

MEMORABLE: PROCEDURAL MEMORY FOR AGENTS

TECHNICAL WHITE PAPER, OCTOBER 2026
BENCHMARK REPORTS SO FAR

Nikhil Krishnaswamy & Advaiyt Sane
Memorable
{nikhil,advaiyt}@memorable.sh

ABSTRACT

Memorable is procedural memory for tool-using agents. It records the tool calls an agent made on a task, turns that trace into a procedure by rules, with ordered steps, the artifacts each step read and wrote, and the command that verified the result, and hands the procedure back the next time a similar task arrives, as a short pointer in the prompt for coding agents or as a checked step-by-step replay for browser agents. This paper reports what that did on every benchmark we have run so far. On *Proced-Mem*, a public retrieval benchmark, our ranker reaches a mean average precision of 0.959 against the published baseline’s 0.795. On two partner coding fixtures the agent needs 19% fewer turns in Claude Code and 40% fewer tool calls in Codex, with the pass rate rising from 80% to 91%, and the learned procedure, at 293 tokens, cuts turns by 19% where a 15,593-token hand-written skill cut none. On *Second Visit*, a browser benchmark in which the page changes between visits, checked replay with handoff finishes 211 and 215 of 240 visits against 174 and 187 for the same agent without memory, at half the model cost. Every result is scored by a check the agent cannot touch, and the headline results held on rerun.

1 RESULTS

Memorable stores the procedure an agent followed on a task and hands it back the next time a similar task arrives. The procedure is the ordered list of tool calls the agent made, what each of them read and wrote, and the command that verified the result at the end, and it is taken from the trace by rules rather than by a model, so the same run always yields the same procedure. On every benchmark in this brief the agent with memory beats the same agent without it. On *Proced-Mem*, a public retrieval benchmark, our ranker reaches a mean average precision of 0.959 against the 0.795 of the baseline the benchmark publishes, which places it first among the five retrievers we ran. On the two partner coding fixtures the agent needs 19% fewer turns in Claude Code while 125 of 125 runs pass, and 40% fewer tool calls in Codex while the pass rate rises from 80% to 91%. Where a hand-written skill of 15,593 tokens left the turn count unchanged, the learned procedure of 293 tokens cuts it by 19%. On a browser benchmark in which the page changes between visits, checked replay finishes 37 and 28 more visits out of 240 than the same agent without memory, and it does so at half the model cost. Every one of these numbers is scored by a check the agent cannot touch, and the headline rows were rerun and held: two judge runs on *Proced-Mem*, two runs of 25 on the Claude Code fixture, three runs of 25 on the Codex fixture, and two held-out page sets in the browser.

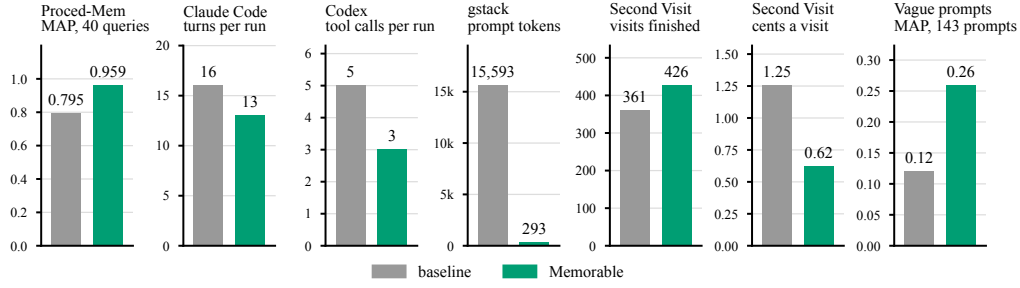


Figure 1: Baseline against Memorable on each benchmark. Second Visit panels pool the two held-out page sets.

Table 1: Results by benchmark.

Benchmark	Baseline	Memorable	Gain	Replication
Proced-Mem (public retrieval benchmark, 40 queries, gpt-5 judge)	MAP 0.795, the benchmark’s own MiniLM baseline	MAP 0.959	+21% MAP ; paired +0.16 (+0.09 to +0.23), better on 26 queries, worse on 4	two judge runs: 0.959, 0.947
Claude Code, GBrain three-bug fixture, Haiku 4.5	16 turns	13 turns	19% fewer turns , $p < 10^{-4}$; 125 of 125 runs passed	two runs of 25 per arm
Codex, Quartermaster three-bug fixture	5 tool calls; 80% pass	3 tool calls; 91% pass	40% fewer tool calls , $p=5 \cdot 10^{-5}$	three runs of 25 per arm
Codex, five task families on one shared store	4 tool calls; 87% pass	2 tool calls; 97% pass	50% fewer tool calls ; the right family recalled on every run	one run of 30
gstack /investigate on Claude Code	15,593-token skill, turns unchanged	293-token procedure, turns -19%	98% less context per prompt	replicated
Second Visit (browser, page changed between visits, gpt-4.1-mini)	174 and 187 of 240 visits	211 and 215 of 240	+37 and +28 visits at half the cost ; ahead on 8 and 10 of 10 tasks	two held-out page sets
Vague prompts (143 real prompts that need the conversation)	MAP 0.12	0.21 with no model call; 0.26 with a resolved query and trace titles	1.8× to 2.2× MAP ; +0.09 (+0.06 to +0.13), +0.14	second judge agrees on the order

2 METHOD

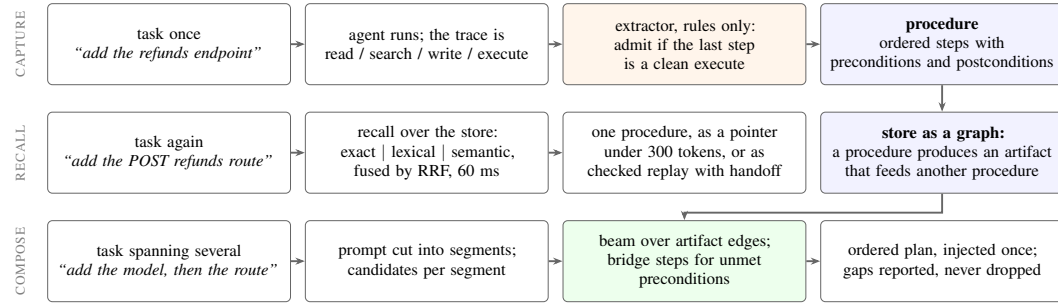


Figure 2: What is stored and the three ways it is used. Top: a run becomes a procedure by rules. Middle: recall hands one back as a pointer, or replays it step by step with a check after each step. Bottom: a plan assembled from several procedures over the artifacts they produce and consume.

Capture. Capture starts from a hook the harness already has. In Claude Code it is gbrain’s SessionEnd hook, in Quartermaster it is one adapter wrapped around the tool set, and anywhere else it is memorable ingest – fed a JSON trace. The hook sends three things for each step, the tool’s name, the one argument that identifies what it touched, and the result code, and nothing more. The transcript, the contents of files and the model’s reasoning never leave the machine.

Extraction, by rules. Each step keeps its tool, its target and an activity class, which is one of read, execute, write, browse or search. An artifact that a step read and that no earlier step created becomes a precondition of the procedure, every artifact the run created becomes a postcondition, and the last execute-class command that exited cleanly is kept as the command that proves the procedure worked. A trace that ends without such a command stores nothing at all. The only model calls on the whole path are a small model writing the title and bge-m3 embedding it, so the same trace always gives the same procedure.

The store as a graph. A procedure produces the artifacts it writes created and consumes the targets of its reads, and an edge runs from every producer to every consumer. Artifacts that more than 15% of the store touches carry no edge, because they would connect everything to everything. The graph is what lets a plan reach a step the prompt never names, such as the migration that a later step reads.

Recall. Recall runs three retrievers over the store and fuses their rankings with reciprocal rank fusion at $k=60$, as Figure 3 shows. The exact retriever fires when a stored path or command appears verbatim in the task, the lexical retriever scores IDF-weighted overlap and needs two or more shared tokens, and the semantic retriever takes the bge-m3 cosine between the task and the stored title, with a floor of 0.50 when another retriever also matched and 0.62 when it stands alone. The whole pass takes about 60 ms. The top hit is rendered as a pointer of under 300 tokens and placed in the prompt once per session in Claude Code and on every turn in Quartermaster, where a timeout and a size cap bound its cost.

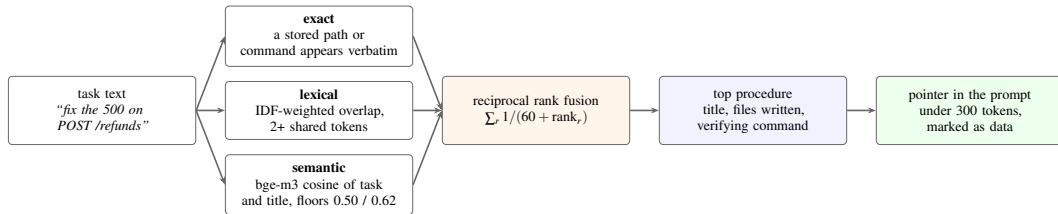


Figure 3: Recall: three retrievers, one fused ranking, one short pointer.

Checked replay in the browser. In the browser the procedure is compiled from two agreeing first visits into steps that each carry an accessibility-tree locator and a check. Replay acts only when exactly one visible control matches under the same landmark ancestors, waits for the check after each step, and at the first failed check hands the task to the model together with the steps already done and the step it stopped on. 132 of 240 return visits finish this way with no model call and no wrong action, and a handoff that succeeds is written back into the procedure so that the next visit replays further.

3 BENCHMARKS

Proced-Mem: MAP 0.959, 21% over the published baseline. Proced-Mem is a public benchmark, arXiv:2511.21730, built from 336 ALFWorld trajectories and 40 queries, with a gpt-5 judge that scores each retrieved trajectory for how useful it would be on the query’s task. We ran its judge and its scoring exactly as shipped. The benchmark publishes two baselines of its own, MiniLM over each trajectory’s task, state and action text at a MAP of 0.79 and MiniLM over the actions alone at 0.72. Our ranker embeds the task text with bge-m3 and fuses the exact and lexical retrievers into the ranking, and it scores 0.959, first of the five retrievers we ran, with 0.947 on a second judge run against 0.802 and 0.835 for our two reruns of the baseline. Paired over the 40 queries the ranker leads the baseline by 0.157, with a 95% interval from 0.09 to 0.23, and it is better on 26 queries and worse on 4. Two thirds of that gain comes from choosing what to embed, since the task text alone under the same MiniLM is already 0.105 ahead, and the stronger embedder and the fusion then add between 0.02 and 0.03 each.

Claude Code and Codex: 19% fewer turns, 40% fewer tool calls. On a rerun of the same three-bug fixture the pointer names the files the previous run wrote and the command that proved the fix, so the agent skips the phase in which it reads the repository to find them. In Claude Code that takes a run from 16 turns to 13 with every run passing and input tokens unchanged, and in Codex it takes tool calls from 5 to 3, with the three replications landing at 5 to 3, 3 to 3 and 4 to 3. Every Codex run that acted at all passed, in both arms, and all 22 failures were runs that made zero tool calls, 15 of them without memory and 7 with it, so the pointer also keeps the agent from giving up on the first miss. When five task families share one store, recall picked the right family on every run and tool calls fell from 4 to 2.

gstack: 98% less context, turns down 19% where the skill moved them 0%. A skill is a general manual that is loaded whole into every prompt. The `/investigate` skill in gstack is 15,593 tokens, and on our fixture it added 55% to the input tokens while changing the turn count by 0%. One session solved the fixture cold, and the procedure learned from that session is 293 tokens, 2% of the skill, and cut turns by 19%.

Second Visit: +37 and +28 visits of 240 at half the cost. Second Visit is ten form tasks with 24 page variants each, where the page changes between the first visit and the second in one of eight ways: layout, attributes, label text, ambiguity, an interstitial, a decoy, an impostor control, or no change at all. Replay on its own handles every layout, ambiguity and decoy change without calling a model, and the handoff covers the label, attribute and interstitial changes. The agent with no memory finishes 174 and 187 of 240 visits on the two held-out page sets, while replay with handoff finishes 211 and 215, ahead on 8 and 10 of the 10 tasks with sign-flip p of 0.018 and 0.002, at 49% and 50% of the model cost. The 5.5 cents it costs to build a task’s memory is repaid after nine return visits.

Vague prompts: MAP 0.12 to 0.26, against an oracle of 0.28. A third of real prompts only make sense together with the turns before them, and they arrive at a median turn of 26, whereas recall as deployed fires on the first prompt of a session, which is vague only 9 times in 376. Running recall on every prompt and waiting for the agent’s first three tool calls lifts MAP on those prompts from 0.12 to 0.21 with no model call, and letting a small model resolve the prompt against the conversation, with titles written from the trace, reaches 0.26 against an oracle of 0.28.

4 DEPLOYMENTS

Partner	Setup	Result on the site
gstack	Garry Tan’s skill suite on Claude Code with gbrain; session-end hook captures, <code>POST /v1/extract</code> builds the procedure, a prompt hook recalls it	293 tokens against 15,593 for <code>/investigate</code> ; turns down 19%; 125 of 125 runs passed
Quartermaster	YC’s Codex harness in a Docker sandbox; one adapter captures every loop; the orchestrator injects the recalled procedure every turn; procedures live in QM’s own Postgres	tool calls 5 to 3 (40% fewer), pass 91% against 80%; five families on one store: 4 to 2 calls, 97% against 87%