

Where the Single Call Stops

An LLM leave-impact investigator against one-call readers
under a fixed output budget, on a constructed organization

Leave Impact Agent, the investigator milestone
Every number pinned to its record at build time

ardabasarici.dev · github.com/arda-basarici/leave-impact-agent

Arda Başarıcı · October 2026

Where the Single Call Stops

An LLM leave-impact investigator against one-call readers under a fixed output budget, on a constructed organization.

When an employee goes on leave, someone has to work out what their absence affects and who can cover it. The investigator is an LLM agent that reads an organization's HR system, issue tracker, calendar and documents and drafts that coverage plan with its evidence. This report measures whether its reading loop earns its cost against three simpler ways of producing the same plan, on an organization built so that the right answer is known.

System	Base level	Padded level
Investigator (the system under study)	24 of 30	24 of 30
Full context, one call	30 of 30	14 of 30
Single shot, one retrieval then one call	22 of 30	10 of 30
Rules only, no model	10 of 30	10 of 30

Four systems answered the same thirty leave scenarios on one generated organization, at two corpus levels, 240 runs in all for about \$9.96 in model calls. The table counts scenarios answered correct whole: every required claim present and right, nothing unexpected claimed, and no finding against the plan.

At the base level a single call shown every document is correct on all thirty and the investigator is correct on fewer. At the padded level the single call stops at its output allowance on 24 of 30 calls, the cap on what it may write and not on what it may read, and falls to 14, while the investigator holds its 24: an observed difference of 33 percentage points of scenarios, with a paired interval of 20 to 47 points at 95 percent.

Two things belong beside that result. The padded level's size was chosen, before the measurement world existed, so that the single call would fail on a development world. And one model, Claude Haiku 4.5, played every role in every system measured, on one world, with one run per scenario. The finding is about these systems under their registered output budgets. It says nothing about agents in general.

Contents

What it was measured on	3
What a correct answer contains	5
What the model adds beyond the rules	7
When the loop helps, and when full context does better	9
What the loop earns	12
What this does not show	14
Reproducing the numbers	16

What it was measured on

The systems in this report were measured on a synthetic organization whose answer key was written before any of them ran. That ordering is the point of the instrument. A benchmark labeled after the fact inherits the labeler's reading. A constructed one has no labeler, so the question becomes whether the construction can be trusted. This chapter describes how the organization is built and how its truth is checked, and says what a reader still has to take on trust.

One organization, thirty scenarios

The generator builds one organization from a seed. The golden world described below has 28 people in 5 teams, and the measurement world is of the same size. Its records are spread across four systems of record: an HR system holding people, skills and leave; an issue tracker holding work items and their comments; a calendar holding meetings; and a document corpus holding policies, runbooks and client notes. Into that organization the generator plants 30 leave scenarios in three tiers of ten. A *structured* scenario can be answered from records alone: who is on leave, which deadlines and meetings fall in the span, who else holds the skill. A *fragmented* scenario depends on facts that live only in prose, a qualification mentioned in a ticket comment, a responsibility named in a runbook paragraph, a staffing clause in a policy. An *adversarial* scenario plants a trap: a stale record contradicted by a newer one, a fact that exists nowhere, a need no one can cover.

Each scenario belongs to a class, and a class states what it needs as a query over the organization. A seeded choice picks one admissible construction, plants the entities and writes the expected answer. The shared rules that every system later uses to compose its report are then run over the planted truth as a consistency check: if the rules and the construction disagree, that is a named construction error, never a retry. The check shares its rules with every system, so a defect the rules themselves carry would pass it; the audits described below are the check on the rules. The same rules are asked again under each single-source outage, which is how the key records which sources a scenario's answer depends on. Not every seed yields a world; a refusal is recorded and reported, never repaired.

Prose enters through a gate

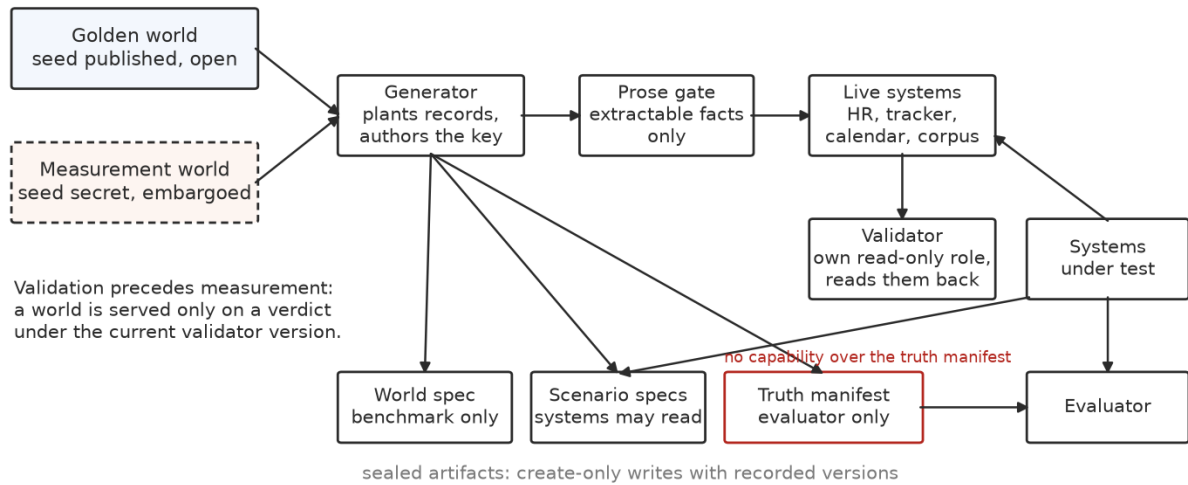
Records are planted directly, but prose goes through a gate. A language model writes the comments, runbook paragraphs and policy clauses from a brief that carries facts, never sentences. A text is accepted only if two checks pass: a model of a different family must extract from it exactly the planted facts and nothing else, and a lexical guard must find no name outside the brief's scope. On the golden world a person also read every text that would change an answer. A generation that fails is discarded, not patched. Accepted text becomes immutable world data, and the world's version is the hash of what was realized, not of the seed, so two worlds from one seed with different prose are two worlds. The gate also sets a limit chapter 6 returns to: prose accepted only for being extractable is near-template prose.

Validation, sealing, and who may read what

A world is written into live systems (a Frappe HR site, a Jira project, Google calendars and an object store), and a validator with its own read-only credentials reads them back. It approves a world only when every enumeration is complete and every record equals its planting, 7 of 7 exactness kinds and seven fidelity kinds, and when each scenario's view under the runtime rule matches the key. A world is served only on a verdict that judged its current manifest under the current validator version; when the validator gains a check, every served world is re-judged.

Three sealed artifacts serve three readers. The world specification is private to the benchmark. The scenario specifications are what a system may see. The truth manifest, the answer key, is readable by the

evaluator alone: the role the application runs under has no capability over the bucket that holds it, every final object is a create-only write with a recorded version, and a boundary probe on every deployment checks that the application still cannot reach the key.



The instrument. One generator produces two worlds through the same pipeline; validation precedes measurement; the application never holds a capability over the truth bucket; the measurement world's contents and seed stay withheld.

Two worlds

Two worlds matter here. The *golden world* is open in its seed and recipe, and it carried the benchmark's audit. Every one of its thirty scenarios passed a human acceptance pass, seventeen received a deep trace from the planted records to the key's outcome, a panel of four model reviewers read the audit blind with every disagreement adjudicated by hand, and the audit record was sealed under its content address. The audit's findings are public; its record holds answer keys and stays with the truth. The *measurement world* is the one the numbers in this report come from. Its seed was drawn from thirty-two bytes of entropy after the registration was frozen, so no system was tuned against it. It was validated under validator version 2, three of its scenarios were traced by the same audit method with a blind model reader, and five of its filler documents were read to confirm they name nothing of the world. Its version is public; its contents, its seed and the digests that would identify them are withheld until the project's last measurement on it, which is why this report cites its results and not its contents.

The measurement world also carries something the golden world does not: a pool of filler documents, requirement-style texts about fictional releases and clients that no scenario owns. A *corpus level* is a count from that pool. The base level includes none of it; the padded level includes the first 160. Levels nest, every answer-bearing document is in every level, and a test holds that a world assembled with the pool and then stripped of it equals the world assembled without, field for field. So the answers are the same at both levels, and only the amount a system has to read changes. Chapter 4 explains why the padding exists.

What a correct answer contains

A system's answer is a report: a set of typed claims about one leave, in a vocabulary frozen at the world milestone and shared with the answer key. An illustrative example first, with invented names. Dana is on leave for two weeks. The tracker shows a release ticket due in that window, so the report claims an *impact*. A policy clause says a release needs an engineer with iOS experience, so it claims a *constraint* keyed by that clause. The HR system lists nobody with that skill, but a ticket comment says "Priya has shipped two iOS releases", so a system that read the comment states that fact with those words, and the rules then assess Priya as viable and propose assigning her: a *candidate assessment* and a *coverage action*. A system that never read the comment assesses nobody as viable and must either leave the impact uncovered or admit an *unknown*. If the HR record and the comment disagreed about Priya, the report would carry a *source conflict* naming which the authority rule trusts.

Those are the six claim types. An impact is something the leave affects, a deadline, a meeting or a standing responsibility. A constraint is what a valid response must obey, keyed by the policy clause it comes from, so a constraint a system cannot trace to a clause cannot be expressed at all. A candidate assessment says whether a named person could cover a named impact, and why. A coverage action says what the plan does about an impact: assign someone, leave it uncovered, or admit it cannot tell. A source conflict records two records that disagree and which one the authority rule trusts. An unknown records a fact the answer needed and the systems did not hold. Every claim carries the evidence it was read from. A claim's identity is computed from its fields rather than from an id, so a report and the key can be compared even when their writers assigned different ids.

Who writes the claims

No model in this evaluation writes a verdict. A model states facts: this comment says this person has this skill, and here are the words that say so. The shared rules, the same deterministic code in every system and in the construction of the key, compose every claim from the structured records plus the facts a model stated. The rules decide who is viable, what the plan does, and where an unknown or a conflict stands. The model only states what it read.

That split was settled by a probe before the harness, or any model-reading system, was built, on a throwaway world with the development model. Asked to write the whole report itself in the codec's format, the model was correct on none of six scenarios; five of the six answers decoded, and most of those missed by one to three rows out of thirty or more. It failed scenarios the rules get right with no model at all. Asked only to state what it read, with the rules concluding, the same model reached 26 of 30 on the first prompt and 30 of 30 after the prompt was amended on that same world. The comparison is partly circular, because the facts arm concludes with the same rules the key was built from, and the probe noted this. It set a direction and is not a result. As a result, in every system here the model reads and states facts, and the rules build the report's structure.

Correct whole

The headline check is *correct whole*, and it is all-or-nothing per scenario: the report is structurally valid, every required claim is present with the right payload, nothing unexpected is claimed, and the plan it proposes raises no finding against the key. One wrong assessment in thirty rows fails the scenario. The check is severe on purpose, because a person cannot approve a coverage plan with one wrong name in it.

Two other checks are registered beside it and carry no claim. *Expected action* reads the coverage actions alone. *Reproduced whole* asks whether the run's own reads reproduce every claim it made: the rules are replayed over only what the run actually retrieved, with its claims as premises, and a claim is reproduced

when the rules conclude it, contradicted when they conclude otherwise, unsupported when they cannot. That replay is how grounding is measured; no model grades another model.

A claim is *grounded* when the run's own reads reproduce it, and grounded is a different property from correct. A system that reads no prose can have every claim it makes reproduced by its own reads and still be wrong wherever the answer lives in prose. A negative claim, that nobody holds a skill or that no clause applies, can be confirmed only after every place the answer could live has been read, so a system that read nothing cannot have its negatives confirmed by silence. Chapter 3 measures that distance.

The key is what a run could have found

The expected answer is runtime truth: what a system could obtain from the live records on the day it runs, not the dated view the generator used while constructing the scenario. The two were compared on forty seeds before the choice was made. They agree on every outcome and on every must-assess pair under the normal condition, and they disagree on 36 of 32,960 pairs the evaluator derives itself, which is why the key follows what a run can obtain rather than what the construction assumed. The key is sealed before any run and read by the evaluator alone.

The terms were fixed before the world existed

Every number in the results chapters comes from one evaluation under a registration that was frozen before the measurement world was drawn and bound to it after the world was accepted. The registration names the four systems and their roles, the two corpus levels, the three checks, one primary comparison, the confidence level and resample count, and the budget. It also sets a retry rule: only an infrastructure failure may be retried, and a fixed rule over the run's own log, not an operator, chooses the counted attempt. The evaluator reads the file at the freeze commit and refuses to run on a difference. The evaluation this report cites carries the label *reported*, which the evaluator derived from that bound registration.

How the intervals are computed

A single system's pass count carries a Wilson interval at 95 percent, the textbook interval for a proportion of thirty trials. A difference between two systems is paired on scenario: each scenario contributes whether the first system passed it and whether the second did, and the difference is resampled 10,000 times over whole scenarios within tiers, giving a percentile interval at the same confidence. Beside each paired difference the report prints its two-by-two table as four counts in a fixed order: the scenarios both systems passed, the first alone passed, the second alone passed, and neither passed. Every scenario ran once, so no term for run-to-run variation enters either interval; they describe variation across scenarios of this generator in one organization and nothing else.

What the model adds beyond the rules

Every system begins with the same frozen prefetch of structured records: it reads the leave, enumerates the whole employee set, reads the leaves that overlap the span, enumerates the components and work items, and reads the events that overlap the span. Every system composes its report with the same rules. They differ only in what documents reach the model and how.

Rules only reads the prefetch and no document, and calls no model. *Single shot* runs one fixed full-text query built from the leaver's component names, then one model call that states facts from what came back. *Full context* puts every document of the corpus level into one call. The *investigator* runs a tool loop: the model reads in turns through role-scoped tools over the same records and a full-text search over document sections, then is asked once, at the end, to state the facts it read with the words they came from.

The investigator as built

The investigator is a Python service running the loop on LangGraph, with Claude Haiku 4.5 reached through Amazon Bedrock and its state in PostgreSQL. Four decisions in its construction are each tied to the failure they prevent.

- *The event log is the authority.* Recovery replays the log, the audit reads it, and the run's export is built from it, so a crashed worker resumes from what was recorded rather than from a framework checkpoint that might disagree with it.
- *A claim fences a generation.* A new claim on a run fences the previous worker's authoritative writes: after it, the old worker authorizes nothing and appends no outcome. A request it had already authorized may still be sent or remain in flight; its outcome is never accepted, and it stays unresolved with its worst-case allocation retained.
- *Budget before inference.* Every model call is authorized against the budget before it is sent, carrying the input bound it was authorized on and the role's output maximum, and an attempt that closes without an export lists its known consumption and its retained liability as two numbers, since an absent export is not zero cost.
- *The evaluator is isolated.* It runs under its own identity, reads exports and never re-reads a vendor, so grading cannot touch the systems it grades.

Before any reported run, the worker was killed at every boundary two reference runs cross, 469 rows in all, recovering as forecast once two forecasts had been corrected, and the integration level of the test suite runs the real log against a PostgreSQL service.

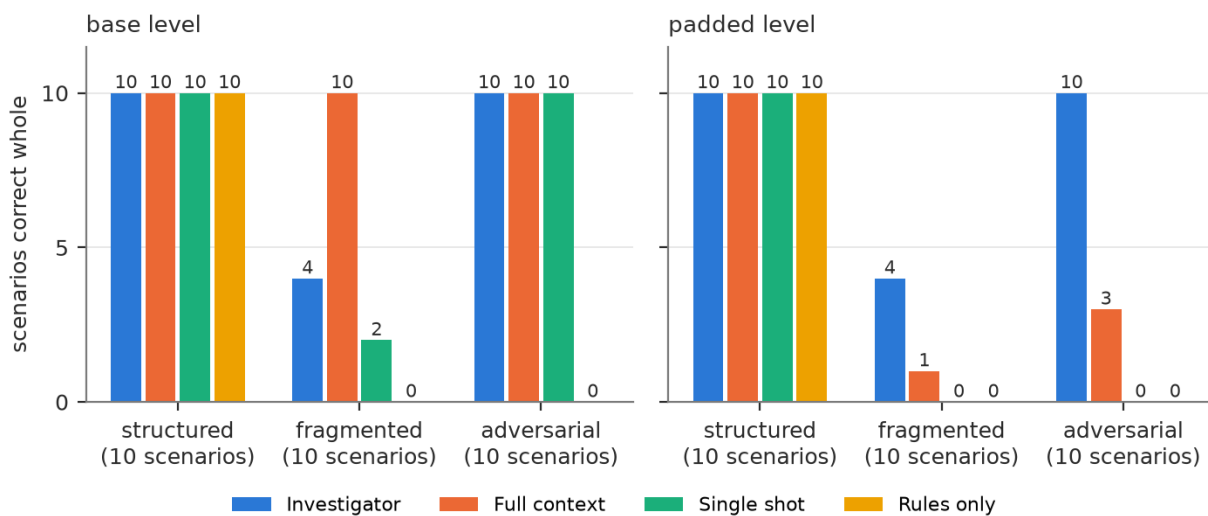
The rules alone

Rules only is the measure of what the structured records are worth. At both levels it is correct whole on 10 of 30 scenarios: all 10 of the structured tier and none of the other twenty. That is its observed performance, and it is also the design intent, since the structured tier was built to need no prose and the other two to need it. On 5 of the twenty it misses, it reports nothing at all, because the prefetch holds no impact for the rules to work from. On the rest it is confident and wrong: it assigns cover where the truth is that nobody qualified exists, or calls a candidate viable when a comment says otherwise, because nothing it read says otherwise. Every claim rules only makes is reproduced by its own reads. This is the distance between grounded and correct described in chapter 2, and it is also the room any model-reading system has: 20 of 30 scenarios at each level.

One retrieval, one call

At the base level, single shot is correct whole on 22 of 30. It keeps the structured tier, takes all 10 of the adversarial tier, and finds 2 of the fragmented ten. One query over the leaver's component names reaches the documents the adversarial scenarios turn on, a stale record beside a newer one, a need with no holder. Of the 31 prose facts the fragmented and adversarial scenarios need, its query returned 23 and the model stated every one it was given; the rest sat in documents the query did not name.

The investigator at the base level is correct whole on 24: the structured ten, the adversarial ten, and 4 of the fragmented. Against single shot that is a paired difference of 0.00 to 0.17, with the lower end at zero, so a difference of zero is not excluded. At this corpus level, where everything a scenario needs is a short search away, the loop's extra reads buy two scenarios and the interval does not separate the two systems.



Scenarios correct whole by tier, system and corpus level. The structured tier needs no prose and every system holds it; the two prose tiers are where the systems differ, and where the padded level moves the one-call readers.

What a model adds, read by tier

Reading the tier figure column by column: the structured tier is ten for every system at every level, so the model adds nothing there. The adversarial tier goes from none under the rules to ten for every model-reading system at the base level: the traps are findable, and finding them is a matter of reading the document that holds the contradiction or the absence. The fragmented tier is the hard column. The rules find none of it, one retrieval finds 2, the loop finds 4, and only the call shown every document finds all ten. That last column is the subject of chapter 4, because at the padded level it is the one that changes.

At the padded level single shot falls to 10, the same count as rules only and on the same scenarios. 25 of 30 of its calls stop at the output maximum; its query returned 10 of the needed facts and the model stated none of them, so its contribution at this level is exactly nothing and the rules' report stands alone. The investigator's counts do not move. The model's contribution, then, has three parts: ten scenarios of trap-finding that any reading which finishes its call buys, a few scenarios of scattered prose that depend on how the reading is done, and a dependence on the call finishing, which chapter 4 measures.

When the loop helps, and when full context does better

The central comparison in this report is between the investigator and a system that reads everything in one call. At the base level the single call wins. At the padded level it stops. This chapter gives both results, the mechanism the records show for the second, and what the experiment can and cannot say about why.

At the base level, the single call is correct on everything

With no filler in the corpus, full context is correct whole on 30 of 30 scenarios. The investigator is correct whole on 24. The paired difference, investigator less full context, is -0.30 to -0.10: the single call passes six scenarios that the loop fails, and the loop passes none that the single call fails (both, loop alone, single call alone, neither: 24, 0, 6, 0). This is a descriptive comparison, not the registered one, and it is stated first because it is the half of the result a reader could otherwise miss. When the whole corpus fits one request with room to answer, a model shown all of it and asked once is the best reader at this level.

Why the corpus was padded

That base-level result was foreseen, and it is the reason the benchmark has a padded level. Before the harness, or any model-reading system, was built, a probe on a throwaway world put the whole corpus, about ten thousand tokens, in front of one call to the development model. The call scored at the ceiling. On a corpus that small a loop could at best tie the single call, so the comparison the benchmark exists for had no room above the baseline. Padding the same world to about a hundred thousand tokens brought the same call down to stating two thirds of the planted facts with inventions among them. Those are preliminary numbers from one world and one model, recorded as a direction and not a result. The direction was that a second, larger workload was needed beside the small one; where a loop earns its place was not tested.

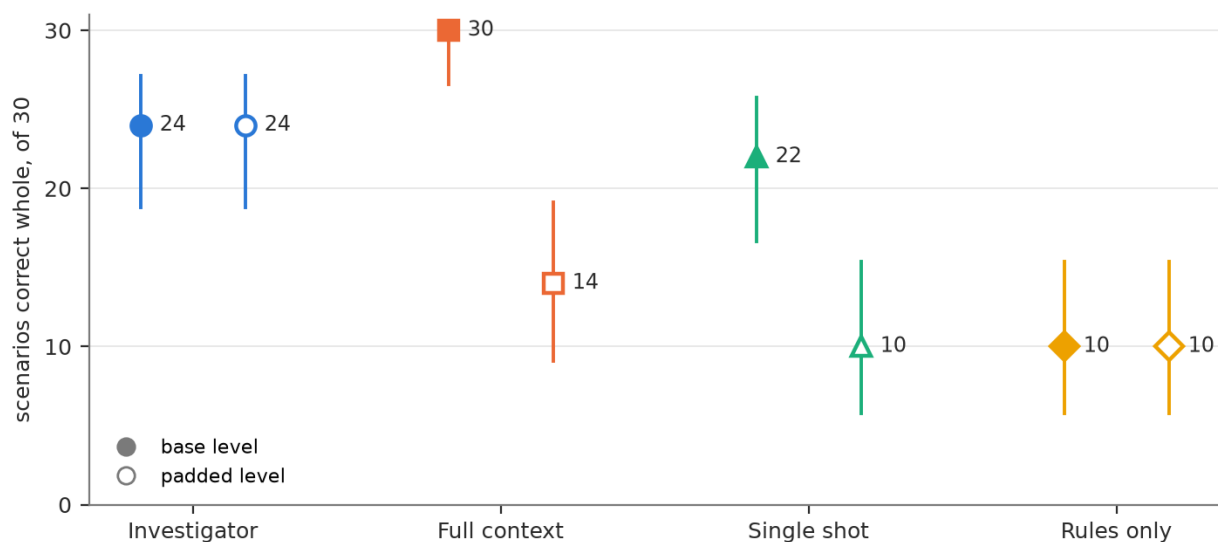
So the measurement world carries the filler pool described in chapter 1, and the base level keeps the small regime while the padded level adds a larger one. The padded level's size was chosen by a rule written before the calibration runs: the smallest of three candidate paddings at which full context is not correct whole on at least one calibration scenario of a development world, never enlarged afterwards. The rule as written said "on every calibration scenario", which no candidate could satisfy, and the freeze read it as "at least one" and said so. Full context graded the same at all three candidates, so the rule chose the smallest, a full-context request of about 40,000 tokens: 160 filler documents, registered as 42,390 to 42,642 input tokens against the base level's 10,359 to 10,611. The padded level was chosen so that the single-call baseline would fail on a development world, before the measurement world existed, and the calibration that chose it had six scenarios.

At the padded level, the single call stops

At the padded level full context is correct whole on 14 of 30. Its level contrast, padded less base, is -0.63 to -0.40, and the loss is entirely on the two prose tiers. The exports record the mechanism directly: 24 of 30 of its calls ended at the registered output maximum of 8,192 tokens with the fact batch cut off. That maximum is the output setting the systems were developed under, carried into the registration unchanged; it bounds what a call may write, not what it may read. What the call is asked to write explains why it matters. The one-call reader is instructed to state every fact of four kinds that the supplied text asserts, each with its exact quote, in one call, so what it produces is an inventory of facts for the rules, not the coverage report, and padding grows the text that inventory is drawn from. Exhaustive extraction over more text is an output-volume problem under a fixed single-call allowance. A call that stops inside a tool use returns its arguments as an empty object, so none of the facts the model had begun to state survive. Single shot shows the same pattern: 22 at base and 10 at padded, with 25 of 30 of its padded calls stopping at the same wall; its level contrast is -0.50 to -0.33.

The stopped calls and the failed scenarios overlap but are not the same set. Of full context's sixteen padded failures, 15 had a stopped call and 1 did not; 9 stopped calls belonged to scenarios that passed anyway. Of single shot's twenty failures, 16 had a stopped call and 4 did not. So most padded failures coincided with a cut-off call and some did not, and a cut-off call was not a failure where the scenario needed nothing it was cut off from. The malformed fact batch is the concrete mechanism; the overlap alone does not say how many scenarios a larger allowance would have passed.

The obvious objection was tested before the runs. A task-relevant instruction, asking only for facts about the person on leave, was probed against the exhaustive one on the calibration world: it nearly ended the stops, none of six full-context calls and one of six single-shot calls, and restored little of what was missed, trading saturation for under-statement, so the registration kept the exhaustive instruction for both one-call readers. That probe is development evidence and sits outside the reported comparison.



Scenarios correct whole out of thirty for each system at each corpus level, with Wilson intervals at 95 percent. Filled marks are the base level, hollow marks the padded level.

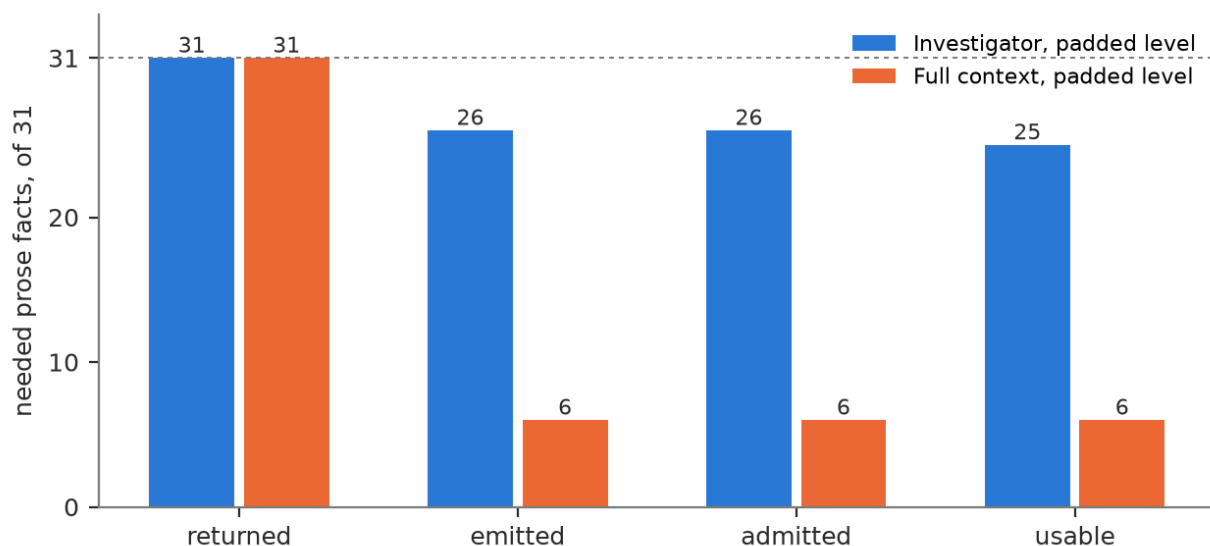
The investigator is correct whole on 24 at the padded level, as at the base. Its level contrast is zero in every resample (24, 0, 0, 6 paired), and the same six scenarios fail at both levels. The registered primary comparison, the investigator against full context at the padded level, is 24 against 14: an observed difference of 33 percentage points of scenarios, with a paired interval of 20 to 47 points at 95 percent. Both pass fourteen scenarios, the investigator alone passes ten, full context alone passes none, neither

passes six (14, 10, 0, 6). The interval excludes zero. Against single shot at the padded level the investigator's difference is 0.37 to 0.57.

The preceding paragraphs make three kinds of statement, and the report keeps them apart. The counts and the interval are what was observed. The stopped calls are recorded evidence of a mechanism: the one-call readers' fact inventories did not fit the output budget the registration gave them. The experiment cannot establish anything about input. It does not show that the model could not read a 40,000-token request, and the freeze ruled, before any run, that output saturation is reported as such and never as evidence of an input limit. A one-call reader with a larger output allowance, or one that finalized in batches, would face a different comparison, and this report makes no claim about it.

Where the investigator's loss sits

The six scenarios the investigator misses are the most informative result in the evaluation. All six are fragmented-tier scenarios, and the mechanism measure says where in the pipeline the facts they need are lost. It follows the 31 prose facts that the required rows of twenty scenarios depend on through four recorded stages, returned, emitted, admitted, usable: whether the document carrying the fact (its *carrier*) was returned to the run, whether the model emitted the fact, whether the record admitted it, and whether it was usable by the rules.



Needed prose facts, of 31, reaching each stage of the recorded pipeline for the investigator and for full context at the padded level. The bars show where facts stop appearing; they do not say why.

For the investigator at the padded level the stages read 31, 26, 26, 25. Every needed carrier was returned, so retrieval is complete and the losses occur after it. By subtraction, five facts were returned and not emitted, and one was emitted and admitted but not usable by the rules. Six missing usable facts and six failed scenarios are different units, and equal totals do not establish a one-to-one match; what the funnel supports is that the five omissions sit at the moment the model is asked to state what it read, a finalization question, and that the one usability loss needs its own explanation. That is the first thing the next design pass has to look at. For full context at the padded level the same stages read 31, 6, 6, 6: every carrier was in the request, and almost nothing was stated, which is the stopped call of the previous section seen fact by fact.

The investigator holds fourteen of the twenty scenarios that any model-reading system can win, at both corpus levels. That invariance is one world's finding.

What the loop earns

A comparison of answer quality alone would flatter whichever system reads most. This chapter puts cost beside quality. It first states what was fixed before any reported run, since costs mean something only relative to the budgets the systems ran with.

Fixed before the runs

The registration was frozen before the measurement world was drawn. It fixed each system's budget as a cap on calls and on tokens, derived from a calibration on a development world: twice the call count of the largest completed calibration run, with a floor of eight, and twice its counted tokens rounded up to ten thousand. The investigator may make 14 calls and spend 200,000 tokens; single shot 8 and 50,000; full context 8 and 191,808, its token cap being the model's window less the output maximum; rules only, which calls no model, 8 and 30,000 as the floor the cap rule sets. Every model call in every system runs under the same output maximum of 8,192 tokens. A run that reaches its call cap or its token cap is graded on what it has and reported as capped. None of the 240 reported runs reached either; what many one-call runs reached is the per-call output maximum, which chapter 4 describes and which caps nothing at the run's level.

The registration also fixed the money: \$45 expected and \$90 as the ceiling, cumulative over the milestone, with the measurement phase itself forecast by hand at about \$12 from the calibration's per-run figures. It also fixed what counts: one run per scenario; a retry only after an infrastructure failure, never after a defect; and a counted attempt chosen by a rule over the run's own log, not by anyone's judgment. Every run closed on its first attempt, so the retry rule was never exercised.

What the measurement cost

The 240 runs cost \$9.9594 in model calls, priced from the provider's published rates by the harness itself; a cost with a missing counter is recorded as incomplete and never priced as zero, and no cell here has one. That figure covers the measurement workload only. Development runs, the calibration, and the generation of the worlds were paid separately, and no single reconciled figure exists for them; chapter 6 records this as a gap.

System and level	Cost of the cell	Per run	Calls per run	Input tokens per run	Output tokens per run	Correct whole
Investigator, base	\$2.2125	\$0.0738	4 to 12	39,678 to 159,172	698 to 3,433	24
Investigator, padded	\$2.1773	\$0.0726	4 to 9	39,678 to 115,610	698 to 2,956	24
Full context, base	\$0.8166	\$0.0272	1	10,517 to 10,877	2,704 to 2,838	30
Full context, padded	\$2.5618	\$0.0854	1	42,232 to 42,592	1,541 to 8,192	14
Single shot, base	\$0.5178	\$0.0173	1	8,755 to 9,765	780 to 1,691	22
Single shot, padded	\$1.6735	\$0.0558	1	12,162 to 12,809	657 to 8,192	10
Rules only, both levels	\$0	\$0	0	0	0	10

Per-run cost is the cell's total divided by its thirty runs. The token columns give the lowest and highest run in the cell; the investigator's figures are summed over its calls. Output is the axis on which the systems are least alike: over its thirty padded runs the investigator wrote 47,806 output tokens over 4 to 9 calls a run, against full context's 211,989 in one call each, because a loop states the facts its searches returned where the one call must state everything it was shown.



Mean model cost per run against scenarios correct whole, one mark per system and level. Filled marks are the base level, hollow marks the padded level; rules only sits at zero cost.

Reading cost against quality

At the base level the investigator costs more than the single call shown everything and is correct on fewer scenarios: 2.7 times the cost per run, \$0.0738 against \$0.0272. If the corpus fits one request with room to answer, there is no case for the loop in this evaluation, on quality or on cost.

At the padded level the single call shown everything is both more expensive and less accurate than the investigator. It now costs \$0.0854 a run, more than the investigator's \$0.0726; the report does not decompose that rise between the level's input and the call's output. It is also correct on 14 scenarios to the investigator's 24. The investigator's cost barely moved between levels; it reads what its searches return and the filler is answer-neutral, though how many filler documents its searches returned was not measured. Single shot stays the cheapest model-reading system at \$0.0558 a run and, at this level, buys nothing over the rules.

What the loop earns, then, is stated conditionally, and the condition is the output budget. On a corpus that fits one request with room to answer, it earns nothing and costs more. On the padded corpus, where the one-call readers' fact inventories outgrow the output allowance every system ran under, it holds its quality and its cost while they lose theirs. Whether a real organization's records resemble either corpus is outside this evaluation, and whether the loop's advantage survives a one-call reader with a larger output allowance is a question it did not pose.

What this does not show

This chapter collects the limits in one place, starting with those that most change what a reader may conclude.

What was registered, and what was found

The evaluation carries the label *reported*: it ran once, under a registration bound to the measurement world, and the primary comparison was the investigator against full context at the padded level on correct whole. The registration reads a comparison as an interval and never as a test of superiority, and the primary interval, 0.20 to 0.47, excludes zero. No other comparison carries a registered claim; the base-level comparison that chapters 4 and 5 draw on is descriptive and is reported as such. The evaluator that produced the evaluation artifact ran at a commit that differs by two files from the commit at which the registration was bound; both are fixes to the evaluator's cells and entryptpoint made after the bind. The artifact names both files, and the registration's bytes are the bound ones.

One world, one repeat, one model

The intervals describe variation across scenarios of this generator in one organization of about thirty people, with one run per scenario. They say nothing about run-to-run variability. During development, a repeat under identical prompts at temperature zero reproduced its runs exactly; that repeat covered a handful of runs and is not part of the reported evaluation. In every system measured, one model, Claude Haiku 4.5, played every role at temperature zero. Nothing here says how a different model, or the same model at another temperature, would rank. A change to the registration after these results, which a changed system would need, makes later evaluations on this world exploratory by the registration's own rule; confirmation needs a world from a new seed.

The thirty scenarios fall into eleven classes within the three tiers, between one and four scenarios each. The bootstrap resamples whole scenarios within tiers and does not account for clustering by class, so scenarios of one class that share a weakness count as independent evidence of it; the six scenarios the investigator misses are spread over three classes.

The padded level was chosen so the baseline would fail

The systems already differ at the base level. Padding reverses the ranking of the investigator and full context, and the padded size was the smallest candidate at which full context already failed on at least one calibration scenario of a development world, chosen before the measurement world existed. The calibration's denominator was six scenarios. The primary comparison therefore sits at a level chosen to expose the single call's limit, and a reader should treat the padded result as a finding about that regime only.

Output saturation is the mechanism, and only that

The one-call readers' fact inventories did not fit the output budget they ran under, and the stopped calls are recorded in the exports, call by call. That budget, 8,192 output tokens per call, is the setting the systems were developed under and was registered unchanged for every system; nothing in the evaluation varies it. Most padded failures coincided with a cut-off call and some did not: of full context's sixteen failures, 1 had no stopped call, and of single shot's twenty, 4 did not. The stop is not evidence of an input-context limit, and the report makes no claim that the model could not read the padded request. A one-call reader with a larger output allowance, or a batched finalization, would face a different comparison. The investigator's level invariance, the same count and the same six misses at both levels, is one world's finding; a world padded past this level's request size would test it again.

The two kinds of system were not tuned alike. The investigator's prompts went through fifteen rounds of development runs on the first development world, each revised against named failures; the one-call readers' extraction instruction was settled in four calibration rounds, with one alternative probed. A reader who suspects the one-call readers could be prompted past their stop is entitled to; this evaluation did not try.

The registered ranges are envelopes, not bounds

The registration recorded the full-context request size per level from the six calibration scenarios: 10,359 to 10,611 tokens at base. The thirty reported base-level runs measured 10,517 to 10,877, above the registered upper end, and the padded runs measured 42,232 to 42,592 against the registered 42,390 to 42,642, an overlap rather than containment. The scenario sets differ, so neither is an error. Still, a range recorded from six calibration scenarios need not contain the values observed on thirty.

What the audit establishes

The measurement world's acceptance rested on the validator's verdict over every record and on spot checks: three scenarios traced end to end by the audit method and five filler documents read. Spot checks establish no defect rate; the untraced scenarios rest on the validator and on the generator's class contracts. Two separate limits apply. The embargo withholds the measurement world from readers until the last measurement on it. Independently, twenty-seven of its scenarios, the thirty less the three traced, have no trace. The golden world's audit was fuller and its findings are public, though its record stays with the truth because it holds answer keys; it checks the key against the rules as coded and against a human reading, and no independent oracle was used.

The prose is near-template

A text enters the world only when a second model can extract exactly the planted fact from it, so the prose any system reads is close to template sentences with names in them. The anchor guard that refuses a stated fact whose quote does not name its subject is partly a property of that generator. On prose that says "she", the same guard would cost recall. This limits both realism and measurement. The results test reading what is stated and say nothing about inference.

Not run, not built

Four outage cells for full context were registered, and the freeze decided not to run them. Four supporting comparisons were registered and decided not built before any result was seen: vector retrieval against full-text retrieval, a multi-agent layout, the primary contrast under a second model, and a model-authored report against rule composition. A retrieval arm built later is exploratory on this world. The outage conditions themselves were exercised once each on a development scenario and are development evidence only.

Recorded gaps

The model spend is the harness's own pricing from the provider's published rates and has not been reconciled against the provider's bill for the two days of the runs. No single figure exists for the milestone's total model spend including development. Run times are recorded per cell in the evaluation artifact and not reported here. The harness publishes its own inventory of attempts, and the evaluator checks its listing against that inventory. This check finds implementation mistakes. It does not prove that every admission was logged or that nothing was spent outside the ledger.

Reproducing the numbers

Every number in the results tables, the figure captions and the results chapters is a pin: a value the build reads from a record in the project's repository at the moment the PDF is rendered, so the prose cannot drift from the data. The statistical figures and the costs are read from the committed public results file. That file is a projection of the single evaluation artifact and is recorded by the artifact's digest; before the file was committed, an offline acceptance command regenerated it and compared the result byte for byte. The caps, levels, output maximum and budget are read from the bound registration file. The operational counts (calls stopped at the output maximum, and input tokens per run) and the world's audit counts are read from the recorded findings by exact text; the script behind the operational counts needs the run exports and is not rerun at build, so the build checks those against the record and does not recompute them. A few numbers in the prose about the instrument and the calibration, such as the probe's counts and the forty seeds of the runtime-truth comparison, are quoted from the cited records, and a number derived by hand says so where it appears.

Handle	Value
Repository	github.com/arda-basarici/leave-impact-agent
Portfolio	ardabasarici.dev
Committed results file	src/leaveimpact/results/m2-reported.json
Evaluation artifact, SHA-256	fc0607bb65c5535de2280268a785029e866e66cdb9418797e1cc975a86ea0721
Measurement world, version	2e6705ba7d11095a9b797abe08495c5497a7c5fe560fa587868763f11192341c
Registration, digest	acb11ce1b1b2856298bd95c941a79c4f1d29fc7a03bb243be9b65ba58df8f4c6
Registration, bound at commit	546e1badbfdc137edbacfb2beab40aadf58f5dde
Evaluator, commit	8acbc66ae5639df5d1d4471f5d6d4abc0d936ae8
Registration and protocol	preregistration/registration.json, preregistration/PROTOCOL.md
Findings	probes/FINDINGS.md, section phase-c
This report's chapters, pins and figures	report/chapters/, report/pins.py, report/figures/

The committed results file is validated by the test suite on every change, and the headline figures this report cites are asserted there against the values the findings record. The measurement world's contents stay embargoed. The golden world's seed and recipe are open and its audit findings are published; its scenario keys and its audit record are released with the project's demo, not before.