

AI Memories at Briefcase

Leo Strijbos

August 2026

1 Introduction to the problem

Here at Briefcase, our north star metric is our autopublish rate: the share of transactions that reach the ledger without a human editing them first. A high autopublish rate means customers are accepting our extraction and classification as it stands, which is the whole product working as intended.

The biggest blocker for this is personal preferences: a bookkeeper wants line item descriptions formatted their way. We extract them our way, correctly, and they edit them. Every edit teaches our autopilot that it can't publish that kind of transaction without approval, so the transaction lands in a review pile it doesn't need to be in.

That's why we're shipping custom AI memories.

2 What are AI memories?

AI memories allow Briefcase to learn from the corrections users make and apply them to due date conventions, document reference preferences, and cosmetic formatting.

A memory is a learned rule about how a particular party wants a particular field filled. It has 4 parts:

1. Scope: who it belongs/applies to
2. Target field: what fields it affects
3. Condition (optional): when is this memory applied?
4. Action: what to do

Scope and target field together define a *cell*, which is a question of the form "how does this client want their due dates set?"

We want to be able to detect such preferences based on past user behaviour. Briefcase's philosophy dictates that this should be invisible to end-users, so adding, modifying and removing memories should happen automatically.

Two fields are mined automatically today: due dates and document references. A third kind of memory, free-text guidance injected into a prompt, is written by hand for customers with a requirement we cannot yet express as a rule. It is the escape hatch, and it is deliberately held to different standards from everything below.

3 Technical challenges

There are a couple of technical challenges with AI memories:

1. Patterns show up at different scopes (organisation-wide, client-specific, and client-and-supplier-specific) and at infinitely many possible conditions (upload channel, job site, invoice recipient name, etc.).
2. We need to act with high certainty. Worst case, we get it wrong, the user gets fed up and turns off autopublish.
3. We need to forget and amend. We will overfit and propose memories that aren't true, and we need to drop them when counter-evidence turns up.
4. Applying a memory has to be auditable. We could inject memories into prompts, but then nothing enforces whether the memory was used. We want to apply them deterministically where we can.
5. Verifying a memory has to be deterministic too. That holds for most of Briefcase's AI features: where we use LLMs to propose values, we use deterministic gates to verify them.
6. The pipeline has to notice its own faults. Every part of the system above runs unattended, on a schedule, against a fleet too large to inspect by hand. A silent gate that rejects everything looks exactly like a quiet week.

These are the kinds of problems we're solving at Briefcase. As an engineer, you get to propose how to solve them.

4 Mining candidate memories

Every night, a job walks correction history and proposes candidates. It works cell by cell, from the widest scope inwards: organisation-wide first, then groups of suppliers that behave alike, then individual suppliers, then whatever corrections the earlier passes left unexplained.

4.1 Cells that fit a single rule

The easy case is a cell whose corrections all agree. For example, a client's due dates are always set to 30 days from the invoice date. There is nothing to condition on, so the candidate is proposed unconditionally and moves straight to backtesting.

4.2 The gates

Fitting a convention is the easy part. A candidate then has to pass five deterministic gates in order, and the miner records the score and the bar at each one so we can see how close the near misses were:

1. **Correction count:** at least 10 corrections at organisation scope, 5 at supplier scope.
2. **Convention fit:** some rule in our vocabulary reproduces the corrected values at all.

3. **Consistency:** the lower bound of the fit’s credible interval clears 0.7 at organisation scope, 0.5 at supplier scope. We gate on the lower bound rather than the point estimate so that six corrections out of six does not outrank forty-one out of forty-five.
4. **Editing sessions:** the evidence spans at least two separate sittings. One bookkeeper working through a backlog on a Tuesday afternoon produces thirty corrections that are really one decision, and we measured that 28% of supplier-scope cells were exactly that. Counting sessions rather than corrections is what stops a single afternoon minting a rule.
5. **Support rate:** at least 85% of the documents the rule would have touched are consistent with it, counting the ones nobody corrected.

Group scopes carry three extra guards, because a cohort is an easy way to launder a single supplier’s habit into a general rule: the cohort must not depend on one member, must hold on members held out of the fit, and must beat what its complement would have done.

4.3 Cells that need a condition

More often the corrections disagree with each other. The bookkeeper does one thing for invoices arriving by email and another for invoices photographed by a site manager, and pooled together those look like noise. A cell that fits no single rule is the signal that a condition exists and we have not identified it yet.

4.4 What the deterministic search taught us

Our first attempt enumerated conditions exhaustively. Given a cell that had failed on consistency, the search tried every `field = value` split over a registry of eight recorded features, kept the split that separated the corrections best, and put it through the guards. It was cheap, exhaustive, and every condition it produced was trivially explainable.

Its first production run searched 476 cells, evaluated 4,122 candidate conditions, found 7, and minted 6. We reviewed those 6 against the customers’ actual books the same evening. Three problems, all architectural:

The conditions encoded people, not rules. Two of the six keyed on upload channel, and the values they had learned were the initials of the person doing the uploading. The rule was really “documents handled by this member of staff get tagged with their initials”; the version we minted would break the day a colleague sent something over WhatsApp. The search cannot see this, because the separation it measured was real. Statistically the condition was fine.

The true condition often wasn’t in the vocabulary. For that customer the honest condition was *uploaded by this person*, which was not a registered feature. The search did the only thing available to it and picked the strongest recorded correlate of the truth. No amount of threshold tuning fixes a hypothesis space that does not contain the answer.

A condition could dodge its own counter-evidence. The third memory suffixed references for suppliers in Spain, minted at 0.857 consistency on seven corrections, one thousandth above the bar. Its slice of the customer’s documents contained two acceptances, both autopublished, and we only counted manually approved acceptances as contradictions. So the support gate ran against zero visible counter-evidence and passed. Checking the published ledger afterwards, 3 of 11 Spanish-supplier documents actually carried the suffix.

That last one generalises uncomfortably: autopublished documents contribute no counter-evidence by design, so the more successful autopublish is, the blinder mining becomes.

Two things went right, and both were the containment rather than the search. Every one of the six sat in shadow mode writing nothing, so the blast radius of being wrong was an afternoon of analysis. And the diagnostics we had shipped the day before meant the whole run was explainable from logs within minutes. The split-half validation guard also worked, killing 54 of the 155 cells large enough to test it, which is genuine overfitting caught.

4.5 The LLM proposes, evidence disposes

The replacement keeps the entire deterministic verifier and changes only where hypotheses come from. One analysis call per cell reads the corrections, the acceptances, and the surrounding context, and returns candidate practices with a proposed condition and a causal story for each.

Three properties matter more than the model choice:

Plausibility is judged before the numbers. Each hypothesis is rated high, medium or low on whether there is a story connecting this condition to this action, and that rating is made without sight of the backtest. A low rating is never minted. A high rating buys a slightly stronger prior and nothing else. This is the judgement the deterministic search had nowhere to put: 0.857 over seven documents is not evidence about mechanism, and the operator-initials rules were statistically clean.

Compound edits get decomposed. A fixed menu of whole transformations either bundles two independent practices into one wrong rule or fits nothing at all, and the cells that fit nothing were never even searched. Asking for atomic practices splits them, and it added a second trigger for the search: cells that fail because no convention fits, not just cells that fail on consistency.

The grammar stays small. Conditions compile to a flat conjunction of at most three predicates over `EQUALS`, `IS_NULL`, `IS_NOT_NULL`, `LTE`, `GTE` and `IN`. There is no `OR` and no `NOT`: two practices are two memories. A conjunction also has to beat both of its single-predicate parents, which is what stops the model buying accuracy with a second clause.

4.6 When the condition isn't expressible

The interesting output is the rejected one. A proposal can come back marked inexpressible, meaning the model has identified the practice and we have no fact to key it on. Those land in a gap log with their causal stories, clustered by practice across the fleet, and surface as demand: this many cells this week wanted a fact we do not record. Vocabulary now grows because evidence asked for it, rather than because we guessed which feature the next customer's habits would turn on. The registry has gone from eight fields to eleven since, and the log is what decides what the twelfth will be.

4.7 Backtesting

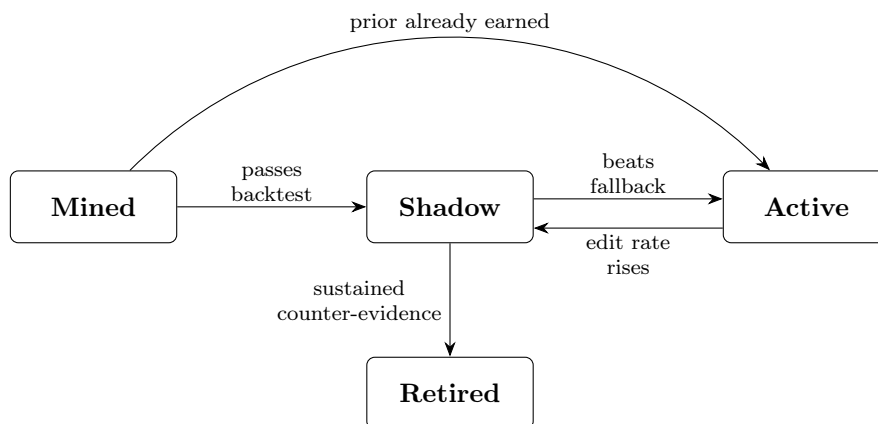
A surviving candidate is replayed against historical documents in its cell and scored against what users actually did. The counter-evidence failure above added two requirements to that replay.

An **exposure floor**: the documents the condition matches must include enough that a human actually looked at, so a rule cannot qualify by isolating a thin, rarely-reviewed slice. And a **published-record check**: of all the documents in the ledger that the condition matches, what fraction actually exhibit the practice? That is the check we did by eye on the Spain rule, made mechanical.

Candidates that fail here never reach a user at all. Candidates that pass are minted.

5 The memory lifecycle

Every memory sits in one of 4 states, and moves between them on evidence and counter-evidence.



5.1 Shadow

The default state a newly-proposed memory enters. A shadow memory runs on every eligible document and writes nothing. It produces the value it would have written, we record that alongside what the user actually did, and the comparison accumulates as evidence.

A shadow memory can be wrong for as long as it likes without a user ever seeing the consequence. That is why we can afford to propose speculatively at the mining stage, and it is how the design gets away with never asking a user to confirm a rule.

5.2 Confidence

Each memory carries a rolling confidence score, updated by Bayesian inference. Every agreeing observation raises it and every disagreeing observation lowers it, and recent events are weighted more strongly than older events. Decisions use the lower bound of the interval rather than the mean, so a rule with three observations does not get treated like a rule with forty.

5.3 Promotion

We choose whether to promote a memory into “active” depending on how the memory compares against the fallback. The fallback is whatever value we would have written had the memory not existed.

A memory that is right 70% of the time is worth having if the fallback was right 40% of the time, and detrimental if the fallback was already right 75%.

Two adjustments came out of running this. The margin is a share of the headroom the fallback leaves rather than a flat number of percentage points, because a flat margin is unpayable wherever the fallback is already good: at a baseline of 0.85, demanding a flat 15 points sets the bar at 1.0, and no amount of clean evidence can ever clear it.

And the bar depends on what the memory touches. Coding a document to the wrong account misstates the ledger, which costs more and takes longer to notice than a malformed reference,

so account memories have to be right more often and be seen working more often before they go live. Cosmetic memories are held to less.

A memory whose mining evidence was long and consistent can skip shadow mode and go straight to active. This is a real trade, made deliberately: such a rule is first tested on a document that gets published rather than one it was only watching, and demotion is what catches it. The patience is only granted for a mining record, never for a high prior that came from the fallback already being good, since that says nothing about the rule.

5.4 Demotion

Once a memory is active it keeps being measured, now using per-field edit rates in production. A user changing a field that a memory filled is counted as evidence against that memory. Confidence falls, and once it drops below what the fallback would have achieved, the memory returns to shadow mode, where it either recovers or accumulates enough counter-evidence to be retired.

5.5 Maintenance

Settlement and the lifecycle pass run every fifteen minutes through the working day, so an active memory that starts going wrong is demoted within the hour rather than overnight. The same pass merges memories that have converged on identical behaviour and adjusts memories that are underperforming without yet failing, by narrowing an over-broad scope rather than discarding the rule completely. A memory that sits in shadow for 90 days without gathering evidence is retired for stalling.

6 Keeping the fleet honest

Confidence only moves when a document arrives that the memory applies to. Plenty of rules are quiet for weeks, and a quiet rule is ambiguous in a way that matters: corrections attributed to a firing memory drop out of the mining input, so a silent cell means either the memory is working or a bad memory is suppressing the very evidence that would retire it.

Two nightly passes address that, and neither of them waits for traffic.

The **census** walks the live fleet and re-reads the published record. If the ledger contradicts a rule enough times, the rule is retired retrospectively, whether or not anyone corrected a document.

The **reviewer** reads each memory in turn against a sample of its own evidence and argues about it in natural language: is this rule still describing what the firm does, is it a proxy for something else, has it drifted. It cannot retire anything on its own. It can only propose, and a retirement proposal is refused unless the fit evidence supports it and the evidence spans more than one editing session. Its value is not the retirements; it is that a rule which is technically passing while being obviously wrong to a reader gets flagged before a customer notices.

7 Teaching the system to improve itself

Everything above runs unattended. That is the point, and it is also the risk. The five gates, the two nightly passes and the condition search all fail quietly, and their failures look identical to a quiet week.

The instrumentation for that is a weekly profile: a deterministic summary of what the miner did and did not do. Which cells missed which gate, and by how much. Which cells have been chronically rejected for weeks. Which memories are flapping between states. What the gap log is asking for. It states symptoms and nothing else, and it makes no attempt to explain any of them.

Explaining them was a human job, and it was the job most likely to stop happening. So it is now an agent.

7.1 The diagnostician

Once a week, two hours after the profile builds, a Temporal workflow starts a Claude agent and hands it the profile's signals as a worklist. Each signal gets an investigation: form a causal hypothesis, verify it against data it queries itself, root the cause in the miner's implementation, cost the fix, and recommend a next move in plain words.

It has four tools. Three of them read the deployed source tree, which the worker image already carries, so it can read the gate it is reasoning about rather than guessing at it. The fourth runs a single read-only SQL statement against production inside a read-only transaction with a statement timeout and a row cap.

There is no shell, no write tool, and no network. That restriction is load-bearing, and getting it right was the most valuable thing the rehearsal produced: the agent framework auto-approves its own built-in read tools before any permission callback is consulted, so our first sandbox was decorative. A probe confirmed the agent could read the container's environment file, credentials and all. Enforcing the sandbox in a pre-tool hook instead closed it. We would not have found that by reading the configuration, only by asking the agent to try.

The doctrine it works under is written down, and most of it is about restraint:

The risk it exists to prevent is a plausible-but-wrong root cause with an excellent story. A model with code access writes better wrong stories, not fewer. Every causal claim has to carry a query that was actually run, with the result attached. A claim without run evidence is marked as a hypothesis and can never be recommended for action. Its own sense of plausibility never overrides a measured statistic; it only justifies looking harder.

Some conclusions are refusals. Where the answer turns on what a customer intends, the recommended next move is to ask them, naming the question and who at the firm can answer it. Where it turns on a measurement that needs an offline backtest, the next move is to run that backtest first. Proposing a change before that information exists is overreach however clean the fix looks, and concluding that a real problem is not worth fixing yet counts as a success.

It does not trust claims about our own repository, including its own. It cannot see branches or deploys, so an entry saying a fix has shipped is treated as a claim to be measured. It has to name the effect the fix should have had and query for it. A fix that shipped without working is a better finding than the symptom it was meant to address.

Findings accumulate in an append-only table keyed by a stable slug, which is the agent's memory across runs: standing knowledge about the system, open diagnoses, and deferred fixes with a demand counter that ticks up each week the same gap costs us corrections. It is told not to re-diagnose what is already there, and to report demand instead.

7.2 What it found

The first supervised production run took 63 minutes, cost about \$12.50, investigated four signals and wrote 23 entries. Its best finding was one we had been looking at from the wrong end for a fortnight.

One bookkeeping firm’s reference cell had been chronically rejected. Their staff prepend their own initials to our reference at review time, four different sets of initials over 689 corrections in 90 days across 142 contacts, and pooled together those sit at 0.638 consistency against a 0.7 bar. The agent first established that the initials are not an extraction-time fact at all: they track neither the uploader nor the sender nor the publisher, because everything arrives through shared forwarding mailboxes and one bookkeeper publishes all of it. So no condition could ever split that cell, and our existing veto was right to reject it.

Then it looked past publication. After these documents reach the ledger, 99.5% of the references are rewritten there, and rewritten uniformly: whichever initials the reviewer typed become one particular set. The firm’s actual convention is a single clean prefix, worth 93% at organisation level, and it is invisible to the miner purely because the miner sources the review-time correction rather than the ledger-final value. The review-time value is a placeholder the firm systematically overwrites.

The recommended next move was not to change the miner. It was to replay the miner against the ledger-final value offline, estate-wide, and measure both the cells this converts and any regressions, and separately to ask the customer whether that prefix is a stable convention rather than one approver’s habit. Worth roughly 420 to 500 corrections a week for that firm alone, at about a day of work to find out. It also noted, unprompted, that around 186 of the 689 corrections predate memories that are now live and would be double-counted by a naive replay.

In the same run it declined to build something its own trigger had fired for, on the grounds that two organisations wanting per-document job tags is demand worth recording and not yet demand worth engineering.

7.3 The loop, and where it stops

The weekly report and the ranked roadmap render on our admin dashboard, and the only recurring human job is pressing one of five buttons per item: implemented as proposed, implemented differently, noted, already knew, or wrong.

Those verdicts exist to grade the diagnostician rather than to file the work. The bar it has to clear is naming the right fix before the team finds it by hand, which means recurring *implemented as proposed* with rare *wrong*. *Already knew* counts against it: correct, but trailing.

We are deliberately not automating the next rung yet. Taking a verdict and opening a pull request is two or three days of engineering, and it is not the constraint. Every fix the diagnostician has recommended acting on immediately has been mechanical instrumentation work, and we have no track record of verdicts to calibrate against. The measurements are the better thing to automate first, because “measure this before deciding” is the recommendation it makes most often and the one a human is slowest to get around to.

8 Status

Three days from idea to production, three weeks to a system that maintains itself.

234 memories are live across the fleet: 179 active and 55 in shadow, with 17 retired. 123 were

mined from corrections, 8 came from the condition search, 59 were ported from the original prototype, and 61 are hand-written prompt guidance. 14 carry a condition. 520 verification events have settled, 347 of them in shadow.

Mining runs nightly, settlement and the lifecycle every fifteen minutes through the day, the profile and the diagnostician every Monday morning. The diagnostician has 23 entries under management and its first unattended run lands next week.