SERIES

THE RECORD · FOUR PAPERS · OPEN LICENCE

The Second *Layer*.

Working notes on a memory shared across repositories, an organisation and the models of several vendors

PAPERS

PAPER 001 **An empty answer is not ignorance** METHOD · 8 MIN

PAPER 002 **The organisation needs its own read** CONSTRUCTION · 7 MIN

PAPER 003 **A brief is a budget** METHOD · 7 MIN

PAPER 004 **What a shared memory cannot know about its reader** LIMITS · 6 MIN

ISSUED

four papers · published 19 September 2026 · CC BY 4.0

digital1.foundation/articles/the-second-layer/

Stichting Digital One Foundation i.o. · Amsterdam

These notes describe work contributed to an entity in formation.

TOC

THE SECOND LAYER · FOUR PAPERS · OPEN LICENCE

Contents

001 **An empty answer is not *ignorance*** 4

METHOD · 8 MIN · PUBLISHED 19 SEPTEMBER 2026

Filter after an approximate index has chosen, or rename the key a history is filed under, and a memory holding the answer returns nothing, with no error raised.

§1 A memory that knows, and says nothing 4
§2 An approximate index filters after it has chosen 5
§3 A renamed identifier strands its history 7
§4 An empty answer describes the read, not the store 8
References 9

002 **The organisation needs its *own read*** 10

CONSTRUCTION · 7 MIN · PUBLISHED 19 SEPTEMBER 2026

Ranked in one read by nearness of scope, a large repository can take every seat. A second read, and the reason its scores compare with the first.

§1 Nearness of scope is a sound ranking rule, and a greedy one 10
§2 The first seat was offered inside a list the organisation could not enter 11
§3 A second read puts the organisation on the repository's footing 12
§4 What a seat cannot judge 13
References ??

003 **A brief is a *budget*** 15

METHOD · 7 MIN · PUBLISHED 19 SEPTEMBER 2026

Overlap measured against the smaller sentence lets a vague line silence a specific one. The symmetric rule, a cap per source, and why disagreement is never repetition.

§1 Distillation says one thing many times 15
§2 Overlap has to be measured against the larger of the two 16
§3 Disagreement is not repetition, and a file is one voice 17
§4 A filter at reading time cannot fix what was stored 18
References ??

004 **What a shared memory cannot know about its *reader*** 20

LIMITS · 6 MIN · PUBLISHED 19 SEPTEMBER 2026

It cannot make an agent read it, see which model did, or tell a stale sentence from a true one.

§1 The memory sits behind a protocol, not inside a model 20

§2 The harness decides when the memory is read 21

§3 A credential names an organisation, not a model 21

§4 What is open, and what would settle it 22

References 23

END **Colophon** 24

SOURCES AND THEIR HASHES, LICENCE, AND HOW THIS DOCUMENT IS SET

001

PAPER 001 · METHOD

An empty answer is not *ignorance*.

A memory shared by an organisation's coding agents can hold exactly what an agent needs and still return nothing, with no error raised. Two unrelated mechanisms do it, and both are general: the first meets any memory that filters after an approximate index has chosen its candidates, the second any memory whose scope key changes form while its history stays filed under the old one. **A read that returns nothing is a claim about the read, not about the store** — and a brief that forgets the difference tells a model that nothing was ever written down.

PAPER · 001 PUBLISHED · 19 SEP 2026 READING · 8 MIN

Canonical · digital1.foundation/articles/the-second-layer/001-an-empty-answer-is-not-ignorance.html

§1

A memory that knows, and says nothing

The memory these notes describe sits beside an organisation's coding agents and is read at the start of a task. What it holds is written in ***scopes***, nested: a repository, a team, and the organisation above them. An agent working in one repository asks for a short brief — what is known here, and what the organisation knows that bears on it — and the brief is assembled from the nearest memories to the task's own words, restricted to the scopes that agent may read.

When a read comes back with no rows, the brief built from it has one honest thing to say and one easy thing to say. The easy one — nothing is known here — is the failure that matters for a memory whose whole job is to stop an agent re-deriving what a colleague already found, because a model handed it will reasonably proceed as if nothing had been written down. Both causes below were found in testing, and both are repaired.

THE PREMISE, PRECISELY

An empty result licenses one conclusion: ***this read, as executed, matched nothing***. It does not license "the store holds nothing relevant", and a brief that says the second has promoted a property of the query plan into a statement about the world.

Two causes were found, independently, and each alone produces the empty brief. The first is in how an approximate nearest-neighbour index interacts with a filter. The second is in the name the history was filed under.

§2

An approximate index filters after it has chosen

A vector index of the kind used here, HNSW, answers "which stored items are nearest to this one" by walking a layered graph and keeping a bounded list of the best candidates it has seen.[2] The bound is what makes it fast. In pgvector, a condition in the query — ***only this repository***, ***only this organisation*** — is not part of that walk. pgvector's documentation says so directly: "With approximate indexes, filtering is applied ***after*** the index is scanned. If a condition matches 10% of rows, with HNSW and the default `hnsw.ef_search` of 40, only 4 rows will match on average."[1]

The index here covers current memories only, so the filter that matters is scope. One store serves every repository in an organisation, so any one repository is a share of it, and a scoped read filters a 40-candidate list down to that share, and when a prompt's nearest neighbours happen to live in other repositories, that share is zero. Nothing in the result distinguishes that from a scope with nothing to say.

The mechanism is reproduced below on a table anyone can build: 100,000 random vectors, 2% of them in one scope. Twenty probe vectors each ask for the ten nearest memories in that scope, first at pgvector's defaults and then with an iterative scan (added in 0.8.0),[1] a candidate list of 100 and a ceiling of 40,000 visited tuples — the three settings changed together, as they are in the repair.

```
-- pgvector 0.8.4, PostgreSQL 16 · docker image pgvector/pgvector:0.8.4-pg16
    CREATE EXTENSION IF NOT EXISTS vector;
    SELECT setseed(0.42);
    CREATE TABLE memories (id int PRIMARY KEY, scope text, embedding vector(32));
    INSERT INTO memories
    SELECT i, CASE WHEN i % 50 = 0 THEN 'acme/app' ELSE 'acme/other-' || (i % 49) END,
           (SELECT array_agg(random() - 0.5) FROM generate_series(1, 32) WHERE i >
    0)::vector
    FROM generate_series(1, 100000) AS i;
    CREATE INDEX ON memories USING hnsw (embedding vector_cosine_ops);
    ANALYZE memories;
    CREATE TABLE probes AS
    SELECT p, (SELECT array_agg(random() - 0.5) FROM generate_series(1, 32) WHERE p >
    0)::vector AS q
    FROM generate_series(1, 20) AS p;
    -- the read: ten nearest memories in one scope
    CREATE FUNCTION hits(q vector) RETURNS int LANGUAGE sql AS $$
      SELECT count(*)::int FROM (SELECT id FROM memories WHERE scope = 'acme/app'
                                 ORDER BY embedding <=> q LIMIT 10) t $$;
    SET enable_seqscan = off;
    -- the plan must be the index; a sequential scan answers exactly and proves nothing
    EXPLAIN (COSTS OFF) SELECT id FROM memories WHERE scope = 'acme/app' ORDER BY
    embedding <=> (SELECT q FROM probes WHERE p = 1) LIMIT 10;
    SELECT 'default (ef_search 40)' AS arm, min(hits(q)), round(avg(hits(q)), 2) AS mean,
    max(hits(q)),
           count(*) FILTER (WHERE hits(q) = 0) AS empty FROM probes;
    SET hnsw.iterative_scan = relaxed_order;
    SET hnsw.ef_search = 100;
    SET hnsw.max_scan_tuples = 40000;
    SELECT 'iterative, relaxed_order' AS arm, min(hits(q)), round(avg(hits(q)), 2) AS
    mean, max(hits(q)),
           count(*) FILTER (WHERE hits(q) = 0) AS empty FROM probes;
```

BUILD	DEFAULTS · MEAN HITS OF 10	DEFAULTS · EMPTY READS	ITERATIVE, EF_SEARCH 100 · MEAN HITS OF 10	ITERATIVE, EF_SEARCH 100 · EMPTY READS
Run 1	0.80	8 of 20	10.00	0 of 20
Run 2	0.95	8 of 20	10.00	0 of 20
Run 3	0.95	7 of 20	10.00	0 of 20

The three runs differ because HNSW construction is randomised, so each build is a slightly different graph; the seed fixes the data, not the graph. The default arm lands where the documentation's arithmetic says it should — 40 candidates at 2% is 0.8 — and between seven and eight of every twenty reads come back with nothing at all, from a scope holding 2,000 rows. The vectors are uniform and scope is assigned independently of them, which is the case the documentation's arithmetic describes. Real memories cluster by topic, and a repository's memories sit near each other, so the rate on a real store can differ in either direction: the experiment shows the mechanism, not a rate. One trap in re-running it: an earlier run that raised `maintenance_work_mem` failed to build the index inside the container, the planner fell back to a sequential scan, and every read returned ten exact answers. A sequential scan proves nothing about the index, which is why the plan is checked before the numbers are read.

DEFECT · FOUND BY COMPARING SCOPED AND UNSCOPED READS

A scoped read can return a fraction of a scope's memories, or none, while reporting success

The fix is the table's right-hand columns, set per transaction on every read: an iterative scan in relaxed order, a candidate list of 100, and a ceiling of 40,000 visited tuples. Relaxed order is acceptable here because every result is re-ranked afterwards by a score that is not pure distance. Measured on the live store, inside a transaction that was rolled back:

```
a read of the live store, scoped to one repository, limit 60
defaults                        20 of 60 rows
iterative scan, relaxed order   60 of 60 rows
```

§3

A renamed identifier strands its history

The second cause needed no index at all. On 18 September 2026 the memory's clients were aligned on one form for naming a repository, `owner/name`, which is unambiguous across organisations; until then some writes had used the bare `name`. Reads and writes from then on used the new form. Nothing was rewritten: memories already stored stayed filed under the name they were written with.

A read asking for `owner/name` therefore saw only what had been written since the change, and none of the history before it. Again nothing was malformed: the scope exists, the read is permitted, the result is well-formed and small. A brief built from it describes a repository that started the day its name changed — a failure any system meets when the key a history is filed under changes form and the history does not move with it.

The repair is on the read side and deliberately narrow. Whenever a read names a repository as `owner/name`, the bare name is appended to the scopes it may see, and ranked as the same repository rather than as one more rung between the repository and the organisation — otherwise the repository's own history would score as if it belonged to a neighbouring scope. A repository marked sensitive has its bare name excluded too, so a history filed under the old key cannot leak past an exclusion written against the new one. Writes are unchanged.

AN ALIAS IS A BRIDGE, NOT A MIGRATION

The alias maps `owner/name` to `name` by its last segment. Two repositories in one organisation with the same final name under different owners — a fork beside its upstream, say — would each be shown the other's pre-rename history. The correct end state is rewriting the stored rows and removing the alias; until then the alias is a known over-reach traded for the history being visible at all.

§4

An empty answer describes the read, not the store

Both defects share a signature: the store held the answer, the read was allowed to see it, and the result was an honest description of a query that asked the wrong question. Neither is detectable by validating the result, because a well-formed empty set is valid. They were found by comparing a scoped read with the same read unscoped, and seeing the scoped one come back empty where the unscoped one did not.

Three limits remain. The iterative scan is bounded: a scope rare enough that 40,000 visited tuples do not reach its members can still come back short, and the read does not say so. The alias is temporary and over-reaches in the case above. And nothing in a result set distinguishes a read that matched nothing from a store that holds nothing; a brief has to be told which one it is looking at. What would settle that is a read that reports how much it examined beside what it matched, so that "searched 40, matched 0" is never printed as "knows nothing". These notes do not claim that change is made.

ON THE PROVENANCE OF THIS MATERIAL

The defects here were found in ***SkilledMind***, a Digital One property, while testing it; that repository is not public, so the one figure quoted from the live store in §2 is its builders' account and cannot be re-run by a reader. The index experiment in §2 is different: it is printed as it was run, apart from its header comment, against a public container image. Deliberately absent: prices, plans and tiers, and deployment internals.

References

1 pgvector, ***README*** at tag v0.8.4, sections "Filtering" and "Iterative Index Scans"; the quotation in §2 is from "Filtering". github.com/pgvector/pgvector/blob/v0.8.4/README.md, read 19 September 2026.

2 Yu. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs", arXiv:1603.09320 (2016).

002

PAPER 002 · CONSTRUCTION

The organisation needs its *own read*.

A memory shared by an organisation's repositories is only a second layer if what the organisation knows reaches the agent working in one of them. Ranking both in a single read by nearness of scope works against that in a large repository: the scope term hands the repository a lead on every candidate, and in the case measured here **every seat went to the repository** while the organisation's rule for the task was not a candidate. The repair is a second read, and the reason its scores can be compared with the first.

PAPER · 002 PUBLISHED · 19 SEP 2026 READING · 7 MIN

Canonical · digital1.foundation/articles/the-second-layer/002-the-organisation-needs-its-own-read.html

§1

Nearness of scope is a sound ranking rule, and a greedy one

The memory these notes describe files everything under one of three nested scopes: a repository, a team, the organisation. When an agent in a repository asks for a brief, the read spans that repository and the organisation, and every candidate is scored on six terms: similarity of meaning, shared words, recency, the standing of the layer it was distilled into, which part of the repository it came from, and ***scope proximity*** — how near its scope is to the one the agent is working in. The weights are fixed:

```
score = 0.55 meaning + 0.17 words + 0.06 recency + 0.05 layer + 0.09 scope + 0.08 path

scope proximity, for a read of [repository, organisation]
  the repository     1.0
  the organisation   0.0        a team, when named, sits between at 0.5
```

The rule is right in its own terms. A note written in this repository is more likely to be true here than the same sentence written elsewhere, and when two memories say similar things the nearer one should win. But

0.09 is a constant offset, and a constant offset does not compare two memories — it compares two populations. To outrank a repository memory, an organisation memory must lead it by 0.09 on the other five terms combined; to enter a list of 32 it must lead the 32nd-best repository memory by that much. In a repository with fewer memories than the list holds, the organisation enters because there is room. In one with a long history it needs a lead that the prompt tested here did not give it.

§2

The first seat was offered inside a list the organisation could not enter

The first repair gave the organisation a seat. When a read spans a repository and the organisation, the two best organisation hits within 85% of the top score are moved to second and fourth place in the brief, so that what the organisation knows is carried even when a repository has a great deal to say. The candidate list was raised from 16 to 32 at the same time, to give the seat room.

It was tested the way it would be used: a brief requested from inside a repository with a long history, for a task squarely inside the organisation's own rules.

DEFECT · FOUND IN END-TO-END TESTING

The seat was checked for in a list the organisation's memories could not reach

The organisation held a rule that governs this task, and on its own footing it cleared the bar. Two things stood in its way. It was not among the 32 candidates, and had it been, it would have been judged at its combined-read score, near 0.41, below the bar of 0.491.

```
prompt: "write a new Foundation paper about shared agent memory"
    read over [repository, organisation], k = 32
      candidates from the repository     32 of 32      score 0.578 … 0.539
      candidates from the organisation    0 of 32

    the same prompt, read over [organisation] alone, k = 5
      0.541  what the organisation's shared memory is
      0.495  papers here make one claim, in 5 to 8 minutes       the rule that
    governs the task
      0.485  ground generations in retrieved context
      0.459  the default model and harness for coding agents
      0.445  an unrelated line

    the bar, 0.85 × 0.578 = 0.491   → two organisation memories clear it
```

The arithmetic of §1 explains it exactly. Scored inside the combined read, the organisation's best memory loses the 0.09 it would earn as the nearest scope and lands near 0.45, below the 32nd repository candidate at 0.539. The seat looked for organisation hits among candidates that the scope term had already made sure contained none. A larger list only moves the problem: it admits the organisation once it reaches below about 0.45 in repository scores, and how deep that is depends on how much the repository holds, not on the question asked.

§3

A second read puts the organisation on the repository's footing

The repair runs a second read beside the first whenever a request spans a repository and the organisation: the same prompt, restricted to the organisation's scope alone, four candidates deep. The two run concurrently, so the second adds a query to the database's work but not, on its own, to the wait; its latency was not measured.

The comparison is what needs arguing. Scores from two different reads are not in general comparable, and a bar of 85% of the top score is meaningless if the two tops were computed differently. Here they are computed the same way: every term is a function of the query and the memory alone, and none is scaled against the other results of its read. In a read of the organisation alone, the organisation is the nearest scope, so its best memory earns the full proximity term — exactly as the repository's best memory does in the combined read. Both tops sit on the same footing, and an organisation memory is held to the bar it would have faced had it been written in the repository itself. The best two across both pools that clear it take the second and fourth places — and an organisation memory that also made the combined list is judged by its score from the organisation read, never by the lower one it carries there, so that reaching the list can never make a memory harder to seat.

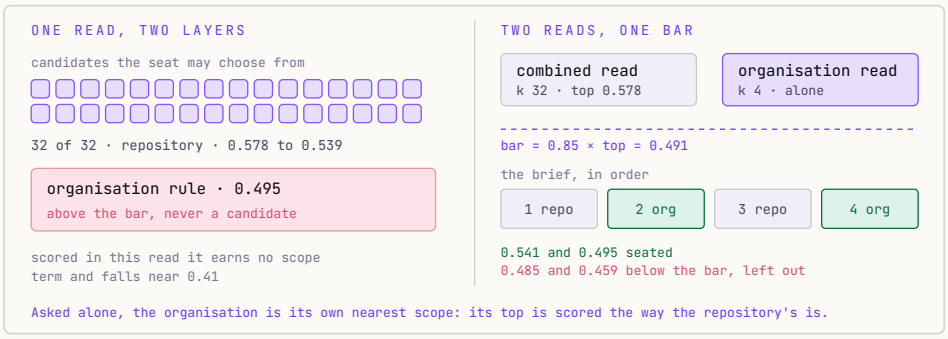

FIGURE 1 A seat is only as good as the list it is chosen from. In the case measured, the scope term kept every organisation memory out of the candidates of one read; a read of the organisation alone scores it on the footing the repository already had, and one relative bar then decides.

After the repair was deployed the same prompt produced a brief with organisation memories second and fourth, at 0.541 and 0.495; the other two the organisation read returned, at 0.485 and 0.459, fell below the bar of 0.491 and were left out. A second prompt — how a new paper here should be structured and how long it should be — put the organisation's rule second; the next organisation hit, at 0.507, fell short of the bar of 0.85 × 0.654 = 0.556, so the fourth place stayed with the repository. That is the intended shape: **the bar is what stops the seat being guaranteed**. A test fixes the property in place: forty repository memories that alone fill the candidate list, one organisation memory on the same topic, and an assertion that it reaches the first two places of the brief. It fails against the first repair and passes against the second.

§ 4

What a seat cannot judge

The seat decides ***where*** the organisation speaks. It has no view on whether what it says is worth hearing. Four limits remain.

A seat is taken by whatever the organisation holds. Text filed at the organisation's level that nobody in the organisation wrote — sample content, a bulk import of reference material — competes for the seat on the same terms as a decision somebody stated, and on a prompt where its wording happens to be closer, it wins. What reaches an agent through this device is exactly as good as what is filed at that level, which makes curation of the organisation's scope a duty rather than a nicety.

The bar is relative. Eighty-five percent of a weak top is a low bar. When the repository has nothing close to the prompt, the organisation's seat is easier to win, which is usually right and occasionally admits a line that is merely less irrelevant than the rest. The figure 0.85 is a chosen setting, not a measured optimum, and nothing here claims otherwise. Nor is the margin wide: on the first prompt the rule cleared the bar by 0.004, so its seat there rests on the third decimal place, and a slightly different wording could lose it.

A read cannot find what was never written. The organisation's strategy, its positioning, the decisions it has taken about how it works — these reach an agent only if a person has stated them at the organisation's level. Distillation from repositories produces what repositories say; it does not produce what the organisation decided. For an organisation that wants its agents to know its strategy, the first step is not in the memory at all.

The second read is the organisation's alone. A team, when named, gives up 0.045 on every candidate and has no read or seat of its own; whether it is crowded out the same way was not tested.

How repeated practices are promoted from repositories to the organisation, and what a new project may then be told on its first day, is the subject of The Inherited Default; this paper assumes that stage and is concerned only with what is read back.

ON THE PROVENANCE OF THIS MATERIAL

The ranking, the defect and both repairs are from ***SkilledMind***, a Digital One property; its repository is not public, so the scores and positions quoted from the live store are its builders' account and cannot be re-run by a reader. The weights and the proximity ladder are printed in §1 so that the arithmetic can be checked without it. Deliberately absent: prices, plans and tiers, and deployment internals.

003

PAPER 003 · METHOD

A brief is a *budget*.

The brief an agent receives at the start of a task has a fixed size, and the agent has no way to tell which of its sentences to discount. A memory distilled from every revision of a file restates the same fact many times, and without a rule against it **the budget can be spent saying one thing**. The rule has to be symmetric, and it has to know the difference between a repeat and a disagreement.

PAPER · 003 PUBLISHED · 19 SEP 2026 READING · 7 MIN

Canonical · digital1.foundation/articles/the-second-layer/003-a-brief-is-a-budget.html

§1

Distillation says one thing many times

The memory these notes describe is filled by distillation: each file an organisation writes down — a design note, a runbook, a decision record — is read, and the durable facts in it are stored as short sentences with a citation back to the file. A file is revised, and distillation runs again. A fact that survived the revision is re-derived, in slightly different words, and stored again. An identical sentence is stored once; a reworded one is stored as a new memory.

None of this is wrong as storage. It becomes wrong at the moment of reading. A brief is assembled by taking the memories nearest to the task, in order, until a character budget runs out — and the memories nearest to a task about the store can easily be the many ways of saying what the store is. Assembled that way, a brief can be mostly one fact in many phrasings from one file. In the test described in §3, six paraphrases of one process, all distilled from one file, are together longer than the 700-character brief they compete for. Every sentence in such a brief is true and well cited, and together they tell the agent one thing.

THE OPERATIVE QUESTION

Not "is this sentence relevant?" — every paraphrase is — but "does this sentence tell the reader anything the brief has not already told them?" That is a question about the sentence and what has already been carried, so it can only be asked while the brief is being assembled.

§2

Overlap has to be measured against the larger of the two

The test is deliberately plain, so that it can be checked by hand. A sentence is reduced to its ***content tokens***: lower-cased words of three or more characters, a fixed list of common function words dropped (***through*** and ***see*** are not on it), a trailing plural folded to the singular. Two sentences overlap by the number of tokens they share. A candidate block is skipped when its overlap with any block already in the brief reaches **0.6**, with two exemptions discussed below.

The question is what the shared count is divided by, and the first answer was the wrong one. Divide by the smaller of the two token sets and the measure asks "is the shorter sentence covered by the longer?" — which sounds like the definition of a repeat and is not.

DEFECT · FOUND BY AN EXISTING TEST OF THE BRIEF

A vague line mentioning a tool silenced the specific line saying where it runs

A general paragraph that mentions a deployment tool in passing covers most of the words of a precise sentence about how deployment works. Measured one-sidedly, the precise sentence looks redundant and is dropped. In the test that caught it, the vague line was a README chunk; the pair below is written for illustration, and the tokens and ratios are computed with the brief's own rule.

```
carried   The stack is Postgres, Redis and the API, started with docker
          compose; see the deploy notes for the box.
          → stack · postgre · redis · api · started · docker · compose ·
            see · deploy · note · box   11 tokens
candidate Deploys go through docker compose on the box
          → deploy · through · docker · compose · box   5 tokens

shared 4
  4 ÷ smaller (5)   = 0.800   ≥ 0.6 → skipped: the precise line is lost
  4 ÷ larger (11)   = 0.364   < 0.6 → carried
```

Dividing by the larger set makes the test symmetric: each sentence must cover most of the other before either counts as a repeat. A vague line can no longer silence a specific one merely by containing its nouns. The cost is that some true repeats of unequal length survive, and that is the right direction to err in — a brief with one sentence too many is slightly less efficient, while a brief missing the one precise sentence is wrong.

§3 Disagreement is not repetition, and a file is one voice

The first exemption is polarity. Two sentences that share every content token and differ by a negation are a pair a brief must not collapse, because they are a disagreement — one team does something, another has forbidden it, or a rule was reversed. A block whose negation differs from a carried block's is never a repeat, however high the overlap. The second exemption is length: a block of fewer than three content tokens is too short for the ratio to mean anything and is never skipped on it.

CARRIED	CANDIDATE	OVERLAP ÷ LARGER	OUTCOME
The store is PostgreSQL with pgvector for cosine similarity recall.	Store technology: PostgreSQL with pgvector, recall by cosine similarity.	0.857	skipped · a repeat
The stack is Postgres, Redis and the API, started with docker compose; see the deploy notes for the box.	Deploys go through docker compose on the box	0.364	carried · more specific
Run migrations on API boot.	Never run migrations on API boot.	1.000	carried · opposite polarity

One more rule is not about wording at all. Paraphrases that escape the overlap test still share a source: they cite the same file. A brief therefore carries **at most two blocks citing the same first source** — in practice, one file; blocks with no source are not capped — and the rest of the budget goes to other files, other repositories and the organisation's own layer. Two, rather than one, because a long document can hold two genuinely different facts a task needs. The overlap test removes restatements that are close in wording; the cap is what stops the loose ones.

The rules have their own tests. In one, three paraphrases of the store's technology are carried once, and both "Run migrations by hand" and "Never run migrations by hand" are carried. In another, six long paraphrases of one process from one file meet a 700-character brief: at

most two are carried, and the room left goes to other sources — here including the organisation's own fact, seated by the mechanism in Paper 002.

§4

A filter at reading time cannot fix what was stored

Every rule above runs while the brief is assembled, on sentences already stored. That is where it is cheap to be careful, and it is also where some errors can no longer be undone.

Loose paraphrases survive. "The memory store is Postgres with the pgvector extension" and "Embeddings live in a Postgres table indexed by pgvector" say the same thing and share two content tokens of six: an overlap of 0.333, and both are carried. Token overlap measures shared words, not shared meaning. The real cure is upstream — distillation that recognises it is re-deriving a fact it already holds, and updates it rather than adding a new one — and that is not what this paper describes.

Polarity is read from a list of words. "never", "not", "avoid", "without", a contraction ending in ***n't*** and a few others mark a sentence as negated. A reversal phrased without any of them is not seen as one:

```
carried    Run migrations on API boot.              run · migration · api · boot
  candidate  Stop running migrations on API boot.     stop · running · migration · api
  · boot

  shared 3 ÷ larger 5 = 0.600   neither negated   → skipped as a repeat
```

The reversal is dropped, and the brief carries the rule it reversed. Extending the list narrows this without closing it, because the ways English reverses an instruction are open-ended. Where a reversal matters, it should be stated with a negation — or, better, the superseded sentence should be retired from the store, so that there is nothing for a brief to choose between.

The cap does not read polarity. If one file already has two blocks in the brief, a third that contradicts them is dropped like any other third block. A reversal is safest stated where it is not the file's third voice, or with the superseded sentence retired.

Order decides which phrasing survives. Of two repeats, the one ranked higher is carried, and the other is skipped. The rules decide how much of the budget one fact may take, not which of its phrasings is best. And when several repositories state the same practice independently, that agreement is evidence in its own right: it is counted, not deduplicated, by the stage described in The Inherited Default.

ON THE PROVENANCE OF THIS MATERIAL

The rules and their tests are from ***SkilledMind***, a Digital One property; its repository is not public, so their behaviour is its builders' account. Every ratio in this paper is computed from the sentences printed beside it by the token rule stated in §2, so each can be recomputed by hand from the printed tokens. Deliberately absent: prices, plans and tiers, and deployment internals.

004

PAPER 004 · LIMITS

What a shared memory cannot know about its *reader*.

Put an organisation's memory behind an open protocol and an agent whose harness speaks that protocol can read from it and write to it, whichever vendor's model it runs on: the knowledge stops belonging to one tool. That is a real property, and it is narrower than it sounds. The memory decides what is served; it does not decide whether it is read, cannot see which model read it, and cannot tell a stale sentence from a true one.

PAPER · 004 PUBLISHED · 19 SEP 2026 READING · 6 MIN

Canonical · digital1.foundation/articles/the-second-layer/004-what-a-shared-memory-cannot-know.html

§1

The memory sits behind a protocol, not inside a model

The three papers before this one describe what a read returns. This one is about who can ask. The memory is exposed through the Model Context Protocol, whose Streamable HTTP transport specifies that "the server **MUST** provide a single HTTP endpoint path … that supports both POST and GET methods".[1] The specification lets a server assign a session; this one does not, so every request stands alone and carries its own credential. Behind it sit the same scopes, the same ranking and the same brief described in Papers 001 to 003, whichever agent is asking.

Two routes reach the same store and the same ranking. An agent that runs on a developer's machine connects through a small local bridge that speaks the protocol over standard input and output, works out the repository scope from the directory it was started in, and calls the memory's own interface. On one workstation that bridge was registered on 19 September 2026 with Anthropic's Claude Code, OpenAI's Codex CLI and Google's Gemini CLI, each pointed at the same store. An agent that runs in a vendor's cloud would connect to the endpoint directly: xAI, for

instance, documents remote MCP tools configured with a `server_url` — "Only Streaming HTTP and SSE transports are supported" — and an optional `authorization` token that "will be set in the Authorization header on requests to the MCP server".[2] That is a configuration these notes rely on but have not exercised (§4).

> WORTH STATING PLAINLY
>
> What is shared across vendors is the ***store*** and the ***rules for reading it***, not a model's understanding. Two agents on two vendors' models that ask the same question in the same repository at the same moment are served the same brief — if each harness asks for it. What each then does with it is its own.

§2

The harness decides when the memory is read

A protocol endpoint answers requests; it cannot make one. Whether an agent consults the memory at the start of a task, halfway through, or never, is decided by the program the model runs in — the harness — and by the model's own choice to call a tool it has been offered. Where the harness supports it, a hook can fetch a brief before the model's first turn, so the reading does not depend on the model remembering to ask. Where it does not, the memory is one tool among many, and a model confident it already knows the answer may not call it.

Writing is the same in reverse. A lesson reaches the organisation only if something in the agent's loop decides it is worth recording and calls the write. The memory can make that call cheap and well-scoped; it cannot make it happen. So the second layer is only as complete as the habits of the agents and people around it, and the agents' habits are configured separately in each harness, by whoever runs it.

§3

A credential names an organisation, not a model

Every request carries a key, and the key identifies the organisation whose memory it may read. It does not identify the model on the other side of the harness. The protocol lets a client name itself when a session starts; that names the harness, not the model, is not checked, and is not kept by an endpoint that holds no session. Where a client forwards the person it acts for — the configured git user, a CI actor — each write is recorded against that name; otherwise against the key's fingerprint. The record can say ***on whose behalf*** the client claimed to write. It can attest neither that claim nor ***which model*** wrote the memory, or read it.

SHARED MEANS SHARED MISTAKES TOO

A write from one agent enters the same store every other agent reads. A mistaken statement recorded by one can be served to any of them, looking no less settled than a true one. And a statement that was true when it was written — where a service runs, which setting is current — stays in the store after the world moves, and apart from a small recency term in the ranking it is served as it was on the day it was written. The memory ranks sentences; it does not check them against the present.

Two things follow for anyone relying on it. The organisation's own decisions — its strategy, how it prefers to work, what it has ruled out — reach an agent only if a person states them at the organisation's level; distillation from repositories produces what repositories say, not what the organisation decided. And retiring a sentence that has stopped being true is a deliberate act that somebody has to own, because nothing in the read path will do it for them.

§4

What is open, and what would settle it

OPEN	WHAT WOULD SETTLE IT	WHO DECIDES
A call from a cloud-hosted vendor agent	These notes rely on the vendor's documented configuration in §1; a live call from that vendor's API against this endpoint has not been exercised here. Running it, and recording what the brief looked like on arrival, would.	the memory's builders
An empty read reported as ignorance	A read that reports how much it examined beside what it matched (Paper 001, §4).	the memory's builders
History filed under a renamed key	Rewriting the stored rows to the new key and removing the read-side alias (Paper 001, §3).	the memory's builders
Paraphrase and reversals without a negation	Distillation that updates a fact it already holds instead of adding another; retiring superseded sentences (Paper 003, §4).	the memory's builders
Stale statements served as current	An owner for retirement at each level, and a record of when a statement was last confirmed rather than only when it was written.	each organisation using it
Which model read or wrote a memory	Nothing in the transport specification read on 19 September 2026. It would need the harness to assert it and a way to check the assertion, which is the subject of The Delegation Record.	the protocol's maintainers and harness builders

None of these is a reason not to share a memory across repositories and vendors. They are the reasons to describe what sharing it does accurately: one store, one set of rules, one brief for any agent that asks — and a store that is only as current, as curated and as consulted as the people and harnesses around it make it. Nothing above is committed; each item is open until someone settles it, and these notes settle none of them.

> ON THE PROVENANCE OF THIS MATERIAL
>
> The endpoint, the bridge and the write record described here belong to ***SkilledMind***, a Digital One property; its repository is not public, so their behaviour is its builders' account. The protocol and vendor statements are quoted from the published documents in the references, with the date read. No vendor is compared with another here, and none should be inferred. Deliberately absent: prices, plans and tiers, and deployment internals.

References

1 Model Context Protocol, ***Specification***, version 2025-06-18, "Transports", section "Streamable HTTP" and "Session Management". modelcontextprotocol.io/specification/2025-06-18/basic/transports, read 19 September 2026.

2 xAI, ***Remote MCP Tools***, "Configuration" table: `server_url` and `authorization`. docs.x.ai/developers/tools/remote-mcp, read 19 September 2026.

END

THE SECOND LAYER · COLOPHON

Colophon

A

The record

This document is generated from the published pages, which are the record. Where this file and a page disagree, the page is right and this file is stale. Each source below is named with the SHA-256 of the file this document was built from, so a reader can check that what they are holding is what was published; the generator fetches each live page as it builds and reports any that has moved since.

INDEX digital1.foundation/articles/the-second-layer/

001 digital1.foundation/articles/the-second-layer/001-an-empty-answer-is-not-ignorance.html
sha256 7be0d030515c8868eb74ee767c06d52232fd6841c17e53d9894f0f3042ea71dd

002 digital1.foundation/articles/the-second-layer/002-the-organisation-needs-its-own-read.html
sha256 ed165755c93f38fe4c14d2ff7405739e02e2a2147f71cdd388d3220344884613

003 digital1.foundation/articles/the-second-layer/003-a-brief-is-a-budget.html
sha256 748ff3fb57ca97400844db896b6668db90dabb78b597bb52765216d930d13db4

004 digital1.foundation/articles/the-second-layer/004-what-a-shared-memory-cannot-know.html
sha256 7c40d6fd97cbffb73c5c0d89e04bae7d16d2baecf520e59c90b7b3b324ac702f

B

Licence

C

How it is set

In the six faces the site itself carries and embeds: Fraunces 300 and 600 for titles and the emphasised word, Manrope 400 and 500 for the text, JetBrains Mono 400 for the ledger's own labels, code and running matter. Those six are Latin subsets and do not carry every character the papers use — → is absent from them. Nothing is drawn in their place and quietly dropped from the text: each is set in a named system face. Every

character in every paper was extracted back out of this file and checked before it was written, so the whole document copies, searches and cites as it reads.

Figures are the papers' own inline drawings, re-toned from the site's night ground to this page by a one-to-one swap of the closed house palette, so green still means it holds and rose still means it breaks — and the two are separated in weight of grey as well as in hue, so a photocopy keeps the distinction. The rail down the left of each page is the ledger from the Foundation's homepage; a filled dot marks each numbered section, and the § number sits in the gutter, outside the measure.

The sheet is A4 as the page asks for it, which Chromium rounds to 594.96 × 841.92 pt when it writes the file — 0.1 mm narrower and 0.03 mm taller than the 595.28 × 841.89 pt of the standard. No printer will show it; a document that prints its own hashes should say it anyway.

---

digital1.foundation
The Second Layer