Enterprise Brain: Outcome-Learned Agent Memory

Rahul Balakavi Saakshi Singhal Rahul Goel Amit Prakash
AmpUp AI
research@ampup.ai

Abstract

We present Enterprise Brain, a memory that lets a small agent recover a large model's performance without changing its weights. Enterprise Brain keeps two kinds of memory: a temporal memory of facts extracted from the organization's sources, each dated and cited, and a procedural memory of how the work is done. Procedural memory starts from skills extracted from the organization's knowledge base and grows with advice learned from the agent's own graded traces. A stronger model groups the agent's past runs into situations, the kinds of work the agent repeats, and writes advice for each situation from the runs that went right and the runs that went wrong. Because Enterprise Brain learns the advice from the organization's own work, the advice is domain-specific: it records conventions of the organization's systems and graders that are not captured by general models. The advice further improves by hill climbing. On public benchmarks across four domains, Enterprise Brain lets a cheap model (Claude Haiku 5.5) surpass or close most of the gap to an expensive one (Claude Sonnet 5.5), at a fraction of the cost. While Enterprise Brain shows significant gains on benchmarks with repeated tasks, it does not degrade performance on benchmarks with fewer repeated tasks.

1 Introduction

An organization's agents do the same kinds of work again and again: an on-call agent investigates the same shop's incidents, an assistant answers the same person's email, an analyst queries the same CRM. Much of this work is graded: tests pass or fail, and the people it is for accept or reject it. Large models do it well, but they cost an order of magnitude more per task than small ones. DolphinBench [Rathi et al., 2026] tests an assistant that acts for one person through their email, chat, calendar and CRM. On it, Claude Sonnet 5.5 costs 11.9¢ a request and Claude Haiku 5.5 0.73¢, so a thousand requests a day cost $119 with Sonnet and $7.30 with Haiku. An organization would rather pay the small model's price, if the small model could do the work as well.

Agent memory is the natural place to close that gap, because it changes what an agent knows without changing its weights. Yet memory systems store what was said and retrieve what is similar to the next request [Packer et al., 2023, Chhikara et al., 2025, Rasmussen et al., 2025, Xu et al., 2025]. Whether the work succeeded does not change what they serve. Methods that learn from an agent's experience let the agent reflect on its own runs and usually distill them into a single prompt or playbook [Shinn et al., 2023, Zhao et al., 2024, Suzgun et al., 2025, Zhang et al., 2025, Ouyang et al., 2025]. An organization's agent faces many kinds of work, though, each with its own procedure and pitfalls. We ask whether a stronger model, reading a small agent's graded runs, can write down what each kind of work requires, so that the small agent does it as well as the larger model.

We present Enterprise Brain, a domain specific memory system with two parts (Fig. 2). Its temporal memory holds facts extracted from the organization's sources, each dated and cited. Its procedural memory holds how the work is done: skills extracted from the organization's knowledge base before any run, and advice learned from the agent's graded traces. A stronger model groups the agent's past runs into situations, kinds of work it finds in the tasks themselves with no labels given. For each situation it reads the runs that went right and the runs that went wrong, and writes advice in four parts: when the advice applies, the steps that led to success, the mistakes the failed runs made, and which runs back each point. Advice is goal-oriented through the outcome: it aims at what the grader rewarded, and the grader's reason says why. It is also domain-specific, because it is learned from the organization's own graded work: it records what holds in this domain and nowhere else; which date closes a deal in this CRM, how this shop's faults show up in its telemetry, what this person's requests require. The advice improves by hill climbing: the agent practises with the current advice, its graded practice joins what the advice is learned from, and a new version is kept only when it beats the best so far on validation. Before each task a small router serves the advice for the task's situation.

We evaluate Enterprise Brain with Claude Haiku 5.5 as the agent and Claude Sonnet 5.5 as the larger model (Fig. 1), and against a generic memory, OpenViking [Volcengine (ByteDance), 2026], which stores the same sources and graded runs and retrieves what resembles each task. On ORCA-bench Enterprise Brain surpasses Sonnet, finding 35.2% of incident root causes against 17.8%, at one seventeenth of its cost. It surpasses Sonnet on CRMArena-Pro as well, answering 76.7% of questions against 67.6%, at one fourteenth of its cost. On DolphinBench it closes 99% of the gap to Sonnet, at one fifteenth of Sonnet's cost per request. On ALFWorld it raises Haiku from 79.1% to 97.8% of household tasks against Sonnet's 99.3%, closing 93% of the gap at one tenth of Sonnet's cost. The advice carries the gain: against Enterprise Brain serving only the skills extracted by the same stronger model,

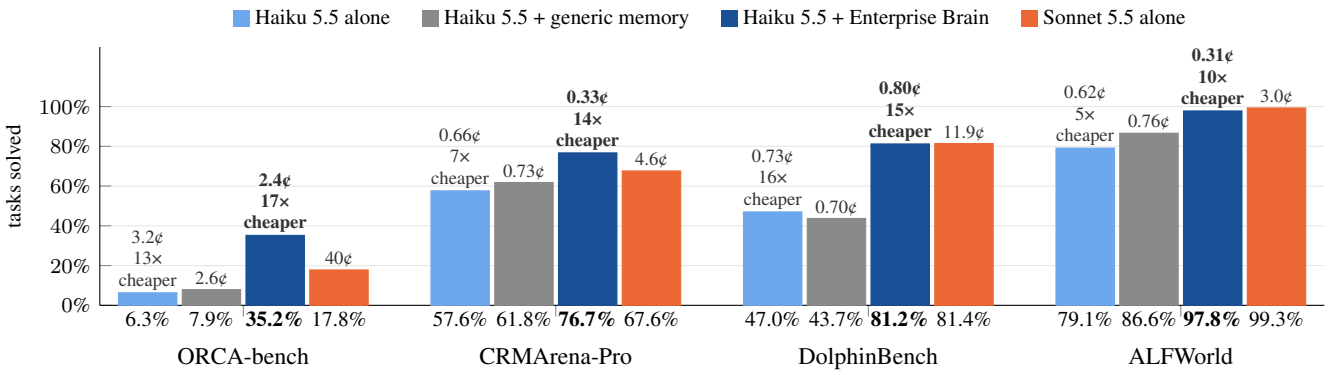


Figure 1: Enterprise Brain closes the gap to a larger model at a fraction of its cost. Tasks solved by Claude Haiku 5.5 alone, Haiku with a generic memory (OpenViking, given the same sources and graded practice runs), Haiku with Enterprise Brain, and Claude Sonnet 5.5 alone: incident root causes found (ORCA-bench), CRM questions answered (CRMArena-Pro), requests completed for one person (DolphinBench) and household games solved (ALFWorld). Under each bar is the share of tasks solved, and above it the cost per task and, for Haiku, how many times cheaper it is than Sonnet. With Enterprise Brain, Haiku beats Sonnet on two benchmarks and comes within 1.5 points on the other two. DolphinBench's generic memory ran in a session of its own, where Haiku with the history as files passed 38.5%.

advice adds +24.9 points on ORCA-bench and +9.4 on DolphinBench. From the same runs the generic memory adds 2 to 8 points, where Enterprise Brain adds 19 to 34. On SWE-ContextBench and τ^2-bench, where there is less for advice to teach, it does not degrade performance. Learning costs a fraction of the practice runs it learns from, and because every later task runs on Haiku instead of Sonnet, the method pays for itself after a few dozen to a few thousand tasks.

This paper makes three contributions:

- Enterprise Brain, a memory that pairs a temporal memory of cited facts with a procedural memory of skills from a knowledge base and advice from graded traces, served to any agent harness (Section 3).
- A method for learning domain-specific advice from graded traces and improving it by hill climbing: a stronger model writes advice for each situation it discovers, from right and wrong runs, and a new version is kept only when it beats the best on validation (Sections 3.3 and 3.4).
- Evidence on six benchmarks. With Enterprise Brain a small model gains 19 to 34 points on four of them, at a tenth to a seventeenth of a larger model's cost: it beats the larger model on two, matches it on DolphinBench and comes within 1.5 points on ALFWorld. On the other two it does no harm (Sections 4 to 6).

2 Related work

Memory for agents. Agent memory systems store what was said and retrieve what is similar: MemGPT pages context in and out of a model's window [Packer et al., 2023], Mem0 extracts and updates salient facts [Chhikara et al., 2025], Zep keeps them in a temporal knowledge graph [Rasmussen et al., 2025], A-MEM links notes [Xu et al., 2025], and MIRIX and Memp add procedural memory [Wang and Chen, 2025, Fang et al., 2025]. OpenViking, which we run as a baseline, keeps an agent's memories, resources and skills in one context database and updates them from the agent's sessions [Volcengine (ByteDance), 2026]. These systems decide what to keep by what was said. Enterprise Brain keeps a temporal memory of cited facts as well, and decides what procedure to serve by how the work turned out.

Learning from experience. Reflexion keeps verbal reflections across attempts [Shinn et al., 2023]. ExpeL, Agent Workflow Memory and ReasoningBank distil insights, workflows and strategies from past trajectories [Zhao et al., 2024, Wang et al., 2024b, Ouyang et al., 2025], and Voyager grows a library of code skills [Wang et al., 2023]. Dynamic Cheatsheet, Training-Free GRPO and ACE evolve a single cheatsheet, experience library or playbook [Suzgun et al., 2025, Cai et al., 2025, Zhang et al., 2025]. AutoGuide and Trajectory-Informed Memory organize guidance by context or task category and select it at test time [Fu et al., 2024, Fang et al., 2026]. In these methods the model that writes the guidance is the model that reads it. Enterprise Brain instead has a stronger model write advice for each discovered situation from a small agent's graded runs, and routes each new task to its situation's advice.

Optimizing text against validation. GEPA evolves prompts by reflecting on traces [Agrawal et al., 2025], Mixture-of-Prompts learns one instruction per region of the input space and routes inputs to it [Wang et al., 2024a], M* evolves the program that stores and retrieves memory [Pan et al., 2026c], and SkillOpt edits a single skill document from a frozen agent's scored rollouts, accepting an edit only when it improves a held-out validation score [Yang et al., 2026]. Enterprise Brain applies the same acceptance rule to a whole memory, a situation map and its advice, relearned from the agent's cumulative practice.

Distilling a stronger model into context. Other work moves a strong model's competence into a weak model's context without training it: rules written from a weak model's mistakes [Boateng et al., 2025], skills written from the divergences between teacher and student rollouts [Yuan et al.,

2026], instructions written by a slow teacher for a fast student [Pan et al., 2026a], and memory built from a larger model's runs [Han et al., 2025]. D-Skill also reports the 134 unseen ALFWorld games, where its students rise from 55–59% to 83–84% with other teacher and student models than ours. In Enterprise Brain the teacher never acts: it reads only the student's own graded runs, so its advice records what this agent gets wrong in this organization and what the grader rewarded instead. Skills from the knowledge base follow Corpus2Skill and Anything2Skill [Sun et al., 2026, Pan et al., 2026b].

3 Enterprise Brain

Enterprise Brain turns three things an organization already has into context for its agents: its sources, its knowledge base, and the graded outcomes of its agents' work (Fig. 2). From the sources it extracts *facts*, its temporal memory (Section 3.1). From the knowledge base it extracts *skills*, and from graded runs it learns *advice*. Together these are its procedural memory (Sections 3.2 and 3.3). The advice improves by hill climbing (Section 3.4). Enterprise Brain never runs the agent: any harness that speaks the Model Context Protocol, or can place text in the agent's prompt, can use it. Everything it learns is stored as versioned documents with the sources or runs it came from. Steps that run rarely and are read on every later task (extracting skills, learning advice) use a strong model, for example Claude Opus 5.5. Steps that run per source or per task (extracting facts, routing) use a small one, such as Claude Haiku 5.5.

Problem setup. Let π_S be a small agent and π_L a larger one, and let $J(\pi)$ be expected task performance on an organization's recurring work. A run is one attempt at a task, $r = (q, \tau, y, \rho, e)$: the task, the agent's trace, its answer, a reward $\rho \in [-1, 1]$ from the organization's grader, and the grader's reason e. From the organization's sources, its knowledge base and a set of graded practice runs R of π_S, Enterprise Brain builds a memory M that serves context $M(q)$ before each task, so that the memory-conditioned agent π_S^M improves on π_S without any change to its weights. We measure how much of the larger agent's advantage it recovers, on test tasks disjoint from R:

$$\textsc{GapClosed}(M) = \frac{J(\pi_S^M) - J(\pi_S)}{J(\pi_L) - J(\pi_S)}, \qquad (1)$$

together with the cost per task of π_S^M against π_L.

3.1 Temporal memory: facts

A schema declares the organization's entity types and their attributes, each with a value type and a cardinality: `one`, where a newer value supersedes the older, or `many`, where values accumulate. A small model reads each source with the schema and writes typed facts $f = (s, p, v, t, \beta, \epsilon)$: a subject, a predicate from the schema, a value, the time t the value became true, the basis β of that time (stated in the text, inferred from a relative date against the source's own date, or the source's date), and the evidence ϵ, the exact part of the source it comes from. Nothing is overwritten: a later fact supersedes an earlier one only for reads after its time, so the agent can ask what was true at any moment, and staleness or a passed deadline is computed at read time from the facts' own dates. The agent searches the facts as it works. Appendix A gives the details.

3.2 Procedural memory: skills from the knowledge base

Before any run is graded, an agent can still be told how the organization says its work is done. A strong model extracts *skills* from the documents the organization chooses (policies, runbooks, product documentation): each skill says when it applies, gives numbered steps with the exact tools, values and checks the documents give, an *If* line for each edge case, and the documents it rests on. A skill that cites none is dropped. Skills are extracted incrementally as documents arrive and are cached by request, so the same documents always give the same library (Appendix A). They are what an agent is served before any advice exists, and what advice starts from: in our experiments, Enterprise Brain with skills and no advice is the *skills only* arm.

3.3 Procedural memory: advice from graded traces

Graded traces. Rewards come from wherever the organization already grades work: tests, a verifier, a reviewer, the person the work was for. The grader's reason is the most informative part of a run, because it says what the grader expected.

Situations without labels. Advice is learned per *situation*: a kind of work at the grain at which one procedure applies. No task type is given. The learning model sorts runs into a two-level map, category → situation, by the kind of work their task needs, seeing only the tasks, never their outcomes, in batches of 60 matched against the map so far, and the map grows when nothing fits. On ORCA-bench one fold's map has a single category, *root cause analysis of reported storefront incidents*, and five situations, among them *diagnose a vague site-wide complaint*. Each situation is linked to the one to three skills that serve it.

Learning advice from right and wrong runs. Enterprise Brain writes advice for a situation once at least two of its runs are wrong, so a situation the agent always gets right has no advice. The learner reads up to 40 of the situation's runs, the newest wrong and newest right ones, keeping at least ten right runs where the situation has that many. For each run it sees the task, the end of the trace, the answer, and the reward with its reason, under a handle the advice can cite. It also reads the skills linked to the situation. The advice has four parts (Fig. 3). *When* says which tasks it applies to, and *Steps* is the procedure that led to the rewarded outcome. *Pitfalls* names the mistakes of the wrong runs, each with the runs that made it, including where a linked skill is wrong. *Evidence* counts the runs it explains. Advice is a general procedure for the domain and never records task-specific information. It records what the runs revealed about the domain: the organization's conventions and what its graders accept. Learning is goal-oriented through the

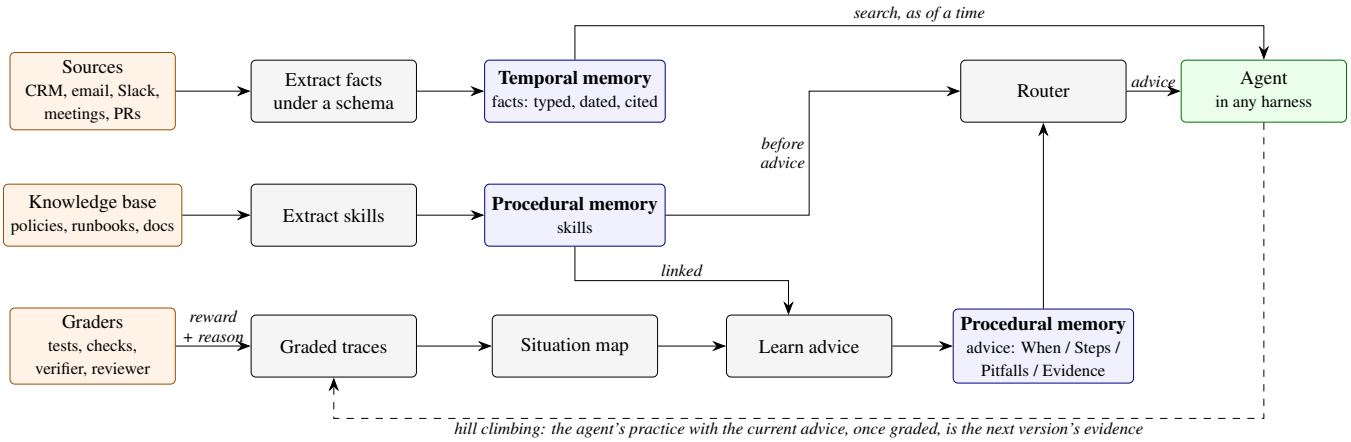


Figure 2: Enterprise Brain. The temporal memory holds facts extracted from the sources, and the procedural memory holds skills extracted from the knowledge base and advice learned from graded traces. Before a task the router serves one situation's advice (or, before any advice exists, its skills), and the agent searches the facts as it works.

outcome: the learner is not shown the task's goal in words, because the rewards and their reasons already say what success meant. Advice learned this way scores higher than advice learned with the goal stated in words (Section 4). Advice is rewritten only when its situation has new evidence (five new runs or two new wrong ones). Otherwise it stays as it was, with no model call, and each version records the runs it was written from.

Serving. Before each task, the small model reads the task and the situation map, which has one line per situation, and names the situation the task belongs to, or none if no situation fits. With more than 60 situations, it sees only the 60 closest to the task (Appendix A.3). A situation with advice is served *the advice alone*, and one with no advice yet is served up to two of its linked skills. Serving the advice without its skills is deliberate: the advice already corrects them where the runs showed them wrong, and adding them back costs accuracy (Section 4).

3.4 Hill climbing: practise, relearn, validate

Advice can only correct mistakes that appear in the runs it learns from, and the most useful runs are the agent's own attempts with the current advice, because they show what the advice still gets wrong. Enterprise Brain therefore improves its advice by hill climbing (Fig. 4). The tasks are split by unit (an incident, a person's requests), with a fixed seed, into train, validation and test (10/10/80). Version v_0 is the memory before any run (skills only). At each round the agent practises on the train tasks with the best version so far, and its graded runs join the pool. The map and advice are learned again from the whole pool, and the new version replaces the best only if it scores higher on validation. The pool is cumulative and on-policy, so advice that misled is visible as such in the next round. Besides adding runs, a round may change how the learner groups them: on DolphinBench, round 2 pools every person's runs into one store, where round 1 kept one per person (Section 6). Validation is only ever scored, never shown to the learner, and the climb stops after two rounds with no validation gain or eight versions. The best version and the baseline then run on test.

4 Experiments

We evaluate Enterprise Brain on benchmarks where memory can help, because an agent does the same kinds of work repeatedly. On each, advice is learned from graded practice runs of the small agent on tasks other than those scored, and every reward comes from the benchmark's own grader.

Benchmarks. *ORCA-bench* [Gong et al., 2026]: from a vague shopper complaint, an on-call agent names every root cause of an incident in an instrumented online shop, and when it started, from the shop's code, metrics, traces and logs. *CRMArena-Pro* [Huang et al., 2025]: business questions about a simulated Salesforce organization of 29,221 records, answered with read-only SQL, and an answer is correct when its token F1 against the expected answer is at least 0.5. *DolphinBench* [Rathi et al., 2026]: an assistant acts for one person through email, Slack, Discord, calendar and CRM tools over five years of their messages, and a request passes when every check on the resulting actions passes. *ALFWorld* [Shridhar et al., 2021]: household tasks in a text environment (find, heat, cool, clean or place an object). A game passes when the environment reports the goal met, and we score all 134 games of its unseen validation split. Section 5 adds SWE-ContextBench and τ^2-bench. Practice and test never share a task (Table 1). ORCA-bench's practice runs come from other days of the shop, and ALFWorld's from training games in scenes the scored games never use. CRMArena-Pro's come from other questions of its training split, which also supplies the 330 we score. The benchmark's own test split was not run. DolphinBench's come from other requests of the same people, split 10/10/80 into practice, validation and test.

Models and conditions. The small agent is Claude Haiku 5.5, in each benchmark's usual harness (terminus-2 with a 150-step cap on ORCA-bench, the benchmark's own loop on CRMArena-Pro and ALFWorld, with a 50-step cap on ALFWorld, Claude Code on DolphinBench). Claude Opus 5.5 extracts skills and learns advice, and Haiku routes. We hill-climb the advice on DolphinBench, whose split has a validation set (Section 6). On the other benchmarks advice is learned once

Advice: Diagnose a vague "users are reporting site issues" report

Situation: root-cause analysis → diagnose a vague site-wide complaint. Learned from 40 graded runs, 10 right and 30 wrong.

When

Use this when the report names no feature or symptom, only something like "users are reporting site issues". In every one of these runs, the expected root causes were runtime feature-flag changes, each with a time, or no root cause at all.

Steps

1. **Pin the clock.** Convert the report time and the current time to UTC.
2. **Get the flag catalogue** from `src/flagd/demo.flagd.json`.
3. **Scan the whole day** for flag-driven symptoms, in every service, not just the report window.
4. **Time each change** from 1-minute histograms of each symptom's first occurrence.
5. **Report every flag change** from the day up to the current time.

Pitfalls

- **Treating "no HTTP 5xx" as no incident.** Most injected faults are gRPC errors or latency that never show in HTTP status codes. (9 runs)
- **Stopping at one flag.** Runs that named only the flag inside the report window still failed when other flags changed that day. (7 runs)
- **Using the flag names from the feature-flag skill.** The skill says `adServiceFailure`, but the deployed key is `adFailure`.

Evidence

Explains all 30 wrong runs: 11 concluded there was no incident, 18 named the wrong cause or only some of the flags, and 1 raised a false alarm on a healthy control.

Figure 3: Advice learned on ORCA-bench (an excerpt, given in full in Appendix E). Each pitfall comes from the runs that made the mistake, and the last one corrects the skill extracted from the shop's documentation. Appendix G shows advice learned on three other benchmarks.

Algorithm 1 Hill climbing the advice

Input: tasks, agent, grader, patience P, at most V versions

Split the tasks by unit into train, validation and test.
best ← skills only, scored on validation
pool ← no runs
repeat
  run the agent with *best* on every train task
  grade those runs and add them to *pool*
  new ← learn map and advice from all of *pool*
  if *new* beats *best* on validation: *best* ← *new*
until P rounds pass with no gain, or V versions exist
Score *best* and the baseline on test.

Figure 4: Hill climbing the advice. Practice runs are learning material and are never scored.

from the practice runs. We report three conditions: Haiku alone in the benchmark's standard setting (on DolphinBench, the person's history as files), Haiku with Enterprise Brain, and Claude Sonnet 5.5 alone, run through Claude Code. To isolate the advice we also run Haiku with generic memory (OpenViking [Volcengine (ByteDance), 2026]) and with Enterprise Brain serving skills only. Within a benchmark every arm has the same tasks, tools and step budget.

Metrics. We report each benchmark's own metric, the gap closed (Eq. (1)), and the model cost per task at list price. Differences are paired by task, and the tables give 95% bootstrap intervals over tasks (clustered by question on ORCA-bench and by source task on ALFWorld, whose 134 games come from 47 tasks).

Benchmark	Scored	Runs	Practice	Grader
ORCA-bench	253	1	454 runs, other days	task grader
CRMArena-Pro	330	1	660 questions	token F1 ≥ 0.5
DolphinBench	480	1	60 requests, 180 runs	action checks
ALFWorld	134	1	60 games, 120 runs	environment's goal check
SWE-ContextBench	217	2	1,006 issues	hidden tests
τ^2-bench banking	97	3	592 conversations	DB and action checks

Table 1: Benchmarks: tasks scored, runs per task, and the graded practice runs advice is learned from.

4.1 Main results

On DolphinBench, Enterprise Brain raises Haiku from 47.0% to 81.2% of requests against Sonnet's 81.4%, closing 99% of the gap, at 0.80¢ a request against 11.9¢, one fifteenth of Sonnet's cost (Fig. 1). On ORCA-bench it raises Haiku from 6.3% to 35.2% of incident root causes, twice Sonnet's 17.8%, at one seventeenth of Sonnet's cost. The agent wins 75 tasks and loses 2, and answers that name only wrong causes fall from 28.5% to 7.1%. On CRMArena-Pro it raises Haiku from 57.6% to 76.7% of questions, above Sonnet's 67.6%, at one fourteenth of Sonnet's cost. It also halves Haiku's own cost, because the agent takes 3.9 steps instead of 5.1 and reads 27.6k input tokens instead of 55.7k. The gain holds under exact match (51.8% to 70.9%) and in mean token F1 (0.569 to 0.757). On ALFWorld it raises Haiku from 79.1% to 97.8% of games (26 won and 1 lost), solving 131 of 134 in 12.8 steps instead of 22.2, at 0.31¢ a game against 0.62¢. Sonnet solves 133, so Enterprise Brain closes 93% of the gap at one tenth of Sonnet's cost of 3.0¢ a game.

	Haiku alone	Skills only	Skills + advice	Advice − skills [95% CI], W/L
ORCA-bench	6.3	10.3	**35.2**	+24.9 [+16.3, +34.0], 69 / 6
DolphinBench	47.0	62.9	**72.3**	+9.4 [+5.2, +13.5], 77 / 32
ALFWorld	79.1	–	**97.8**	+18.7 [+9.6, +28.7], 26 / 1[a]

Table 2: What each memory adds, with the same agent, tasks and tools. *Skills only* is Enterprise Brain serving the skills Opus extracted from the documents. On DolphinBench the full system of Fig. 1 also searches the temporal memory. [a]Against Haiku alone, since ALFWorld has no knowledge base.

4.2 What each memory adds

Skills that the same stronger model extracted from the documents help where the work is documented (DolphinBench, 47.0% to 62.9%) and little where it is not (ORCA-bench, 6.3% to 10.3%). Advice learned from graded traces adds +24.9 points on top of them on ORCA-bench and +9.4 on DolphinBench (Table 2): what the outcomes taught, beyond what the teacher could write from the documents alone. Without runs, the teacher's advice adds little where the domain is private. In one session, Opus wrote each situation's advice from its tasks alone, and Haiku answered with that advice, with the learned advice and with none. On the 295 CRMArena-Pro questions that receive advice, the no-runs advice scores 48.1% and no advice 45.1%, a difference of +3.1 points. The learned advice scores 63.7%, +15.6 points above the no-runs advice. ALFWorld is public and the teacher already knows its mechanics, so there the teacher alone is enough: both kinds of advice solve all 58 advised games, and Haiku without advice solves 49. Other ways of learning from the same runs do worse, or as well at a higher cost. We reimplemented two from their papers. Each has the same teacher learn from the 660 graded runs behind CRMArena-Pro's advice, into one playbook of up to 16,000 characters, which every question receives whole through the same tool. One summarizes the runs, with their rewards and reasons, into a single prompt. ACE keeps an itemized playbook whose entries carry counts of the runs they helped and harmed [Zhang et al., 2025]. On 330 held-out questions, with Haiku run as in the no-runs comparison, no advice scores 45.2%, ACE's playbook 51.5%, the summary 60.9% and the learned advice 63.3%. The learned advice beats ACE by +11.8 points and matches the summary (+2.4 points) while serving a quarter as much text and costing 0.37¢ a question against the summary's 0.64¢. Learning from graded outcomes carries most of the gain: the summary alone adds +15.8 points over no advice. Two design choices matter. Serving the advice alone beats serving it with its skills: on ORCA-bench's 132 Medium tasks the advice alone scores 55.3%, the advice with its linked skills 41.7%, the skills alone 11.4% and no memory 12.9%. And the outcome carries the goal: advice learned with each task's goal stated in words scores 32.8% on ORCA-bench against 35.2% without it.

4.3 Against generic memory

A generic memory stores what it is given and retrieves what resembles the request. We give one, OpenViking [Volcengine (ByteDance), 2026], the inputs Enterprise Brain has: an index of the same sources, summarised by Haiku as Enterprise Brain's facts are extracted, and the same graded practice runs, each stored as a document with its task, trace, outcome and verdict and summarised by Opus, Enterprise Brain's learner (Fig. 1). The agent searches both; on ALFWorld, whose agent has no tools, the top runs go in the prompt within the advice's budget, and on ORCA-bench a task sees only other days' runs. The runs help the generic memory a little: it answers 61.8% of CRMArena-Pro's questions against 57.6% alone, solves 86.6% of ALFWorld's games against 79.1%, finds 7.9% of ORCA-bench's root causes against 6.3%, and passes 43.7% of DolphinBench's requests against 38.5% with the history as files. From the same runs Enterprise Brain gains 19 to 34 points. Enterprise Brain does better because it learns from the outcomes instead of storing them. A generic memory returns the runs that resemble a task, right or wrong, and leaves the agent to judge which to follow; on ORCA-bench, where most practice runs were graded wrong, retrieving them raises the share of reports naming only wrong causes from 28.5% to 33.2%. Enterprise Brain compares a situation's right and wrong runs and writes down what the right ones did and what the wrong ones missed, so the agent no longer has to judge. On ALFWorld every practice attempt at heating failed: retrieving them solves 7 of 23 heating games, and advice learned from them solves 21. Searching the sources alone changes little: 57.3% on CRMArena-Pro, 9.1% on ORCA-bench and 46.8% on DolphinBench, against 42.0% with the history as files.

5 Advice does not degrade performance

A memory that learns from outcomes should also do no harm where there is little to learn. We test this on two benchmarks. *SWE-ContextBench* [Zhu et al., 2026]: resolve GitHub issues so the hidden tests pass, in repositories whose earlier issues the agent practised on. We score the 217 targets that share no pull request with any practice issue. τ^2*-bench banking* [Barres et al., 2025]: serve simulated bank customers while following 698 policy documents, cross-fitted by task in three folds so no task is served advice learned from its own conversations. Resolving an issue turns on writing a correct patch more than on following a procedure, and on τ^2-bench only 55% of tasks fall in a situation with advice (Table 8).

On SWE-ContextBench, advice learned from 1,006 earlier issues raises Haiku from 27.6% to 30.6% of issues resolved (+3.0 points, Table 3), the only memory whose interval excludes zero. On τ^2-bench, the other memories cost the agent accuracy: searching the policies scores 20.3% (Section 4.3) and skills extracted from them 20.6%, against 27.8% from reading them as files. Advice recovers that loss (+10.3 points

	Haiku alone	Skills only	Haiku + Enterprise Brain	Enterprise Brain − Haiku [95% CI]
SWE-ContextBench	27.6	–	**30.6**	+3.0 [+0.5, +6.0]
τ^2-bench	27.8	20.6	**30.9**	+3.1 [−3.4, +10.0]

Table 3: Advice does not degrade performance. Haiku alone is the benchmark's standard setting (on τ^2-bench, the policies as files). Sonnet 5.5 alone scores 35.5% and 46.7%.

over skills only) and leaves the agent at 30.9%, +3.1 points from where it started. On the tasks it reaches it adds 19.5 points, and on the rest it changes nothing (Table 8). Sonnet stays ahead on both benchmarks.

6 Analysis

Advice is domain knowledge. Each piece of advice in Fig. 5 states something true of one environment. On ORCA-bench, the shop's faults surface as gRPC errors and latency that never reach the HTTP status codes, and the grader expects every flag change of the day, so the agent that reported no incident finds all five faults once it is told where to look. On CRMArena-Pro, a deal in this CRM closes when its contract is signed, so a period is filtered on the signing date, a rule written nowhere in the schema. On DolphinBench, this person's requests are graded on the action taken, so the advice tells the agent to send the email rather than stop at a draft. On ALFWorld, the environment heats an object only while the agent holds it at the microwave. Without advice, the agent put an egg inside first and spent the rest of its 50 steps on rejected commands. With advice it solved 21 of 23 heating games instead of 4. The advice also corrects the organization's own documents where they disagree with what the grader rewarded: the ORCA-bench advice in Fig. 3 warns that the skill extracted from the shop's documentation names the feature flags differently from the deployed keys. A larger model knows more in general, and none of this in particular. Advice is learned from this agent's graded work in this domain, which is why a small model with it can do the domain's work as well as a larger model without it, and why retrieval over the same runs, which matches tasks by wording, does not find it.

Hill climbing and practice. Climbing raises DolphinBench's validation accuracy from 61.7% with skills alone to 75.0% in four rounds (Fig. 6, left). Pooling every person's runs into one store in round 2 gives the largest step, and rounds 3 and 4, which add the agent's own practice with the advice before them, carry it to 75.0%. The kept version, round 4, adds +10.7 points over skills alone on the 410 test requests never looked at during development. More graded practice raises accuracy wherever we varied it. On ORCA-bench, accuracy rises from 10.3% with skills alone to 16.2%, 19.8% and 35.2% as advice learns from 135, 284 and 454 graded runs (middle). On CRMArena-Pro's hardest kinds of question, nine graded runs per kind take the agent from 32.5% to 61.0%, after which accuracy levels off (right).

Where the gain sits. A benchmark's gain splits into *coverage*, the share of tasks served advice, and *effectiveness*, the gain on those tasks:

$$\mathbb{E}_q[\Delta] = \Pr[C]\,\mathbb{E}[\Delta \mid C] + \Pr[\neg C]\,\mathbb{E}[\Delta \mid \neg C], \qquad (2)$$

where C is the event that a task is served advice. A task without advice gets the same context in both arms, so the last term is zero by design. On ORCA-bench, advice reaches 98% of tasks and adds 25.4 points on them, and on CRMArena-Pro it reaches 89% and adds 21.7 (Table 8 in the appendix). On ALFWorld it reaches only the heating, cooling and lamp tasks it was learned for, 43% of games. It takes them from 55.2% to 98.3% and leaves the other 76 games as they were. On τ^2-bench it reaches 55%, adds 19.5 points on those tasks and nothing on the rest, which is why the benchmark-level effect is small. Coverage can be raised on its own: routing more tasks to advice, or learning advice for more situations, raises the gain without better advice.

More practice behind a situation's advice does not make it help more. Let $n(q)$ be the number of graded practice runs in the situation a task's advice came from. Across ORCA-bench's 8 situations and CRMArena-Pro's 20, the rank correlation between $n(q)$ and a situation's gain is 0.00 and -0.04. On CRMArena-Pro, tasks whose advice rests on at most the median 31 runs gain 24.8 points and the rest 15.1, and the sales-cycle advice of Fig. 5, written from four runs (two right, two wrong), takes its 17 questions from 5 right to 15. On ALFWorld every situation's advice rests on 20 runs, and the gain runs from nothing on lamp tasks to 17 more heating games solved out of 23. The number of distinct mistakes the advice names does not predict the gain either (rank correlations -0.15 and -0.20): on ORCA-bench the two checkout situations gain most (+41.4 and +70.0 points) and name as many pitfalls as the two vague-complaint situations, which gain least (+6.8 and +21.2).

	One-time ($)				Per task (¢)		Break-
Benchmark	Practice	Skills	Advice	Routing	Enterprise Brain	Sonnet	even N
ORCA-bench	97.32[a]	23.34	4.59	0.48	2.4	40	≈ 330
DolphinBench	n/m[b]	≈141[c]	n/m[b]	in serving	0.80	11.9	≥ 1,270
CRMArena-Pro	4.18	–	n/m[b]	–	0.33	4.6	≥ 98
ALFWorld	0.86	–	0.44	in serving	0.31	3.0	≈ 48

Table 4: Costs. One-time costs are practice and learning (skills and advice on Opus 5.5), and per-task costs are serving. Break-even N is the one-time cost divided by the per-task saving over Sonnet. [a]The baseline's own trials, reused. [b]Run on a subscription, not metered. [c]At API rates, from 750 calls.

Cost. Learning is cheap next to the practice it learns from (Table 4): on ORCA-bench, extracting skills cost $23 and learning advice $4.60 on Opus, against $97 of practice. Serving adds one short routing call per task, and advice shortens the agent's work: on CRMArena-Pro it halves the cost per question. Against Sonnet, the one-time cost is repaid after 48 tasks

ORCA-bench: an on-call agent

Task. "Shoppers cannot use their cart," reported sometime in the morning today.

Without advice. "The available telemetry shows no incident: cart add-item and get_cart request rates never reached zero." An empty report: wrong.

The advice. "Treating 'no HTTP 5xx' as no incident. Most injected faults are gRPC errors or latency that never show in `http_server_request_duration` status codes."

With it. Names all five injected faults (the cart failure, three payment failures and the product-catalog failure): graded correct.

Learned from 40 graded runs of the situation: 10 right, 30 wrong.

CRMArena-Pro: a CRM analyst

Task. "During the winter of 2023, which agent took the longest average time to move opportunities from start to finish?"

Without advice. Picks the winter's deals by when they were created (`o.CreatedDate`) and names the wrong agent.

The advice. "Filter the period on `c.CompanySignedDate`. The signing date is the close event. Do not filter on `o.CloseDate`."

With it. Filters on the signing date and names the right agent, in two tool calls (one for advice, one query) instead of three.

Sales-cycle questions: 5 of 15 right without advice, 15 of 15 with it.

DolphinBench: a personal assistant

Task. "Email privacy@scaffold.dev the handling instructions for Acme's customer-record copy request received this morning."

Without advice. "I haven't sent anything yet … I don't want to send the wrong ones to privacy." 0 of 3 checks.

The advice. "**Send it.** Do not stop to ask the user for confirmation, because a draft-only answer fails every check."

With it. "I sent the handling instructions to privacy@scaffold.dev with the subject 'Handling instructions: Acme customer-record copy request.'" 3 of 3 checks.

From advice on replying to work correspondence, learned from six graded runs (2 right, 4 wrong) of other requests.

Figure 5: Advice is domain-specific. Three scored tasks, what the agent (Claude Haiku 5.5) did without advice, the advice it was served, and what it did with it. Each piece of advice was learned from other tasks' graded runs in the same environment, and each states something true only of that environment: how this shop's faults surface, which date closes a deal in this CRM, what this person's requests require.

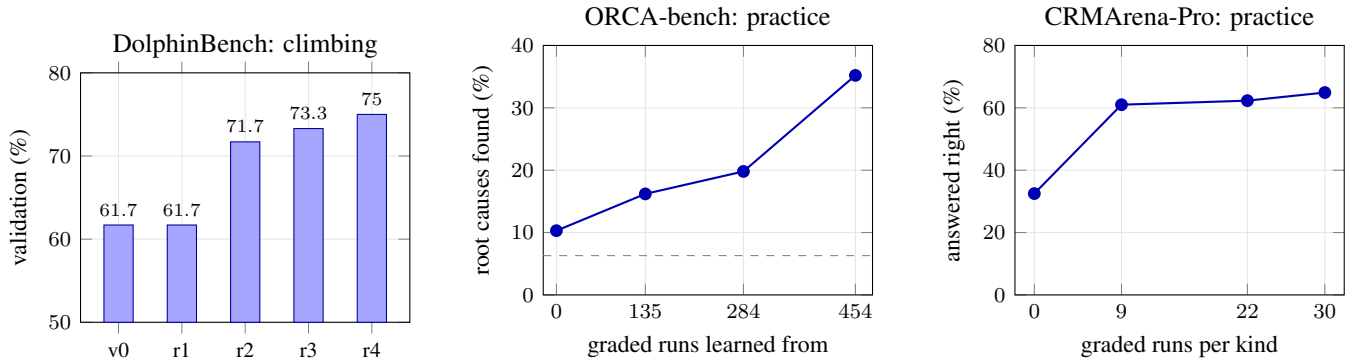


Figure 6: Hill climbing and practice. Left: validation accuracy of each version of DolphinBench's advice (60 requests). Round 1 learned one store per person, round 2 pooled every person's runs, and rounds 3 and 4 added the agent's practice with the advice before them. Middle: ORCA-bench, all 253 tasks, each point scored only on days its advice never saw (the 135- and 284-run points from a later session). The dashed line is Haiku alone. Right: CRMArena-Pro, the 77 questions of the 13 kinds the agent got most wrong, with advice learned separately at each point.

on ALFWorld and 330 on ORCA-bench, and after at least 98 on CRMArena-Pro and 1,270 on DolphinBench, whose learning costs were partly unmetered. Relearning costs less than the first build. In deployment the practice runs are the agent's own graded work, so a relearn costs only the learning: $4.59 on ORCA-bench, which the 38¢ each task saves against Sonnet repays within 13 tasks. If the practice had to be bought again at $97, a relearn would pay for itself every 260 tasks.

7 Limitations

The main limitation is the data. Each benchmark gives Enterprise Brain a small, fixed set of graded practice runs, from 120 on ALFWorld to 660 on CRMArena-Pro, far fewer than an organization's agents produce over months of work. With so little practice, some kinds of work never get advice, since advice is written only for a situation with at least two wrong runs: on τ^2-bench only 55% of tasks fall in a situation with advice, and on ALFWorld every heating attempt failed, so that advice was written from failures alone. The results therefore show what advice learns from a few hundred graded runs, not what it learns as an organization's runs accumulate. Accuracy was still rising with more practice where we measured it (Fig. 6), so these numbers likely understate what continued learning would reach. The benchmarks also fix their tasks, so they cannot show advice following an organization's work as it changes.

8 Conclusion

We introduced Enterprise Brain, a memory for an organization's agents that learns how the organization's work is done from the agents' own graded work. It keeps a temporal memory of what the organization's sources say and a procedural memory of how the work is done: skills extracted from the organization's documents, and advice that a stronger model writes for each kind of work from the runs that went right and wrong, improved by hill climbing as the agent practises with it. Because the advice comes from the organization's own outcomes, it holds what no general model knows: the conven-

tions, procedures and pitfalls of this one domain. That lets a small model do work that otherwise needs a larger one, at a fraction of the cost, and where there is little to learn the advice does no harm. A generic memory given the same runs gains only a few points, because it stores them without learning from their outcomes. As an organization's graded work accumulates, its memory can keep learning from it.

References

Lakshya A. Agrawal et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Emmanuel Aboah Boateng, Cassiano O. Becker, Nabiha Asghar, et al. Concept distillation from strong to weak models via hypotheses-to-theories prompting. In *Proceedings of NAACL*, 2025. arXiv:2408.09365.

Yuzheng Cai, Siqi Cai, Yuchen Shi, et al. Training-free group relative policy optimization, 2025.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Gaodan Fang, Vatche Isahagian, K. R. Jayaram, et al. Trajectory-informed memory generation for self-improving agent systems, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, et al. Memp: Exploring agent procedural memory, 2025.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2403.08978.

Albert Gong, Kyuseong Choi, Abhineet Agarwal, Jason Schechner, Ryan Huang, Raj Agrawal, Anish Agarwal, and Raaz Dwivedi. ORCA-bench: How ready are language model agents for oncall?, 2026. arXiv:2607.28545.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, et al. LEGOMem: Modular procedural memory for multi-agent LLM systems for workflow automation, 2025.

Kung-Hsiang Huang et al. CRMArena-Pro: Holistic assessment of LLM agents across diverse business scenarios and interactions. *arXiv preprint arXiv:2505.18878*, 2025.

Siru Ouyang et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Ningxin Pan, Hanyu Li, Yeqing Shen, et al. The world won't wait: Evaluating and teaching agents under a live clock, 2026a.

Qianjun Pan, Yutao Yang, Junsong Li, et al. Anything2Skill: Compiling external knowledge into reusable skills for agents, 2026b.

Wenbo Pan, Shujie Liu, Xiangyang Zhou, Shiwei Zhang, Wanlu Shi, Mirror Xu, and Xiaohua Jia. M*: Every task deserves its own memory harness. *arXiv preprint arXiv:2604.11811*, 2026c.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Soumil Rathi, Deshraj Yadav, and Taranjeet Singh. Dolphin-Bench: Mapping the Pareto frontier of agent memory, 2026. arXiv:2609.24971.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. arXiv:2010.03768.

Yiqun Sun, Pengfei Wei, and Lawrence B. Hsieh. Corpus2Skill: Distilling enterprise knowledge into navigable agent skills for QA and RAG, 2026. Findings of EMNLP 2026; arXiv:2604.14572.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Volcengine (ByteDance). OpenViking: Self-evolving context database for AI agents. `https://github.com/volcengine/OpenViking`, 2026. Version 0.4.23.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Ruochen Wang, Sohyun An, Minhao Cheng, Tianyi Zhou, Sung Ju Hwang, and Cho-Jui Hsieh. One prompt is not enough: Automated construction of a mixture-of-expert prompts. In *Proceedings of ICML*, 2024a. arXiv:2407.00256.

Yu Wang and Xi Chen. MIRIX: Multi-agent memory system for LLM-based agents, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, et al. SkillOpt: Executive strategy for self-evolving agent skills, 2026.

Shuo Yuan, Dacheng Li, and Joseph E. Gonzalez. D-Skill: Distilling teacher knowledge through agent skills, 2026.

Qizheng Zhang et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Jiayuan Zhu, Junde Wu, Minhao Hu, Shengda Zhu, Jiazhen Pan, Weixiang Shen, Yijun Yang, Fenglin Liu, Jianye Hao, Yueming Jin, Qirong Ho, and Min Xu. SWE context bench: A benchmark for context learning in coding, 2026. arXiv:2602.08316.

A Method details

This appendix gives the details of Enterprise Brain that the experiments depend on. For implementation details, contact the AmpUp team.

A.1 Temporal memory

A scope's schema declares the types of entity it holds, their attributes, and how each attribute's values combine and age. A small model reads each source with the schema and writes typed, dated facts, each citing the exact part of the source it comes from. What simple statements add up to, such as whether a commitment was kept, is derived per subject from every source about it.

A.2 Procedural memory: skills

Each skill is also written as a step graph (Fig. 7) whose steps each name what they need and what they give, with edge cases as steps of their own. Every model reply is cached by its request, so building the same library again makes no paid call and yields the same skills.

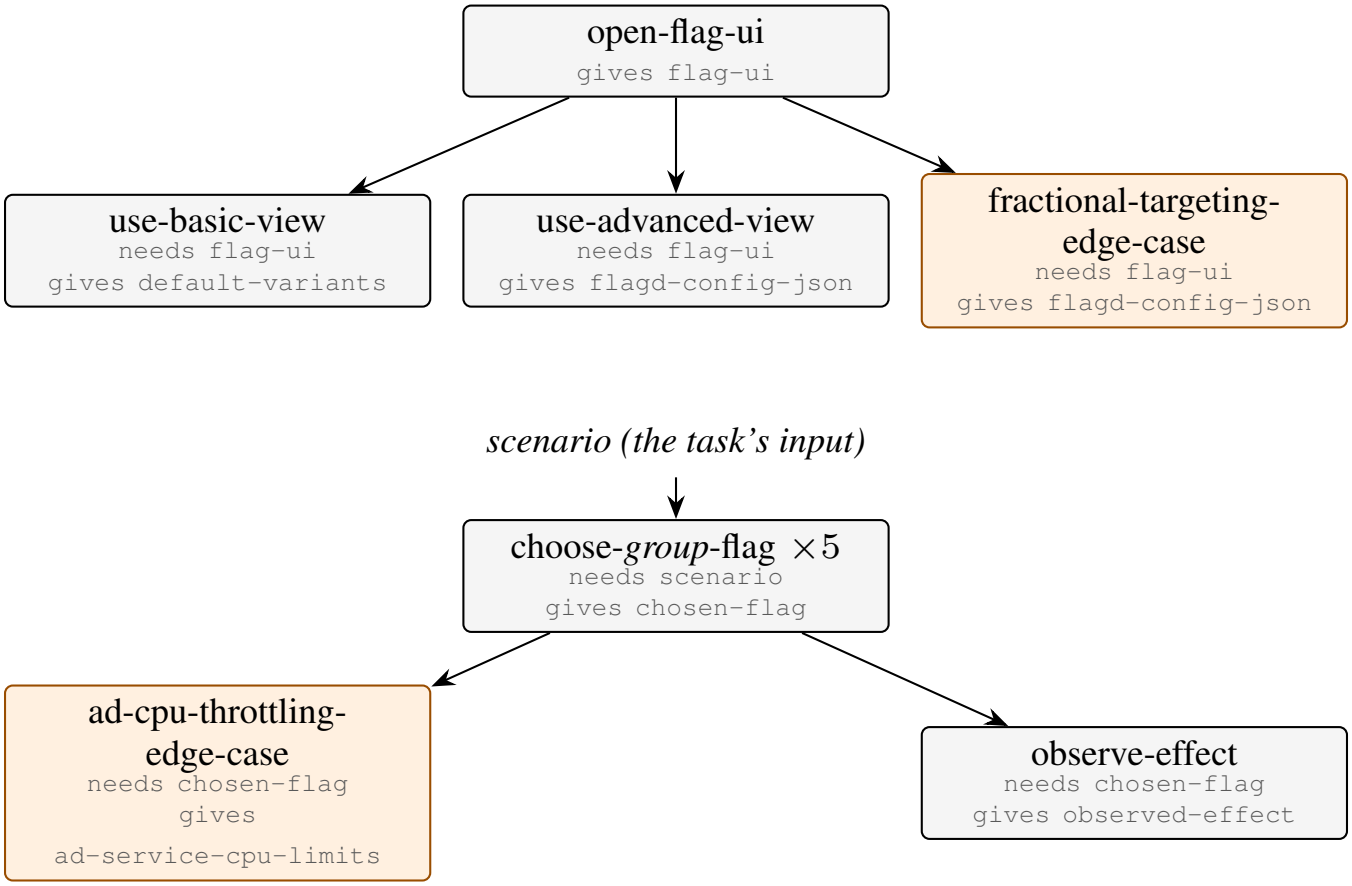


Figure 7: Part of a skill's step graph, extracted on ORCA-bench from the shop's feature-flag documentation (13 steps; the five flag-choosing steps are drawn as one). Each step names what it needs and gives, and an arrow joins a step to the steps that need what it gives. Edge cases (orange) are steps of their own. The skill cites its source document.

A.3 Serving

Above 60 situations, the router is shown only the 60 closest to the task, ranked by BM25 and embedding similarity, which adds no model call. The engine our experiments ran on had no shortlist (Appendix B).

A.4 Written goals

Earlier versions of Enterprise Brain were built around written *goals*, statements of what the organization wants from a kind of work, which steered extraction, built a skill library per goal, and were shown to the learner. Advice learned under each task's own goal scored no better than advice learned under a neutral one on any benchmark at full size (32.8% vs. 35.2% on ORCA-bench, 30.0% vs. 30.6% on SWE-ContextBench, 28.2% vs. 30.9% on τ^2-bench), and a skill library built for the goal served advice no better than one built for a neutral goal. Enterprise Brain therefore takes the goal from the outcome: the schema says what is worth keeping, and the rewards with their reasons say what success means.

A.5 A worked example

Table 5 follows one real example through each step.

Step	What goes in	What comes out
Extract skills (§3.2) ORCA-bench	The shop's documentation, including `docs/demo/feature-flags/_index.md`.	A skill, *Toggle the demo's feature flags to simulate failure scenarios*, with 13 steps and its step graph (Fig. 7): each step names what it needs and gives, edge cases are steps of their own, and the skill cites the file.
Map (§3.3) ORCA-bench	The tasks of one fold's graded practice runs, without their outcomes.	One category (*root cause analysis of reported storefront incidents*) and five situations, each linked to two skills; four have enough wrong runs for advice.
Learn advice ORCA-bench	The 40 runs of *diagnose a vague site-wide complaint* (10 right, 30 wrong) and its two linked skills.	Advice with seven steps and pitfalls that cite the runs that made each mistake (Fig. 3); one pitfall corrects the linked skill, whose flag names differ from the deployed ones.
Learn advice CRMArena-Pro	Graded runs on sales-cycle questions, many of which filtered the period on the wrong date.	"Join the opportunity to its contract … **Filter the period on `c.CompanySignedDate`.** The signing date is the close event. Do not filter on `o.CloseDate`."
Climb (§3.4) DolphinBench	Version 0's 60 graded practice runs; then each kept version's own practice runs, pooled.	Advice for 4, 7, 13, then 14 situations; validation 61.7% → 71.7% → 73.3% → 75.0%. Served advice says: "**Send it.** Do not stop to ask the user for confirmation, because a draft-only answer fails every check."

Table 5: One real example through each step of Enterprise Brain, from what goes in to what comes out. Quotes are from the memory as built and served in our experiments.

B Setup details

Model	Job
Claude Haiku 5.5	the agent on every benchmark; routes tasks to advice; OpenViking's summaries
Claude Opus 5.5	compiles skills and step graphs; sorts runs into situations, links them, learns advice
Claude Sonnet 5.5	the larger model, with no memory
GPT-6.1 Sol	the model call inside ORCA-bench's grader
GPT-6 Sol	the model call inside DolphinBench's grader
GPT-5.2	τ^2-bench's simulated customer

Table 6: Every model in this study and its job. Embeddings: Enterprise Brain's search used bge-small-en-v1.5, run locally, on ORCA-bench, SWE-ContextBench and τ^2-bench, and text-embedding-3-small through OpenRouter on DolphinBench; OpenViking used text-embedding-3-small.

Practice and test. ORCA-bench's 454 practice runs, from the shop's six days, are dealt by day into two folds of 238 and 216 runs. Each task is served only from the fold that does not hold its day, so its advice rests on about half the runs. On SWE-ContextBench the agent practised on 1,006 earlier issues from the same repositories. We score the 217 targets of 357 that share no pull request with any practice issue (on all 357, advice adds +2.0 points). On τ^2-bench the advice is learned from 592 graded practice conversations (74 tasks, 8 each). Each fold's store learns from the other two folds' tasks, so each conversation sits in two of the three stores.

Engine versions. In every run, extraction, skill extraction and learning were done before the run and only routing was live. Each benchmark ran on a fixed build of Enterprise Brain, recorded with its results. The build used on ORCA-bench, SWE-ContextBench and τ^2-bench differs from Section 3 in three ways. Its router saw every line, with no shortlist, and took only an exact key: when the routing model shortened a key, our harness repaired it to the one line it named, affecting up to 16 of a benchmark's tasks. Its learner prompts carried a line for the scope's goal, filled with a neutral statement in every arm we report. And the current engine adds advice sections the experiments did not use.

Statistics. On ORCA-bench, clustering by incident or by day instead of by question gives narrower intervals, and every one of the six days gains. We report the widest. Scores also vary between sessions: the same two sets of advice, byte for byte, scored 18.9% and 34.8% on the same 132 ORCA-bench tasks in one session and 24.2% and 31.1% in another, and the τ^2-bench files baseline moved 6.8 points between sessions. The goal ablations were specified, with predictions and refutation criteria, before they ran. The rest of the study is exploratory, and we do not correct for multiple comparisons.

C Settings

Table 7 lists every setting of the learning loop and the value used in our experiments. All are the engine's defaults unless marked.

Step	Setting	Value
Extraction	model	Claude Haiku 5.5
Skills	model	Claude Opus 5.5
	batch	12 sources or 60,000 characters, oldest first
Situation map	model	Claude Opus 5.5
	batch	60 tasks, in arrival order; one re-sort of misfiled tasks
	links	at most 3 compiled skills per situation
Advice	model	Claude Opus 5.5
	first advice	at least $m = 2$ wrong runs
	window	at most 40 runs: newest wrong, newest right, at least 10 right
	rewrite	at least 5 new runs, 2 new wrong runs, or changed skills
Serving	router	Claude Haiku 5.5
	shortlist	60 situations (current engine; none in our runs)
	no advice yet	up to 2 linked skills
Climbing	split	10/10/80 train/validation/test, by unit, seed 20261016
	stop	patience 2 rounds; at most 8 versions

Table 7: Settings of Enterprise Brain's learning loop.

D Coverage and effectiveness

Table 8 splits each benchmark's gain into the share of tasks served advice and the gain on them (Eq. (2)).

			Tasks served advice		Tasks without	Median $n(q)$
Benchmark	Comparison	Coverage	Before $\rightarrow$ after (%)	Diff [95% CI], W/L	Diff [95% CI]	(IQR)
ORCA-bench	advice − skills	98.0%	10.5 $\rightarrow$ 35.9	+25.4 [+16.9, +34.8], 69/6	n=5: 0.0	74 (65–77)
CRMArena-Pro	advice − SQL only	89.4%	56.3 $\rightarrow$ 78.0	+21.7 [+16.6, +26.8], 68/4	−2.9 [−8.6, 0.0]	31 (30–34)[a]
DolphinBench	advice − skills	91.7%	64.1 $\rightarrow$ 73.6	+9.5 [+5.2, +14.1], 72/30	+7.5 [−5.0, +20.0]	15 (9–24)[b]
ALFWorld	advice − none	43.3%	55.2 $\rightarrow$ 98.3	+43.1 [+26.8, +59.3], 25/0	+0.0 [−3.8, +4.0]	20 (20–20)[c]
SWE-ContextBench	advice − none	88.9%	24.9 $\rightarrow$ 28.0	+3.1 [+0.3, +6.0], 17/7	+2.1 [0.0, +6.2]	141 (30–392)
τ^2-bench banking	advice − skills	54.6%	20.1 $\rightarrow$ 39.6	+19.5 [+10.1, +29.6], 19/3	−0.8 [−7.6, +6.8]	16 (8–40)

Table 8: Coverage and effectiveness (Eq. (2)), against the same system serving no advice. $n(q)$: graded practice runs in the situation a task's advice came from. [a]From the run's learned store, archived outside the repository. [b]From a local copy of the store. [c]From the run's frozen store.

E Learned advice in full

The advice of Fig. 3, as learned for one fold of ORCA-bench, unabridged. Run handles (R1–R40) refer to the graded runs it was written from.

```
advice/root-cause-analysis-of-reported-storefront-incidents/diagnose-a-vague-site-wide-complaint (version of 2026-10-20)

# Advice: Diagnose a vague "users are reporting site issues" report on the OpenTelemetry demo storefront

## When
Use this when the report names no feature or symptom, only something like "users are reporting site issues". The report
     gives a time, a window, or a part of the day. You have the Grafana API (Prometheus, OpenSearch logs, Jaeger) and
     the demo source at `/app/opentelemetry-demo`.

In every one of these runs, the expected root causes were **runtime flagd feature-flag changes, each with a time**, or
     **no root cause at all** (a healthy control). No other outcome was rewarded. Chronic errors, latency artifacts and
     infrastructure noise were never the answer.

## Steps
1. **Pin the clock.**
  - Convert the report time or window, and the "current time", to UTC (ET is UTC-4).
  - The telemetry snapshot ends at about the current time. Confirm this by finding the last log or span timestamp, so
       you don't read an empty tail as recovery.
  - If the report is a single point equal to the current time (for example "reported at 9:00 AM", now 09:00), treat the
       task as a likely healthy control and go to step 7.
2. **Get the flag catalogue from the source.**
  - Read `src/flagd/demo.flagd.json` for the real flag keys. These keys are what you will name in the report.
  - The keys seen in these runs were adFailure, adManualGc, adHighCpu, cartFailure, paymentFailure, paymentUnreachable,
       productCatalogFailure, kafkaQueueProblems, imageSlowLoad, and others.
  - Every defaultVariant is "off". Expect that: the faults are switched on at runtime, so the file never shows them as
       active.
3. **Scan the whole day for flag-driven symptoms, from 00:00 UTC of the report day to the current time.** Do this for
     every service, not just the report window. Look for each flag's signature:
  - **adFailure:** ad WARN logs "GetAds Failed with status UNAVAILABLE". Frontend `/api/data` 500s. About 1 in 10
       GetAds calls fail.
```

```
  - **adManualGc:** ad logs saying "Feature Flag adManualGc enabled". Ad p95 latency rises from a few ms to seconds,
      and frontend latency follows.
  - **adHighCpu:** ad CPU rises.
  - **productCatalogFailure:** product-catalog GetProduct errors, all on one product id.
  - **paymentFailure:** payment WARN "Payment request failed. Invalid token. app.loyalty.level=gold". Checkout
      PlaceOrder errors.
  - **paymentUnreachable:** checkout client-side Charge errors while payment's server side shows no errors.
  - **cartFailure:** cart EmptyCart errors or 10-15 s latency. Cart logs "Wasn't able to connect to redis". That error
      comes from the hard-coded `badhost:1234` store behind the flag, not from a real Valkey outage.
  - **kafkaQueueProblems:** Kafka lag. Accept it only if it shows a clear onset.

  Data sources to use:
  - **Span metrics:** `sum by (service_name, span_name, status_code) (increase(traces_span_metrics_calls_total[10m]))`,
      plus `histogram_quantile` latency by service and span.
  - **OpenSearch logs:** severity and the service field (`resource.service.name`; use its `.keyword` subfield for
      aggregations).
  - **Flag evaluations:** `feature_flag_evaluation_*` broken down by key and variant, where present.
4. **Time each change.**
  - Use 1-minute histograms to find each symptom's first occurrence.
  - The flag change happens a few minutes before the first symptom. Typical gaps were 3-5 minutes for errors. For a
      10-minute bin, the change is at or just before the bin start.
  - When an error ratio steps up again later (for example paymentFailure going from 10% to a higher percentage), list
      that step as a separate flag change of the same flag.
5. **Report every flag change from the day up to the current time.** This includes changes before the report window and
    changes after it.
  - In these runs, the grader expected all of them. Example: a report window of 14:48-15:48 still expected changes at
      18:39, 20:09, 20:39 and 21:09 UTC, because the current time was later.
  - Give each flag key and its change time in the Summary, Timeline and 5 Whys. Say what the flag does (cite the code
      path).
6. **Write `/app/report.md`** with the Summary, Timeline, 5 Whys and Remediation sections.
  - `/app` is not writable for `sed -i`. Write the edited text to `/tmp` first, then `cat` it into the report.
7. **Healthy-control check, used when the report time equals the current time.**
  - Look back about an hour from the report.
  - If no new flag-driven onset falls in that hour, leave `/app/report.md` empty, even if flags were flipped earlier
      that day.
  - Before you do, confirm the baseline is flat on error spans, error logs and status codes, so the empty report rests
      on data and not on failed queries.

## Pitfalls
- **Treating "no HTTP 5xx" as no incident.** Most injected faults are gRPC errors or latency that never show in
    `http_server_request_duration` status codes (R1, R9, R11-R14, R20, R28, R29). Check span-metric status codes and
    service logs.
- **Concluding "no incident" from unreachable telemetry.** Empty index lists, HTTP 000 responses and NaN results are
    transport failures or wrong field names. Retry with `-w '%{http_code}'`, write output to a file, and try both the
    direct hosts and the Grafana proxy (R1, R6, R7, R29).
- **Mistaking known red herrings for the root cause.** These were never the answer:
  - Accounting's chronic EF Core "Order parsing failed / EntityState Detached" errors (R3, R16, R23, R27).
  - Accounting's 15 s `order-consumed` p95: the span wraps an idle Kafka Consume (R4).
  - flagd's ~500-600 ms p95: these are long-lived EventStream streams (R5).
  - Load-generator errors.
  - Histogram 10 s / 15 s ceilings at low traffic.
- **Misreading cart's "Wasn't able to connect to redis".** It means cartFailure, not a Valkey outage (R2, R11).
    Likewise, checkout→payment failures can be paymentFailure, paymentUnreachable, or both (R17, R21, R24).
- **Stopping at one flag.** Runs that named only the flag inside the report window still failed when other flags
    changed that day (R8, R10, R17, R18, R19, R22, R25). Always sweep the whole day up to the current time.
- **Vague causes and drifting times.** "Ad dependency failure, mechanism unconfirmed" scores near zero (R26). So does a
    time far from the change, such as the end of a 30-minute bin or a time 40 minutes after the first symptom. Name
    the flag key and give a time just before the onset.
- **Using the flag names from the feature-flag procedure.** In the procedure "Toggle the demo's feature flags", step
    choose-*-flag uses names like adServiceFailure, cartServiceFailure and paymentServiceFailure, and its "Checkout:
    paymentServiceUnreachable". The deployed keys are adFailure, cartFailure, paymentFailure and paymentUnreachable.
    Name flags from `demo.flagd.json`.
- **Raising a false alarm on a report-at-now control.** On a control, asserting earlier flag changes as the root cause
    failed (R30). R40 got credit for describing an earlier ad slowdown with no confirmed flag; leaving the report
    empty is the safer choice.

## Evidence
- **Runs:** 40 in total, 10 right (R31-R40) and 30 wrong.
- **What the procedure explains:** all 30 wrong runs.
  - 11 wrongly concluded no incident.
  - 18 named the wrong cause or only some of the flags.
  - 1 raised a false alarm on a control.
- **Right runs:** 8 controls left the report empty. R31 and R40 wrote reports on controls but named no flag cause and
    were still credited. R36 named both flags of its day.
- **Grading:** 18 rewards cite model judge scores. The rest look rule-checked (no-report / no-incident verdicts). None
    was shown to be given by a person.
```

F A situation map

The routing index of the same fold: one line per situation, with the skills it links.

```
advice/index (version of 2026-10-20)
```

```
# Advice index

One line per kind of task the memory advises on, by its key (in backticks), with the procedures it serves.

## Root cause analysis of reported storefront incidents: Find the root cause events, and when they happened, behind a
     shopper-facing issue in the demo shop. Use telemetry and source code. Turn the report time into an investigation
     window, remember the incident may have started earlier, and allow for several root causes or none.
- `root-cause-analysis-of-reported-storefront-incidents/diagnose-a-vague-site-wide-complaint`: Diagnose a vague
     site-wide complaint: The report names no specific feature or symptom, e.g. 'users are reporting site issues'.
     (procedures: neutral/demo-feature-flag-scenarios, neutral/demo-architecture-service-map)
- `root-cause-analysis-of-reported-storefront-incidents/diagnose-ad-or-promotion-banner-problems`: Diagnose ad or
     promotion banner problems: The report is about ads, ad placement or promotion banners being blank, delayed or
     broken. (procedures: neutral/ad-service-getads-flags-and-gc, neutral/demo-feature-flag-scenarios)
- `root-cause-analysis-of-reported-storefront-incidents/diagnose-checkout-order-or-payment-failures`: Diagnose
     checkout, order or payment failures: The report is about placing orders, checkout errors or timeouts on POST
     /api/checkout, order flow, payment, or buying trouble. (procedures: neutral/demo-feature-flag-scenarios,
     neutral/demo-architecture-service-map)
- `root-cause-analysis-of-reported-storefront-incidents/diagnose-product-recommendation-errors`: Diagnose product
     recommendation errors: The report is about recommendation sections failing or returning errors (not just slow
     images). (no note yet) (procedures: neutral/recommendation-service-cache-failure-flag,
     neutral/demo-feature-flag-scenarios)
- `root-cause-analysis-of-reported-storefront-incidents/diagnose-slow-pages-or-slow-loading-images`: Diagnose slow
     pages or slow-loading images: The report is about a sluggish storefront, a slow or troubled homepage, or product
     images loading late or having issues. (procedures: neutral/frontend-product-image-slow-load-flag,
     neutral/demo-feature-flag-scenarios)
```

G More learned advice

Figures 8 to 10 show advice learned on three more benchmarks, excerpted as in Fig. 3. Each was served on the held-out tasks of its situation, and each caption gives how Haiku did on those tasks with and without it.

Advice: Heat an object, then put it in or on a target receptacle

Situation: moving household objects to a place → heat, then place an object. Learned from 20 graded runs, all wrong. The advice opens: "No successful source runs: proposed repairs in this note are unvalidated hypotheses."

When

The goal reads "put a hot <object> in <receptacle>" or "heat some <object> and put it in <receptacle>". There is a hard cap of 50 decisions.

Steps

1. **Find the object.** Use `go to <place>` before any `open` or `take`: actions aimed at a place you are not at return "Nothing happens."
2. **Hold it.** An object sitting inside the microwave is not held and cannot be heated.
3. **Go to the microwave.** Do not detour to stoveburners, pans, pots or the toaster.
4. **Heat it in hand.** On arrival, issue `heat <obj> with microwave 1`. Do not move the object into the microwave first.
5. **Place it** only after seeing "You heat the <obj> using the microwave 1."

Pitfalls

- **Loading the microwave, then heating.** The object is then in the microwave, not in hand, and heating fails. Every run that loaded the microwave this way failed.
- **Trying other appliances.** Spending 10–30 decisions on stoveburners, pans, pots, the toaster or `use <appliance>`. None of these ever produced a heat, and they exhausted the 50-step budget.
- **Placing without a confirmed heat.** Placing the object without a "You heat…" message, then taking it back out to retry. Unheated placement does not win, and the shuffling burns steps.
- **Inventing verbs** such as `cook X with toaster` or `heat X with pan`. They are not valid and waste decisions.

Figure 8: Advice learned on ALFWorld (an excerpt). Every practice attempt at heating failed, so the learner marked its repair as unvalidated. The repair held: keeping the object in hand while heating raised Haiku from 4 to 21 of the 23 held-out heating games.

Advice: Find the month with an unusual spike in cases for a named product

Situation: finding trends in cases and sales discussions → find a month with an unusual spike in cases. Learned from 31 graded runs: 12 right, 19 wrong.

When

The question names a product and a window ending today ("past N months", "last three quarters"), asks whether any month had significantly more cases than the others, and wants only the month name.

Steps

1. **Link cases to the product only through the order item**: `"Case".OrderItemId__c = OrderItem.Id` with `OrderItem.Product2Id`. Do not add cases matched by subject, description or issue.
2. **Compute the window from today's exact date.** A quarter counts as 3 months. Cut off at the exact day, not the first of the month.
3. **Count cases per month** with `substr(CreatedDate,1,7)`, not `strftime`.
4. **Answer the window's maximum month**, even when it has only 1 case and every other month has 0. If no cases fall in the window, answer None.
5. **Return only the full month name**, with no year.

Pitfalls

- **Answering None because the peak is "only 1 case".** Every wrong run found the correct lone month or peak month and then declined to answer. (19 runs)
- **Using `strftime` on CreatedDate.** This CRM's date format makes it return NULL, which collapses every month into one group. (5 runs)
- **Widening the case set.** Adding cases matched by issue or text changes the counts.
- **Ignoring the window.** Counting all history, or rounding the start to a whole month or year, creates false ties or different peaks.

Evidence

Explains all 31 runs: each of the 19 wrong runs had the expected month as the window's only or peak month but answered None, and the 12 right runs returned that month, or None when the window held no cases.

Figure 9: Advice learned on CRMArena-Pro (an excerpt). The grader counts a single case in an otherwise empty window as a spike, and 19 of the 31 practice runs found that month and answered None. With this advice Haiku answers 15 of the 15 held-out spike questions, against 5 without.

Advice: Review a proposed change against standing team decisions, and record the verdict on the PR

Situation: reviews of issues, PRs and docs → review a technical change against standards. Learned from 12 graded runs, 6 right and 6 wrong.

When

A task asks you to review or comment on a specific PR or proposed change: a copy or UI change, a config knob, or a schema or label contract. The verdict is graded on the write action you take against the PR, not on your chat answer.

Steps

1. **Fetch the PR** with `get_pr`, using the PR id from the task. Missing diff text is not a reason to skip the review.
2. **Find the governing precedent in memory.** Search with the knob, label or screen name plus words like incident, regression, rollback and contract, and open the source episodes. Collect the original decision and its date, later incidents that reaffirmed it, and the concrete numbers.
3. **Compare the PR to the precedent point by point.** Where it reverses an accepted contract or repeats a past incident's cause, the decision is `request_changes`.
4. **State the required behavior explicitly**: the allowed and rejected values, each prior incident with its date and fix, and the evidence and staged rollout required before acceptance.
5. **Submit** `review_pr(pr_id, decision="request_changes", body=...)`. Do not approve, and do not just comment.

Pitfalls

- **Answering in chat without calling `review_pr` or `post_pr_comment`.** This fails every check, even when the analysis is right. Never stop to ask "want me to submit?". (5 runs)
- **Recommending "hold the PR / ship nothing" for copy reviews.** State which copy is kept, and that the specific new elements are not shipped. (1 run)
- **Vague findings** ("conflicts with contract"). The checks need the concrete values, incident dates, interval numbers and the canary step.
- **Searching only one memory query.** This misses reaffirming incidents and edge-case decisions, such as whitespace-padded values being rejected rather than trimmed.

Evidence

All six right runs called `get_pr`, then `review_pr` with `request_changes` and a body citing the precedent. Five wrong runs never submitted the write action; the sixth is only partly explained.

Figure 10: Advice learned on DolphinBench (an excerpt). It joins what the team decided, found in memory, with what this person's reviews are graded on: a submitted review that requests changes and cites the precedent. On the 39 held-out requests it was served, Haiku passes 34 with it and 14 with skills alone.