csv`, a spend ledger and the `dist_*.csv` tables.

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). The Opus 5.5 team-knowledge round cost $17.93 for 216 runs, under a $20 cap, for about $29.36 across rounds 1–2. Round 3 cost $17.41 including the $7.50 bound for the lost Sonnet run, under a $30 ceiling.

11 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. **Give agents local memory for repeat problems, and a shared layer for the team.** Local notes fixed repeat problems reliably but never reached a teammate; only shared memory did (96–100% vs 13–67%).
4. **Measure cost per verified fix, not cost per session.** Memory sessions use more tokens; the saving appears in fixes delivered and time to a verified fix.
5. **Keep the tool surface lean**, and keep shared lookups cheap when local memory already has the answer. Tool schemas are paid on every request.
6. **Don't rely on agents to search unprompted.** Retrieve automatically at the start of a task, or add an explicit search-first instruction.

7. **Treat memory hygiene as a first-class feature:** prompted, structured capture; visible age and verification; down-weighting outdated entries; and retrieval tested against realistic, overlapping memories.
8. **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 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; and better ranking when memories share vendor and subsystem words. We have drafted these as product issues for the project.

12 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), GAPS.md (round 2, including the Opus 5.5 round and the hard retrieval set), DEEPDIVE.md (memory-type grouping and the condition-A re-check) and DISTRIBUTED-RESULTS.md with NEW-EXPERIMENT-PLAN.md (round 3). They and their analysis files (`gaps/analysis/*.csv`, `deepdive/dist_*.csv`) are the source for every number in this report. The "cost per successful fix" figures in section 4.6 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.