Technical report

A governed SQL data agent: what it is built from, what the levers measure, and where the claim stops. Every figure names its run, its grader and its knowledge setting.

DATED	SECTIONS	SUITES	STATUS
2026-09-03	15	BIRD · Spider	v1

A governed SQL data agent: what it is built from, what the levers measure, and where the claim stops. Every figure names its run, its grader and its knowledge setting.

Draft, 2026-09-03; §3 restated on beacon v7, §6 re-graded 2026-09-06. Every number here is pinned to an artifact and a convention; §7 says which.

> **§3 now carries beacon v7's grading of the same stored runs; §6 still carries the 2026-09-06 re-grade.** The runs are unchanged — deferral rates, never graded against gold, are identical under either — but the grader is not, and §7.3b makes the grader part of what a number must name. §6's verifier ladder has not been replayed under beacon v7, and is flagged where it appears.

---

1. What mnemiq is, and the claim it makes

mnemiq answers natural-language questions against a governed SQL database. The claim is not that it writes better SQL than a frontier model. It is narrower and more useful:

> **Against a semantic contract, GENERATION can be moved to a 14B open-weights model without losing accuracy on BIRD-shaped work — and the system can refuse rather than guess when it cannot.**

Both halves are measured below, and so is the boundary where the claim stops holding (§5).

Say what was and was not localised. In every measured arm the contract itself was built by **hosted gpt-5.5**, held constant, with only generation on the local model. So the result is "*generation localises*", which is the expensive, per-query, data-touching half — not "*mnemiq runs on a 14B*". Enrichment is a one-off per source and is not measured here as a local workload. Anyone reading this as a fully air-gapped story is reading more than the evidence supports,.

The engine is **18,212 lines of Python** across 139 tracked files. It is designed against a separate trust plane, which holds and enforces the policy the engine consults.

1a. The vocabulary — what kind of thing each part is

This report describes four kinds of thing that are easy to confuse because **all of them move the number**. That is why an earlier draft called them all *levers*, and why that word had to go: it sorted by effect, and since everything has an effect it grouped everything. Two sections later the same word meant both the intervention ("*no lever*") and the route it worked through ("*the deferral lever*").

The sort here is **what changed**, not what it changed.

Techniques — what we built. Four tiers, and the test is what happens when one fails:

TIER	TEST	EXAMPLES
Enforcement	failing it produces a **refusal**	access-scoped retrieval, `check_access`, RLS/CLS policy rewriting, the shape check, the Oracle read-only gate, fail-closed defaults
Components	can be removed and the run repeated	the seven enrichment phases (§2), retrieval, generation, the decider, each verifier layer, the transpiler, lineage and the audit record, the agreement gate
Methods	a named strategy, tagged by stage	*enrichment*: deterministic-first ordering, grounded-or-bare · *generation*: constrained decoding, assertive prompting, self-consistency · *training*: QLoRA
Hyperparameters	it has a value	`retrieval_k`, `verify_threshold`

Knowledge — content, not code, and it appears **twice** in the flow, which is worth separating because only one half is ours. **Sources** are what a deployment brings: the database itself, and optionally governed certified records, a code data dictionary, ontology records, verified examples (§1b enumerates them). **The semantic contract** (§2) is what enrichment makes of them — meanings, relationships, views, dimensions, metrics, the job log. Sources are an input to stage 1; the contract is the output of stage 2 and the input to retrieval. Both vary per deployment with no code changing, and conflating them hides that the first is the customer's and the second is ours.

The field's own sub-taxonomy for knowledge fits the contract's contents almost exactly: *schema* knowledge (`enrich_structural`, `enrich_semantic`), *value* knowledge (`ground_codes`, `apply_certified`), *domain* knowledge (ontology grounding, `apply_dictionary`), *query-pattern* knowledge (`enrich_facts`, `enrich_examples`), and *calculation* knowledge (metrics).

Content is not a lesser category for not being code — but the size of its effect is not something this report measures, and the one number available is not ours. BIRD's published with/without- knowledge gap is **20 points** (§7, rule 1a); §3's deferral fixes are **+28 to +35** from the 14B's 17.0% with no generation method — which is an *enriched* arm, so those gains are not enrichment-inclusive — and the increments among them are **−0.6** (model size) and **+1.4** (self-consistency over guided decoding). So knowledge lands between those two groups — **below every deferral fix, an order of magnitude above every refinement**. Read that as scale, not as a measurement: the 20 points is GPT-4 on a different protocol, not an arm we ran. **The honest statement is that the knowledge dimension is large enough to matter and is absent from §3's table.**

Model — the weights an endpoint serves: 7B / 14B / 32B, gpt-5.5, a QLoRA checkpoint. Procured, or derived by fine-tuning — either way not something this codebase authors. QLoRA the *method* is a technique; the checkpoint it emits is a model.

Arms — a named combination of the above, measured. This one exists only inside a results table: `14B + guided`, `14B + guided + SC5`, and **no generation method** for an arm with none applied. Not "no method": every §3 arm was enriched, and enrichment has methods of its own. Not "baseline" either — this report already spends that word on the published reference figures ("Against published baselines", §3). An arm is not something you can deploy.

Two consequences worth stating, because both are easy to get backwards:

- **A setting is not a peer of a technique, it is how you configure one.** `verify_judge=False` is the *ablation of a component*, which is why the `verify_*` booleans resist being their own tier.
- **"Not a technique" is not a demotion.** Model and Knowledge sit outside the umbrella because we did not author them, not because they matter less.

1b. The flow, end to end

Nine stages. §2 onward takes them in order and in depth; this is the shape they make, and which of them a deployment can vary.

#	STAGE	CONSUMES → PRODUCES	VARIES?
1	**Knowledge sources**	the database (schema, values) + what an operator supplies	**yes** — what you have
2	**Enrichment** (§2)	sources → the semantic contract, in seven fixed-order phases	**yes** — which phases run
3	**The semantic contract**	content-addressed by `content_version`	it *is* the output of 2
4	**Retrieval** (§2a)	contract + identity → a `ContextPacket` of `k` cards	**yes** — `retrieval_k`, embedder
5	**Generation** (§2a)	packet + question → candidate SQL	**yes** — model, decoding, prompting, self-consistency
6	**The decider** (§2b)	candidate → approved or refused: shape, authorization, RLS/CLS rewrite, transpile, `EXPLAIN`	**no — fixed**
7	**Execution**	approved SQL → rows, against the source	**no — fixed**
8	**Verification** (§2c)	rows + question → answer or deferral	**yes** — which layers, what threshold
9	**Answer and trace** (§2d)	lineage, the audit record, the disclosure tier	**no — fixed**

Stages 5 and 6 are a loop, and that is the design. `plan_query` is *propose, decide, repair*: a refusal the corrector can act on goes back to the model with the decider's reason attached, up to **three attempts**. The loop is closed by a deterministic checker, not by the model reviewing itself — which is why a repair is a narrowing rather than a retry. §2b, "Refusals that are repairs".

What an operator can actually supply at stage 1, today: **governed certified records** pulled from the trust plane's record store, which feed the `apply_certified` phase; a code data dictionary (`dictionary_path`); ontology records (`ontology_records_path`); and the optional table-facts and verified-

example phases. Not binding suggestions — `binding_suggestions_path` is where enrichment *writes* them. **Query logs and prose documentation — a wiki, a Confluence space — are not ingested.** They are the obvious next sources and the engine reads neither; anything claiming otherwise is describing a plan.

Where knowledge varies in this report, and where it does not. The **ten-arm table** of §3 holds it constant — its own preamble and §7 rule 1a say so, and that is exactly why that table cannot price enrichment. Three tables here *do* vary it — §3's published baselines, and §4b's BIRD and Spider 1 tables:

- §3's **published-baselines** comparison is enriched against unenriched. That is the difference it exists to measure.
- §4b runs our own `tier1 structural` against `tier2 semantic` — deterministic enrichment, then deterministic plus the LLM semantic layer — on two suites and two metrics.

§4b works those four comparisons through and finds the semantic layer's effect unresolved — one separation, three overlaps, and a direction that reverses between suites. The numbers stay there: they are beacon's rates on beacon's denominators, which §4b says must never be tabled beside §3's and §4's.

Certified records — the input a clean knowledge sweep would move — are varied only in a private-corpus ablation, which is not in this report.

No competitor's knowledge is established to vary either, which is not the same as none varying. Cortex ran one arm with its Verified Query Repository empty, a condition it was given. Genie ran two, `pre-FK` and `post-FK`, and §4b records that beacon kept those labels without their definitions — so *whether* that pair moved a knowledge condition is unknown here, and unknown supports "not established", not "nothing changed".

Five of the nine can be searched; three are fixed. (The remaining one is row 3, the contract itself — not a stage you tune but the artifact stage 2 emits.) A customer engagement sweeping knowledge, enrichment, retrieval, model+generation and verification reaches five — and **the decider, execution and the trace are not in that space at all**, which is why §1a puts Enforcement outside the things you tune.

Fixed does not mean immutable, and the distinction is worth being exact about. All three *are* configured: `write_enabled` opens the governed write path and `authz_path` supplies the grants (stages 6 and 7), and the trace carries three separate disclosure tiers — answer text, identity detail and result rows (stage 9). Every one of them is **default-closed**, set by **policy**, and decided by whoever owns the data — not searched for a better score. No knob in an accuracy sweep reaches any of them.

2. How enrichment works — and why the ORDER is the point

Enrichment turns a bare schema into a **semantic contract**: columns with meanings, relationships, views, dimensions, metrics, and a job log recording what was attempted and what failed. The contract is content-addressed (`content_version`) so a replica can tell whether it holds the same one.

The pipeline runs in a fixed order, and the order carries the design:

#	PHASE	WHAT IT DOES	LLM?
1	`enrich_structural`	schema discovery and per-column profiling — distincts, nulls, types, relationships, views	**no**
2	`ground_codes`	resolves coded values against evidence already in the database	**no**
3	`apply_certified`	applies certified code schemes where one is declared	**no**
4	ontology grounding	embeds an ontology corpus and matches definitions; **degrades to local** if no embedder	embeddings
5	`enrich_semantic`	the LLM proposes meanings for what remains, with protected terms held back	**yes**
6	`apply_dictionary`	the operator's data dictionary — **overrides everything above**	**no**
7	`enrich_facts`, `enrich_examples`	optional table facts and worked examples	yes

Deterministic first, LLM in the middle, operator last. The LLM never gets first say on a column whose meaning is already recoverable from the data, a certified scheme, or an ontology; and whatever it does say, a human dictionary entry overrides. That ordering is what makes the contract auditable: for any column you can ask which layer decided its meaning, and the answer is a rank, not a guess.

The job log is the other half. A column that could not be profiled is recorded as **failed with a cause**, not silently as an absence — a distinction the codebase had to learn twice (§6).

2a. Retrieval and generation — what the model sees, and who has the last word

What the model is shown

Retrieval is **hybrid and scoped to the identity's grants before anything is ranked**. An object the caller may not access is "*never scored, never ranked, never counted and never returned*" — not filtered out at the end, but absent from the candidate set.

The reason it happens first rather than last is that **metadata is itself confidential**. A table named `layoff_plans` discloses by its mere existence; a packet that ranked it and then dropped it would have leaked through the

ranking, and a count of what was considered would leak it too.

That was true of tables and, until it was found, false of columns: cards are rendered at enrich time from the whole snapshot. Cards are now re-rendered against the identity's column policy before entering the packet.

Ranking is scoped to tables, not to columns, and the distinction matters. Both retrieval arms — BM25 and the embedding cosine — score the *stored* enrich-time card, which carries no policy. The grant scope restricts which objects are candidates, so a denied table never ranks at all. Membership of the candidate set is therefore decided by the grant scope, and ranking operates only within what that scope admits.

What the model receives is a `ContextPacket`:

FIELD	WHAT IT CARRIES
`cards`	the schema cards, re-rendered against this identity's column policy
`definitions`, `metrics`, `dimensions`	the certified business layer — what terms mean here
`concepts`	candidate codes resolved against scheme-bound columns
`examples`	worked examples, where enrichment produced them
`history`	prior turns, scoped to the same grant set
`grant_fingerprint`	a hash of the grants this packet was built under
`enrichment_version`	which contract produced it

The last two are not decoration. The fingerprint enters the **cache key**, alongside the canonical plan and the enrichment version, and the code is explicit about why: "*the grant fingerprint is not an optimization — it is the authorization boundary.*" Two callers with different entitlements asking the same question cannot share a cache entry, so a cached answer can never be served across a grant boundary.

Who has the last word

`plan_query` is the loop: **propose, decide, repair — and defer rather than guess.** The model proposes SQL; `decide` (§2b) has the final say; a refusal the model can act on comes back as feedback and it tries again.

Not every refusal comes back. A logic-lint failure or an ungrounded filter value is returned to the model with the detector's message, because those are repairable mistakes. An *unauthorised reference* is not, and the reason is worth stating exactly as the code does: "*retrying it would be inviting the model to find another route to data this identity may not have.*" Feeding an authorisation failure into a repair loop turns a security control into a search problem, with the model doing the searching.

The corrector is deliberately narrow — "*a constrained, surgical edit: fix exactly the stated problem, change nothing else*" — and problem-agnostic: it is handed a message by whichever detector fired and does not know which one that was.

Many candidates, one answer

In `deep` mode the engine generates **five** candidates, **executes them**, and clusters them by their *result* — the clustering function takes the executed Arrow tables, not the SQL. Two different queries returning the same rows are one cluster. The default selector takes the largest cluster, first on ties.

The deferral gate has two halves and the second is the interesting one. Where at least one candidate executed, the engine answers only if the largest cluster clears `min_agreement` (0.6 in `deep`) *and* **all five candidates executed**. Four of five executing with all four agreeing — agreement 1.0, far above the threshold — still defers.

The reason is measured, not cautious: "*a reduced set never passes — unanimity among survivors is survival bias, not agreement*", and the numbers behind it are 4-of-4 and 3-of-3 unanimity landing at **11% and 0% correct**. Fragmentation is treated as a verdict on the **question**, reached before any candidate is picked.

Where clusters disagree, a judge may pick between them — under two constraints that bound what it can do:

- **It only ever swaps one executed answer for another. It never writes SQL.** The candidates it chooses among have all already passed `decide` and run.
- **It fails closed to the majority.** An unreadable reply, an out-of-range pick or a dead endpoint falls back to the deterministic choice rather than to the judge's absence.

That is why agreement is a signal the engine records rather than a mechanism it trusts: §3 measures what self-consistency buys **on top of** guided decoding: +1.9 points, taking the residual deferral guided leaves from 10.3% to 7.8%. The table carries no self-consistency-without-guided arm, so it cannot tell you whether the two would reach the same deferrals by different routes.

2b. The decider — how a generated query becomes an approved one

Enrichment (§2) builds the contract. This is what happens to a query written against it, and it is the largest single body of engineering in the repo: `mnemiq/sql/`, **seventeen modules totalling 3,494 lines** (eighteen files; the package `__init__` is empty). Nothing here involves a model. Every step is deterministic, and a query that fails one is refused with a code, not repaired by asking nicely.

`decide()` runs these in order, and **the order is part of the design**. THREE are conditional: step 6 runs only where harvested column values exist, step 8 only where the identity has a non-empty policy, and step 10 only where an adapter was supplied — which is not the default. A deployment with none of the three runs seven of the ten.

#	STEP	WHAT IT ENFORCES	ON FAILURE
1	`check_shape`	one statement, a SELECT, no DDL/DML, **no** `SELECT *`, and a `LIMIT`	refuse
2	`check_access`	every table and column against what this identity may see	refuse
3	`check_cls`	denied columns; masked columns outside a bare projection	refuse
4	`check_views`	a view that reads a row-filtered table	refuse
5	`lint`	SQL that runs and answers wrongly	refuse — **repairable**
6	`check_values`	a filter literal that exists in no row of its column	refuse — **repairable**
7	*lineage*	what the query, **as asked**, reads	recorded
8	`apply_row_and_mask`	injects row filters and column masks	rewrite — or refuse, on a policy-invalid predicate
9	`expand_tables`	federation: `catalog.table` → `catalog.schema.table`	rewrites
10	`prove`	the statement against the real source	refuse

Shape before access, and access before the database. Shape is first because "*a DROP is a DROP regardless of what it names*" — there is no point resolving identifiers in a statement that must not run whatever they resolve to. Access is second for a sharper reason, and it is the one worth taking away:

> The database's own permission error **is not the control**. By the time it fires, the engine has already confirmed to the model that the table exists.

A `permission denied for table salaries` is an answer. It tells a caller the table is there, which is a disclosure the caller was not granted, and it arrives *after* the query left the building. So an unauthorised query never reaches the source; the check that stops it runs here, against the same visibility map the retrieval layer used.

Two locks, and why one is not enough

Access control is enforced twice, at different layers, against the same map:

1. **The model is never shown a table the identity cannot see.** Retrieval scopes the semantic store to the caller's grants, so a forbidden table is absent from the cards the model reads.
2. **`check_access` re-checks every table and column the generated SQL names.**

The second exists because the first is not sufficient, and the reason is specific rather than defensive: "*It can still name one — `users`, `employees`, `salaries` are in every schema it was trained on.*" A model does not need to have been shown a table to guess it. **The first lock removes the suggestion; the second makes acting on the guess useless.**

The same argument applies to the `LIMIT`. It is **injected into the tree, not requested in the prompt**, because "*asking a model to add LIMIT is a request, and rewriting the tree is a guarantee.*"

`SELECT *` is refused outright, and this is the single largest guard in the module (`guard.py`, **751 lines** — nearly twice `lineage.py`, the next largest). The reason is that a star is a promise the engine cannot keep: it expands, at execution time and inside the source, to whatever columns the table has — including the ones this identity was denied. A star nested inside a subquery, a CTE, a set operation or a parenthesised body expands the same way, so the guard recurses through wrappers rather than pattern- matching the top level. The guard's fallback is **fail-closed** — an unrecognised shape is refused rather than passed — which retired the class.

What the policy rewrites, and the one thing it must not disclose

Step 8 injects row filters and column masks. Step 7 — lineage — deliberately runs **before** it, and that ordering is load-bearing in a way that is easy to get backwards.

An injected row filter reads whatever the *policy* names, and the standard shape for that is an entitlements table which no caller is ever granted. Compute lineage after the rewrite and the answer tells its caller which tables their own policy consults. So provenance is read from the query **as asked**: what the engine reads on the policy's behalf is not the caller's lineage, and the audit trace — a different channel, with a different audience — is where that belongs.

Narrowing disclosure is **table-granular on purpose**, and the code is explicit that this is a limit rather than a simplification: column granularity would need the loop to know which projected column each masked source column reaches, and it does not, because a masked column can be consumed by an aggregate and never appear in the output at all. "*Claiming per-column detail the loop cannot support is how a disclosure becomes a lie.*"

Column masking is also refused rather than applied wherever applying it would change the answer: `check_cls` permits a masked column in a bare projection and nowhere else, because masking a column that is filtered or aggregated on silently alters the result.

Refusals that are repairs

Two of the ten steps refuse queries that are **valid SQL the source would happily run** — and would answer wrongly. `lint` catches shapes that are silently incorrect; `check_values` catches a filter literal that appears in no row of the column it filters. Neither is a security control. Both feed the correction loop, which is why `prove` carries the **source's own error message** rather than

a generic sentence: that message "*is what a correction loop reads to repair the SQL, so replacing it with a generic sentence would cost the repair its only evidence.*"

`prove` exists because the contract can be stale and only the source knows the truth — and it asks in the dialect the source speaks, since `EXPLAIN` is not Oracle syntax.

Identifier resolution, which is where the bugs were

Every guard above has to answer one question — *which object does this name refer to?* — and the answer depends on the engine that will run the statement, not the one it was parsed as. Unquoted names fold **up** in Oracle and **down** everywhere else; quoting makes a name case-sensitive in Postgres and Oracle but not in DuckDB or SQLite.

The resolver is a single module (`sql/identifiers.py`) for a reason found the hard way: threading the executing dialect to *some* call sites and not others is worse than threading it to none. In one statement, `check_access` reported a name as a CTE while the audit list reported the same name as a base table read — and a denied column came back approved. Same input, two answers, because two resolvers disagreed about what an identifier meant.

2c. The verifier — the only layer that sees the answer

Everything in §2b runs **before** execution and judges the *query*. The verifier runs **after** and judges the *result*, which is the one thing no amount of static analysis can reach: a query can be authorised, well-formed, grounded in real values, and still return the wrong rows.

It is a cascade of three layers, and the first that fires short-circuits the rest. "On in every mode" below means every mode's default rather than an invariant:

LAYER	WHAT IT CAN SEE	NEEDS A MODEL	DEFAULT
`sanity`	the result table	no	**on in every mode** that verifies at all
`grounding`	the question and the plan SQL	no	**off**
`judge`	question, cards, SQL and a 5-row preview	yes	`deep` only

Why sanity is first and always on

`sanity_check` defers on results that cannot be an answer at all: **zero rows**, a single scalar that is null or NaN, or a table whose every cell is null. That is the whole of it — a page of code with no model and no configuration.

It is first because it is the cheapest and because the class it catches is the largest: "*an empty result presented as an answer is the single biggest catchable wrong-class (~17%).*" An empty result is not a rare pathology; it is

what a wrong join or an over-narrow filter usually produces, and without this layer the engine reports it as a finding.

The measured price is the reason it is on everywhere: §7 puts it at **31 wrong answers caught for 2 correct ones deferred**. Nothing else in the cascade is close to that ratio.

Why grounding is off

`grounding_check` looks for literals the question all but demands appear verbatim in the SQL — quoted phrases and four-digit years — and defers when they are missing, on the theory that a dropped filter is a common and invisible failure.

Its scope is deliberately narrow, and one exclusion shows the reasoning: values containing `/` are dropped, because a model reformats `2012/8/23` to `2012-08-23` and their absence is therefore *not* evidence of anything. The layer only flags what it is confident about, so that a miss is a real signal.

It is still **off by default**, and the measurement is why: 28 caught for **16** correct answers lost. That is a different trade from sanity's 31-for-2, and it is not one to make silently on a customer's behalf. Grounding is a dial position — safety-first — rather than a free default.

The judge, and what it costs to have one

The judge is the only layer that can assess whether the result *answers the question*, and the only one that needs a model. It scores confidence between 0 and 1; the verifier defers below a threshold, which defaults to **0.5** — the cutoff §7's judge row is measured at.

Two properties bound it. It runs **last**, after both deterministic layers have declined to fire, so it is never consulted about a result sanity already rejected. And it is **fail-open**: an unreachable judge scores 1.0 and the answer proceeds.

That last decision is deliberate, and the record is what carries it: the engine reports which layer produced a verdict, including a `judge_unavailable` state, and the wire carries a derived `verified` field — `judged`, `basic` (deterministic layers only) and `unavailable`. `null` means no verifier read the answer at all — which includes deferrals and execution failures, not only modes without a verifier. **The confidence score itself is withheld from the wire**, because a bare `0.62` beside an answer reads as a probability of correctness that has not been earned.

Fail-open remains the behaviour when the judge cannot answer. What changed is that the record stops claiming the answer was checked.

2d. Lineage and the audit record — what an answer says it read

An answer that names the tables it read is more useful than one that does not, and **more dangerous if the naming is wrong**. A bare list is worse than no list at all, because `[]` reads as "*nothing was read*" when the truth may be "*read through something we could not classify.*"

So the list never ships alone. It travels with a **completeness marker** taking one of three values, and the distinction between the last two is the whole design:

VALUE	MEANING	THE CASE THAT PRODUCES IT
`COMPLETE`	every read resolved, **no `exp.Func` node anywhere in the tree**, and the view inventory actually consulted	a call-free projection over base tables, asked with a view inventory in hand
`INCOMPLETE`	reach beyond the list is **demonstrable**	a view — its body is in the inventory, so the engine can show there is more
`UNKNOWN`	completeness could not be **established**	a function — the engine does not know that `all_ssns()` reads anything

`COMPLETE` is rare, and the report would mislead by implying otherwise. The test is *any* `exp.Func` node in the tree as **sqlglot** parses it — the parser the whole decider is built on — and that is broader than "called a function" sounds. `CAST` counts. `COALESCE` counts. `CASE` counts, as `Case` and `If`. A bare `DATE '2024-01-01'` literal parses as a `Cast` and counts too. (Plain arithmetic and `LIMIT` do not.) So a filter on a date forfeits the marker with no function visible in the text. Any such node removes `COMPLETE` — `SELECT count(*), upper(region) FROM claim` cannot record `COMPLETE`, and a test pins exactly that: "*an aggregate is not an unaccounted object... but a call is still a call.*" (A call does not by itself produce `UNKNOWN`; a reaching view still wins, as below. What a call removes is the strong claim.)

Measured on BIRD mini-dev: of the statements in `gpt55-duckdb-pg.regraded-2026-09-06.jsonl`, **438 of 481 (91.1%) contain a function call** and therefore cannot record `COMPLETE`; in `minidev-pg-14b-guided-fixed.regraded-2026-09-06.jsonl` it is 336 of 437 (76.9%). Both artifacts are named because they must be: `minidev-pg-14b-qlora.regraded-2026-09-06.jsonl` also holds 437 statements, of which 343 contain a call — a matching denominator with a different numerator, which is exactly the confusion §7.3 exists to prevent.

These are upper bounds on `COMPLETE`, not the rate itself. A call-free statement can still miss the marker for reasons this section has not enumerated — an unavailable view inventory, an inventory that was never asked for (the default), or an unresolved scope each append a reason, and any reason forfeits `COMPLETE`. So the true `COMPLETE` rate is at most 8.9% and 23.1% on these two arms, and in a deployment that never supplies a view inventory it is zero.

That is conservatism by design rather than a gap: the engine will not certify it saw everything a `count()` might reach, and would rather be unhelpful than confidently wrong. An auditor should read `COMPLETE` as a strong claim rarely made, not as the normal state of a healthy trace.

`INCOMPLETE` says "*there is more and here is why*". `UNKNOWN` says "*we cannot tell*". Collapsing them would let a function call be reported with the confidence of a view, and functions are the case where the engine genuinely cannot see through. Ordering follows: `UNKNOWN` is not allowed to mask a demonstrable `INCOMPLETE`, because a statement that reads a view *and* calls an unclassifiable function is more use to an auditor named as incomplete-because-of-the-view than filed as unclear.

Two fields carry the detail, and they are separate because they are **different namespaces**: `unresolved` holds object ids and function names — things in the caller's world — while `reasons` holds this engine's own reason codes. Merged into one array they reach the audit store indistinguishable, and a consumer rendering objects would display `scope-unresolved` as though it were a table.

The split is not airtight, and the report should say so rather than sell it as one: `unresolved` also admits a closed set of four engine-prefixed entries — `unconfirmed-identity:…`, `unmodelled-source:…`, `table-function:…` and `ambiguous-view:…` — plus a bare `unnameable-function`. **Three of the four carry caller-derived names behind the prefix** — `unconfirmed-identity:claim` a table, `ambiguous-view:V` a view the caller named, `table-function:…` a callable it invoked — so a consumer rendering the field naively shows a prefixed *object name*, not merely an obvious non-object like `unmodelled-source:lateral`. Consumer-side care is needed for most of the set, not one member of it. The namespaces are separated at the level that matters, not perfectly.

The audit record, and where the boundary is enforced

One trace is emitted per answer. What goes in it is not decided by what the runtime happens to have:

> **The payload is a disclosure decision, not plumbing.**

The request is built **from a tier table**, not from the event object. Every field the emitter may send appears in exactly one tier, which is what makes the table binding rather than advisory — and the consequence is deliberate: "*a field nobody classified cannot ship, because 'the event carries it' is what code does when nobody decided.*" Adding a field to the identity context ships it **off** until someone tiers it.

The second decision is where that boundary sits, and it is the one worth taking to a security review:

> The boundary is enforced **before the request leaves**, not by the receiver discarding on arrival. "*Filtering at the receiver means the bytes already crossed the wire, already sat in a log, already existed on a second machine. 'We do not store it' and 'we do not send it' are different claims and only the second survives a security review.*"

Two placements inside the trace are load-bearing. The answer text goes in an **artifact**, never the top-level field, because the artifact is both what every grader reads and what the trust plane redacts for identities without **reviewer, admin or operator** — "*gradeable and protected are the same place.*" And the semantic references name **what the packet selected**, not what the snapshot held: the record says what the answer used, not what was available to use.

2e. Sources, dialects, and writes — where the engine meets a real database

One dialect to write, many to run

The model writes **DuckDB SQL**, always. The engine parses it as DuckDB, and DuckDB executes it — against a source it has `ATTACH`ed. That means DuckDB-native functions work regardless of what the underlying database is, and the generator has one dialect to be good at rather than four.

Oracle is the exception, and it is a considered one. DuckDB ships `ATTACH` scanners for PostgreSQL, MySQL and SQLite only, and the third-party Oracle scanners reviewed did not meet the bar for a governed path. Oracle is therefore read directly through `oracledb` in thin mode — **1,185 lines**, against 190 for the DuckDB adapter — the largest by a factor of six, which is what a direct integration costs when you decline to bet on someone else's scanner.

The operator-relevant consequence is that **Oracle cannot join a federated query**: it has a direct adapter but no DuckDB scanner, so it must be the only source in a manifest. It is rejected as *unfederatable*, not silently degraded.

The refusals are not a clean taxonomy, and the seam shows. A single-source manifest naming a kind the engine has no adapter for is rejected as an **unknown kind**; the same kind in a *multi-source* manifest is rejected as **unfederatable** instead, because the federated path constructs its adapter directly and never consults the resolver. The advice it then gives — configure it as the only source — is advice that path would refuse.

The Oracle read-only gate, and why privileges cannot answer the question

Every other instrument in the adapter asks *what may this caller do*. That question turns out not to answer the one that matters, and the reason is definer's rights:

> A principal holding `SELECT` on one view and nothing else **caused a row to be inserted**, because a view resolves its references with the **view owner's** rights.

So the caller's privileges cannot establish that a read plane cannot cause a write. What can is `ORA-16000` — the error a **database in read-only mode** returns. "*ORA-16000 is not a privilege verdict. It is the database saying that nothing writes here, whoever asks.*"

Two refinements followed, and both are corrections of things that looked like controls:

A boot sample is not a control. The read-only assertion originally ran once, at runtime construction, and nothing re-probed it. A database reopened `READ WRITE` while the process lived took the assurance with it, silently. The verdict's own text admitted it was time-bounded and the runtime did nothing with that — "*an admission standing in for a control.*" It is now re-probed on a bounded cadence, on the already-leased connection, and says so when the assurance lapses.

The probe had to be measured, not reasoned about. Testing for a write refusal inside a `SET TRANSACTION READ ONLY` block looks like it must fail to distinguish anything — a write there is `ORA-01456` in either mode. Measured rather than assumed, it does distinguish: Oracle checks the database open mode *first*.

The same file carries the reason `prove` (§2b) is dialect-aware. `EXPLAIN <sql>` is Postgres and DuckDB syntax and **is not Oracle's** — it returns `ORA-02000`, "*missing PLAN keyword*", so every Oracle plan would have been refused before execution. Oracle's own `EXPLAIN PLAN FOR` is not the fix either: it **inserts** into `PLAN_TABLE`, which the read-only adapter cannot do. The adapter's tests never traversed the deciders, which is why nothing caught it.

What the ENGINE cannot close, the database can. A small number of source-side constructs can carry effects past any statement-level gate. No amount of engine code closes them, which is why they are handled as deployment preconditions agreed with the data owner rather than as open engineering.

A read-only database closes **both**, and this is measured rather than argued. From the shipped deployment guide:

DATABASE MODE	STATEMENT	RESULT
`READ ONLY`	SELECT through the definer-rights view	`ORA-16000`, no row
`READ ONLY`	`validate()` on the SQL-macro statement	`ORA-62565`, no row
`READ ONLY`	`SELECT count(*) FROM t`	works normally

So the recommended deployment is not one that tolerates these paths; it is the one control that closes them without depending on what the governed schema happens to contain. Reads are unaffected. §6 records the first path's history; the parse-time path is documented in the guide rather than here.

`Writes, and defaults that fail closed`

Writes run through a separate decider — deployment switch, then shape, then table and column authorisation, then target-writable, then proof. `UPDATE` and `DELETE` must be **bounded by a** `WHERE`; an unbounded one is refused rather than executed.

Two signature decisions carry the argument, and both were once the other way:

`write_enabled` defaults to `False`. "*A security parameter whose default is permissive means any caller that forgets it fails open; defaulting to disabled means forgetting fails closed. That is surprising for a library API and correct for a decider.*"

The switch must reach the decider, not only the adapter. It originally set `read_only` on the connection and nothing else — so a deployment with writes disabled still **approved** the write, **executed** it, and reported a refusal assembled from whatever the read-only attachment happened to raise. That is "*the database as the control, which is the principle this project defines itself against*" — the same principle §2b states for reads, found violated on the write path.

The policy argument is required, with no default. It previously defaulted to `None` and became an empty policy — a permissive default in the signature whose sibling is documented as fail-closed for exactly that reason. Empty is a legitimate value; that is precisely why it must be *stated*. "*No policy supplied*" and "*a policy that restricts nothing*" are the same denial and different facts, and collapsing them loses the second.

---

3. What moves the number, measured — BIRD mini_dev PostgreSQL

n = 487, DuckDB-over-PG execution path. **EX = exact match.** The frontier row is `gpt55-duckdb-pg.jsonl`, chosen because it ran the same execution path as the local arms.

> **Read this before reading the table.** In every local arm the **isolated variable is generation**. Enrichment ran on **hosted gpt-5.5**, and embeddings on a hosted endpoint, **held constant across all of them**. So no row here is "bare", none is fully local, and the 17.0% → 50.9% movement is the **deferral** it converts, not enrichment — enrichment does not vary within this table. What the table isolates is which *generation* model and which method (§1a). A fully-local deployment is a different configuration and is not measured here.

ARM	ARTIFACT	EX	GOT-FACTS	DEFERRAL	WRONG
hosted gpt-5.5	`gpt55-duckdb-pg`	50.7%	63.4%	1.0%	48.2%
Qwen2.5-Coder-7B, no generation method	`minidev-pg-7b`	6.4%	6.6%	92.0%	1.6%
7B + guided	`minidev-pg-7b-guided`	33.9%	35.1%	26.7%	39.4%
14B, no generation method	`minidev-pg-14b`	**17.0%**	17.7%	**73.9%**	9.0%
14B + assertive	`minidev-pg-14b-assertive-fixed`	44.8%	47.8%	17.5%	37.8%
14B + guided	`minidev-pg-14b-guided-fixed`	**50.9%**	54.0%	10.3%	38.8%
14B + guided + SC5	`minidev-pg-14b-guided-sc5`	**52.3%**	57.0%	7.8%	39.9%
14B + QLoRA	`minidev-pg-14b-qlora`	51.1%	52.4%	10.3%	38.6%
32B + guided	`cloud-32b-guided`	**50.3%**	53.6%	9.0%	40.7%
DeepSeek-V2-Lite 16B MoE + guided	`cloud-deepseek`	37.6%	41.9%	18.1%	44.4%

> **All ten arms above are beacon v7's grading of the same stored runs.** They are not this report's own 2026-09-06 grading, which scored the same runs between **0.2 and 4.3 points lower** on EX; the predictions are identical and the whole of the difference is the grader. Deferral rates are untouched, because a deferral was never graded against gold. §7.3b is the rule this table satisfies: a number names its grader, not only its run.
>
> **All ten move together because restating one arm would have been worse than restating none.** The findings below are cross-arm *differences* — one of them is 0.6 points wide — and the shift between graders is up to 4.3 points per arm. A table mixing graders would have presented that as like-for-like. Deferral rates are unchanged: a deferral was never graded against gold, so no grader touches it.

Three findings, in order of how much they should change your mind

1. Over-deferral was the entire gap, not capability. The 14B's 17.0% with no generation method sits under a **73.9% deferral rate**. It was not answering wrong; it was not answering. Every intervention that worked — assertive prompting, guided decoding, QLoRA, self-consistency — worked by converting refusals into attempts. The three arms that cut deferral to ~10% or below — guided decoding (10.3%, 50.9%), QLoRA (10.3%, 51.1%) and guided + self-consistency (7.8%, 52.3%) — land in the same **51–52%** band. Two of those three are independent of each other; the third is not independent of the first, because the SC arm is `14B + guided + SC5` and contains guided decoding, adding 1.4 points to it. **Assertive prompting is the exception that sets the rule**: it cut deferral 73.9% → 17.5% and reached 44.8%, six points short. So **among the local mini_dev arms** it is *how far* deferral falls that tracks EX, not merely that it fell — and not *how* it fell: guided decoding constrains the decoder, QLoRA changes the weights, nothing is shared between them, and both land on **exactly 10.3%** deferral with 0.2 points between their EX.

Both qualifiers are load-bearing, and each has a counter-example in this report. On Spider the same 14B's guided arm defers *less* than its unguided one (17.0% against 20.7%) and scores *lower* (4.4% against 5.9%) — there the recovered attempts are wrong, so converting refusals costs rather than pays. It does not hold across models here either: the hosted arm defers **1.0%** and scores **50.7%**, below `14B + guided + SC5` at 7.8% and 52.3% — and that is a mini_dev row, so the suite alone is not the scope either.

Separate the ordering from the fit, because only one of them is robust. Ranked by deferral, the nine local arms — spanning Qwen 7B, 14B and 32B and DeepSeek-V2-Lite — rise in EX at every step but two: 92.0/6.4 · 73.9/17.0 · 26.7/33.9 · 18.1/37.6 · 17.5/44.8 · 10.3/50.9 · 10.3/51.1 · 9.0/50.3 · 7.8/52.3. One exception is the pair tied at **10.3%**, where `14B + guided` and `14B + QLoRA` differ by 0.2 points of EX with no deferral difference to explain it — a tie, not a reversal, and a place the ordering says nothing. The other is the 32B at **9.0%**, which defers less than both of those arms and scores below both: a genuine reversal, and the one place the ordering points the wrong way. The *magnitude* only maps inside a family: `cloud-deepseek` sits 0.6 points of deferral from `14B + assertive` and 7.2 points below it on EX, a wider miss than the 6-point shortfall singled out above. So deferral **orders** the local arms of this suite and does not **price** them, and it does neither for the hosted arm or on Spider.

2. Model size is not what moves it. 14B + guided is 50.9%; 32B + guided is 50.3%. **−0.6 points for double the parameters** — the larger model is behind, not ahead. (This comparison has now read +0.6, +0.9 and −0.6 under three graders of the same two runs. Its sign is inside the noise; its magnitude, under a point either way, is the finding.) The 24GB-practical 14B is the sweet spot, and the 32B is not worth its hardware for this workload.

3. The local model reaches frontier parity on EX. 52.3% against the hosted 50.7% on the same path. **The right word is still parity, not "beats"**: 255/487 against 247/487 is 1.6 points with a ~3.2 point standard error, so the ordering is unresolvable at this n — comfortably so, and more comfortably than under the grader this table previously used. The hosted arm ranks **fourth** on EX, below `14B + guided + SC5`, QLoRA and `14B + guided`, and ahead of the 32B. **Do not read that as the local models winning.** On got-facts the hosted model still leads by **6.4 points** (63.4% vs 57.0%), which is the same conclusion §5 draws at greater length: exact match is measuring output formatting as much as it measures answers, and the arm that loses least to formatting is not thereby the better system. The parity claim is specific to exact match, and exact match is the metric this report spends a section arguing against relying on.

Against published baselines

From `bird-bench/mini_dev` at `b3d4bcb`, EX Evaluation table, PostgreSQL column. **The two reference rows** run *the benchmark's external-knowledge field, a column listing, and no semantic enrichment*, as **one generation at temperature 0**. Read at that revision: `run_gpt.sh` sets `use_knowledge='True'`; `generate_schema_prompt_postgresql` emits only `column_name, data_type, is_nullable` — **no keys and no example rows**, so "full schema" would overstate it; and `gpt_request.py` calls with `max_tokens=512, temperature=0, stop=["--", "\n\n", ";", "#"]` while the instruction says the model need not mention intermediate steps. **Both mnemiq rows below are enriched, by hosted gpt-5.5.**

Enrichment is the difference being measured, and it is not the only one — the others run OUR way, so do not call this margin conservative. The 52.3% row is five candidates against their one generation; state it beside the single-shot 50.9% row or not at all. An earlier draft of this caption claimed the reference models got chain-of-thought and ours did not, making the margin conservative. That was wrong: `cot='True'` never reaches the prompt — `gpt_request.py` uses it only to name the output file, `prompt.py` appends the trigger phrase unconditionally, and the answer-only instruction plus those stop tokens mean no reasoning is emitted either way. The published PG predictions for these two rows carry no `_cot` infix. The caption must never travel with the mnemiq rows, and neither may our own 17.0%, which is enriched too (§3).

SYSTEM	EX
GPT-4	35.8%
Llama3-70B	29.4%
mnemiq, 14B + guided	**50.9%**
mnemiq, 14B + guided + SC5	**52.3%**

+15.1 points over GPT-4 at 50.9%, **+16.5** at 52.3%. This is a claim about the **pipeline**, not about model quality: a 14B inside mnemiq against a frontier model on the standard protocol. It must never be phrased as "our 14B beats GPT-4".

> A previously circulated figure, "bare Qwen-32B 22.96", has been **withdrawn**. That number is from the same repository's LiveSQLBench-Base-Lite table — a different benchmark — and mini_dev's EX table carries no Qwen row at all. Our own no-generation-method 14B at 17.0% is the honest same-protocol anchor, and it is lower than the figure removed.

4. What none of it moves — Spider 2.0-lite

n = 135, local SQLite databases, 30 schemas. **Headline is exact match**, because got-facts on this suite is scored on a strict subset of gold columns for 46 of 135 items (§7).

> **Artifact paths.** These artifacts are not held in a commit, so a filename alone does not identify one — `spider2-results.jsonl` in particular exists in more than one copy with **different contents**. Any republication must carry the checkout, not just the name.
>
> **Re-graded 2026-09-06, and only one arm moved.** All six Spider artifacts were recomputed on the current rule through Spider's OWN grading path — against the published gold result CSVs, accepting any alternative the benchmark accepts. The four local arms and `spider2-full-k24` re-grade to **exactly the labels they had**; `spider2-results` moves 5, from 37.8% to **35.6%**. Every figure in the overlap analysis below is unchanged, because it turns on which LOCAL cases are solved.

ARM	ARTIFACT	EX	WRONG	DEFERRAL
hosted gpt-5.5	`spider2-results.regraded-2026-09-06.jsonl`	**35.6%**	37.8%	8.1%
14B, no generation method	`spider2-qwen2.5-coder-14b.jsonl`	5.9%	71.1%	20.7%
14B + guided	`spider2-14b-guided.jsonl`	4.4%	76.3%	17.0%
32B, no generation method	`spider2-32b-baseline.jsonl`	2.2%	**12.6%**	**85.2%**
32B + guided	`spider2-32b-guided.jsonl`	5.2%	68.1%	25.2%

Neither constrained decoding nor doubling the parameters moves this. Every local arm sits in a **2–6%** band against a hosted **35.6%** on the identical items — a 6–16× gap.

Do not report an ordering among the four local arms. Best-config against best-config is 8 correct versus 7, one case in 135, and the overlap analysis shows why: across all four arms **14 distinct cases** are solved, **only one by all four**, three only by a 32B arm and six only by a 14B arm. Each arm draws a different handful — the signature of near-chance success, not a ranking. Five of the 14 are cases the hosted model misses.

Why it fails differently, and why that matters

On mini_dev the 14B's failure was **refusal** (73.9% deferral, 9.2% wrong). On spider2 it is **confident error** (71.1% wrong, 20.7% deferral). These methods convert refusals into attempts, and here the attempts are wrong — so the tool that fixed mini_dev cannot fix this, and measurably did not.

A failure-mode re-grade of the 96 wrong cases: **all 96 executed**, zero dialect errors, one item in 135 failed on the model's own syntax. The errors are semantic — 39.6% wrong granularity or grouping, 24.0% wrong aggregate,

20.8% right structure with the wrong computation, 13.5% over-filtered to empty. And they are broad, not confined to a few large schemas: the local 14B solves at least one case in **8 of 30** databases, the hosted model in **25 of 30**.

The one thing scale did buy: calibration

The 32B's no-generation-method arm refuses **85.2%** of questions and is wrong on **12.6%**, where the 14B answers freely and is wrong on **71.1%**. Its refusals are reasoned — "*the tables provided do not contain the flight route distances*" — and forcing it to answer collapses it onto the 14B's profile (68.1% wrong). **The parameters bought knowing-when-you-don't-know, not capability.**

For a system whose claim is *refuse rather than guess*, the 32B no-generation-method arm is the safest local configuration measured anywhere in this program. It answers 15% of questions. That is a governance posture, not a benchmark score, and it should never be quoted as one.

4b. Against Databricks Genie and Snowflake Cortex Analyst

Three suites, three products, **three repeat runs of one configuration each**, so what the ranges below show is run-to-run variance on both sides rather than a best-of. Source: beacon, one grader version throughout — the suites carry **zero regrade events** (`GET /v1/suites/{id}/history`, empty for all three), so no row was scored under a different rule than any other. All 77 runs across the three suites report `completed` and none carries an `invalidated_at` (`GET /v1/suites/{id}/runs`): 43 on BIRD, 15 on Spider 1, 19 on Spider 2. The matrix returns fewer ROWS than runs where a row aggregates repeats — Spider 2's 19 runs collapse to 18 rows because one row carries two — so the two counts differ by design and not by drift.

Every figure IN THIS SECTION comes from beacon's `GET /v1/suites/{id}/results-matrix`, and the `config` column reproduces beacon's stored `config_label` verbatim so a row can be found rather than reconstructed. The suites:

SUITE
BIRD mini-dev
Spider 1
Spider 2.0-lite

Each id was checked by fetching its matrix and confirming the shape of the WHOLE matrix — not just the rows quoted here — against what that suite should return: BIRD 43 rows spanning n=479–496, Spider 1 15 rows spanning n=1016–1022, Spider 2 18 rows spanning n=34–260 — the 260 being an aggregated row that SUMS its repeats (`n_runs=2`, and COMPLETE single runs on that suite return 125–134, so 260 is two runs averaging 130 — the split is not

recoverable from the matrix and repeats of one config do differ in n elsewhere on this suite), and the 34 an incomplete push, which is itself a single run and why the band above is qualified. A transposed but valid id returns another suite's matrix and renders perfectly, so row count and denominator span are the tell; the cited rows alone would not be, since several suites contain runs of similar size.

> **The condition these numbers were taken under, which belongs with them and not in a footnote.** All three systems saw the schema and nothing a human wrote. mnemiq's `tier1 structural` arm runs deterministic enrichment only — no LLM semantic layer, no operator dictionary, no worked examples. Genie ran on bare spaces. **Cortex Analyst ran without its Verified Query Repository**, and that is an asymmetry rather than a matching handicap: a VQR is human-confirmed question→SQL, and mnemiq has no equivalent of it. Our `enrich_examples` are LLM-generated, which is a different mechanism, and on this suite they measured *worse* — so "mnemiq would score higher with its own exemplars" is not a claim this evidence supports.
>
> Populating a VQR from the benchmark's own gold SQL would be answer leakage, not a fairer configuration. Doing it properly means building one from a held-out split, which has not been done. **So read every row as cold-start behaviour on schema alone, and read none of them as any product's ceiling** — least of all Cortex's, whose design leans hardest on the mechanism that is switched off.

What the configuration labels mean, since §7.3 lets runs share a table only when each row names its condition:

LABEL	MEANING
`tier1 structural`	mnemiq with deterministic enrichment only — schema discovery, per-column profiling, code grounding. The LLM semantic layer is **off** (`--no-semantic`), and there is no operator dictionary or worked-example set
`tier2 semantic`	the same, plus phase 5 — the LLM proposes meanings for columns nothing deterministic could resolve (§2)
`autopilot views · no verified queries`	Cortex Analyst against auto-generated semantic views, with the Verified Query Repository empty
`bare-spaces·pre-FK` and `bare-spaces·post-FK`	Genie against a minimal workspace. **beacon records the label and not its definition**, so this report cannot tell you what changed between them

That last row is a gap and is worth naming rather than smoothing. The obvious reading of `FK` is whether foreign keys were declared to the product before the run, which would matter for join inference — but **the direction reverses between suites**: pre-FK is the better Genie arm on BIRD (39.5 against 37.5 median) and the worse one on Spider 1 (63.1 against 63.6). A single mechanism that helps on one suite and hurts on another needs an explanation, and nothing in the run record supplies one. Both arms are reported above rather than the flattering one; neither is quoted as Genie's number.

BIRD mini-dev

SYSTEM	CONFIG	N	EX — MEDIAN (RANGE)	GOT-FACTS — MEDIAN (RANGE)
mnemiq (gpt-5.5)	tier2 semantic	479–480	**52.3%** (51.1–53.2)	**64.8%** (64.7–65.3)
mnemiq (gpt-5.5)	tier1 structural	482	**51.0%** (50.0–51.5)	64.1% (63.3–64.1)
Databricks Genie	bare-spaces·pre-FK	496	39.5% (37.9–39.7)	53.2% (53.2–53.6)
Databricks Genie	bare-spaces·post-FK	494–496	37.5% (36.4–38.3)	52.6% (52.4–54.2)
Snowflake Cortex	autopilot views · no verified queries	494–495	33.6% (32.3–33.9)	49.0% (48.5–49.1)

Spider 1

SYSTEM	CONFIG	N	EX — MEDIAN (RANGE)	GOT-FACTS — MEDIAN (RANGE)
mnemiq (gpt-5.5)	tier1 structural	1022	**69.2%** (68.4–69.5)	**80.2%** (79.6–80.4)
mnemiq (gpt-5.5)	tier2 semantic	1022	68.5% (68.0–68.7)	79.5% (79.4–79.6)
Databricks Genie	bare-spaces·post-FK	1021–1022	63.6% (62.9–64.2)	77.4% (77.3–77.7)
Databricks Genie	bare-spaces·pre-FK	1016–1020	63.1% (62.7–63.9)	76.4% (75.8–76.6)
Snowflake Cortex	autopilot views · no verified queries	1021	56.2% (56.2–56.9)	77.1% (76.9–77.2)

Spider 2.0-lite

SYSTEM	CONFIG	N	EX — MEDIAN (RANGE)	GOT-FACTS — MEDIAN (RANGE)
mnemiq (gpt-5.5)	tier2 semantic	130–132	**36.6%** †	**59.8%** †
mnemiq (gpt-5.5)	tier1 structural	130–132	34.6% (34.1–40.0)	56.9% (56.8–63.8)
Databricks Genie	bare-spaces·pre-FK	34–125	29.1% (28.8–29.4)	44.2% (41.2–47.2)
Databricks Genie	bare-spaces·post-FK	133–134	27.1% (26.3–28.4)	45.1% (44.8–46.6)
Snowflake Cortex	autopilot views · no verified queries	132	17.4% (one run)	24.2% (one run)

† The `tier2` row is carried from the tracker as a three-run **mean**; no per-run range is published for it. Every other cell in these three tables is a median with its range, and the two are not the same statistic — the tracker's headline figures for the rows above are the means 52.2/64.9 (BIRD `tier2`), 69.0/80.1 (Spider 1 `tier1`) and 36.2/59.2 (Spider 2 `tier1`), computed over exactly these runs.

What these show, and one thing they show that is not the headline

mnemiq's range clears both products on every suite and both metrics — six of six, no overlap. The narrowest of those six is Spider 1 got-facts, where mnemiq's floor of 79.4% sits 1.7 points above Genie's ceiling; the widest is BIRD exact match, where mnemiq's worst run beats Genie's best by more than ten points.

The semantic layer's own contribution does not separate. `tier2 semantic` and `tier1 structural` differ only in phase 5 — the LLM proposing meanings for columns nothing deterministic could resolve — and they were run on BIRD, Spider 1 and Spider 2.0-lite, two metrics each. **One of those six separates**: BIRD got-facts, where `tier2`'s floor of 64.7% sits 0.6 above `tier1`'s ceiling. The other five overlap, and **on Spider 1 both metrics favour `tier1`** — the arm without the semantic layer. On Spider 2.0-lite `tier2` leads on both, by 0.4 and 0.6 of a point against a `tier1` range six points wide, which separates nothing.

Non-overlap is the bar this section sets for itself — *six of six, no overlap* — and five of these six fall short of it. It is also the shape this report refuses to credit elsewhere: a mechanism that leads on one suite and trails on another needs an explanation the run record does not supply. **So the LLM semantic layer's contribution to accuracy is unresolved here** — which is a statement about this evidence, not about enrichment, whose deterministic phases are present in *both* arms and are not what varies.

The ordering between the two products does not hold everywhere, and the exception is the interesting one. Genie clears Cortex in five of six comparisons. It fails on **Spider 1 got-facts**, where the two are indistinguishable — Genie 75.8–77.7% against Cortex 76.9–77.2%, one range inside the other — while their exact-match scores differ by seven points (62.7–64.2 against 56.2–56.9). Two products, near-identical at finding the right data, seven points apart at returning it in the shape a strict grader accepts.

Spider 1 is the suite worth reading twice, and it is not the flattering one. Cortex trails mnemiq by 13 points on exact match and sits **within 3 points on got-facts** (77.1% against 80.2%, ranges 76.9–77.2 against 79.4–80.4). Almost its entire exact-match deficit is answer *shape* rather than comprehension — it is finding the right data and returning it in a form strict grading rejects. That is precisely the argument §5 makes about exact match on our own numbers, and it does not stop applying when the arm it flatters is someone else's. A reader who takes the 13-point EX gap as a capability gap has been misled by the metric, and this report has already spent a section saying so.

mnemiq's Spider 2.0-lite band is wide and that is a real finding, not noise to smooth over. 34.1, 34.6 and 40.0 across three runs of one configuration is a six-point spread on n≈131 — far wider than Genie's two

points on the same suite, and wider than mnemiq's own spread on Spider 1 (1.1) or BIRD (1.5). Warehouse-shaped schemas destabilise this pipeline as well as depress it, which §4 measures from the inside and this table measures from the outside.

What is not claimed

These are **beacon's** `exact_rate` and `got_facts_rate`. They are not the figures in §3 and §4 and must never be tabled beside them — those are mnemiq's own harness on its own denominators, and the two disagree by construction (§7.1, §7.1b). Nothing here says a 14B is competitive with these products; every mnemiq row above is gpt-5.5. And nothing here is a ceiling: each system was measured on what it does out of the box, which is a useful question and a different one from how well it does once someone has curated for it.

---

5. Where the claim stops

- **Warehouse-shaped schemas.** Open weights at 7B–32B are not close on spider2, and it is not a prompt problem or a size problem. The honest recommendation for that workload is a hosted model or a verifier.
- **The wrong-rate floor.** On mini_dev a stubborn **38–41%** wrong rate survives every calibration method tried **on the 14B** — assertive 37.8%, QLoRA 38.6%, guided 38.8%, guided + self-consistency 39.9% — and 2× the parameters (the 32B sits at 40.7%). Only a verifier moved it, and only by trading EX for it.
- **Enrichment was held fixed in the spider2 arms.** All four local arms used contracts built by the 14B, so §4 is a statement about *generation*. A 32B enriching for itself is untested.
- `got_facts` **on spider2 is not one metric.** See §7.

---

6. What the engineering record looks like

The findings register is counted by script rather than typed. The dominant defect class, found repeatedly, is a **collapse between an absence and a failure** — a column that could not be measured looking identical to one with nothing to measure; a control that cannot fail; a test that passes for a reason unrelated to the code.

Two examples that shaped the product:

- **The Oracle read-only basis.** `read_only=True` rested on a statement gate that cannot see a SELECT reaching a writing function through a view. The database's own read-only mode closes it, which is now measured at boot and re-measured on a TTL, with three distinguishable verdicts (`constrained` / `gate_only` / `unverifiable`) rather than a boolean.
- **Per-column profiling outcome.** A column that failed to profile was persisted identically to one that could not be profiled. Closed for the per-column path; **still open** where every column of a

table fails, which is the case most likely to be a misconfiguration.

---

7. Conventions and reproducibility — read before quoting any number

1. **Denominator.** beacon's `n_graded` includes DEFER, and that is what we publish. mnemiq's `answerable` denominator excludes correct refusals, needs a field BIRD and spider2 do not carry, and is therefore uncomputable on two of three suites. It is reported for the private corpus only, labelled. Every quoted cell names its denominator.

1a. **Every BIRD and spider2 number here is run WITH the benchmark's external knowledge, and no deployment gets that.** Both suites ship a human-written hint, and the harness appends it to the question by default (`load_bird(with_evidence=True)`; spider2's reference documents). Measured in the artifacts: **485 of 487** BIRD records carry a `Hint:`, and **13 of 135** spider2 records do.

This is not a detail. It is the largest single effect BIRD reports about itself — GPT-4 scores **54.89% with evidence and 34.88% without** (the BIRD paper, arXiv 2305.03111), a 20-point gap — and BIRD's own leaderboard carries an *Oracle Knowledge* column so a submission must declare it. Two consequences:

- **A quoted number must name the setting**, exactly as it must name its grader (3b). Ours are all with-knowledge. A comparison against a without-knowledge figure is not like-for-like and is worth more than any finding in §3.
- **The ten-arm table does not isolate what ENRICHMENT contributes.** Enrichment is held constant across the arms *and* a human hint is present on 99.6% of BIRD questions, so they isolate model and method against a fixed knowledge background and say nothing about the value of the knowledge itself. **§3's "Against published baselines" is the exception and is unaffected**: it compares enriched against unenriched with the external-knowledge field present on *both* sides, which is exactly the like-for-like this rule asks for.

1b. **On spider2, beacon and this report deliberately publish DIFFERENT metrics, and both are right.** Beacon follows the suite's declared `got_facts`, which is the benchmark's own convention and which the upstream evaluator supports. This report uses `exact_match` (rule 2 below). Same runs, same rows, different numbers — on `spider2-32b-guided`, beacon publishes **9 PASS** under got_facts where §4 publishes **7 correct** under exact_match. Neither figure travels without naming its metric. The gap is exactly the 3 results of 840 where the strict and tolerant readings disagree; state it as counts, because 0.36% of the corpus is 28% of the number a reader looks at. 2. **spider2 headline is `exact_match`.** `got_facts` there honours upstream `condition_cols`, which restricts scoring to a subset of gold columns on **46 of 135** items — 49 carry the annotation and 3 of those name every column, so restrict nothing — — so summing it adds "whole table right modulo shape" to "one label column right". 3b. **A number names its GRADER, not only its run.** Two figures from the

same artifact are not comparable if they were graded under different rules, and nothing in the artifact says which rule was used — the outcome label is just a word. This is not hypothetical: the 2026-08-06 change — a 1% relative numeric band replaced by beacon's bounds, and column order counting for exact match — moved 49 labels across the ten arms of §3, up to 2.5 points on one arm, against findings that turn on differences under a point. Re-graded artifacts carry the date in the filename (`*.regraded-2026-09-06.jsonl`) and every changed record keeps its `outcome_as_run`, so a reader can see both readings. **Re-grade the whole comparison or none of it** — one arm on the new rule beside nine on the old is worse than a table that is uniformly stale, because it looks like-for-like.

3. **One quantity, one run.** A number is never sourced from two runs. Two runs may share a table when each row names its condition.
4. **Pooled figures are labelled.** Spider's 36.5% is 148/405 — the mean of three hosted runs over the *same* 135 items, and not 405 independent observations. Publish it as "mean of three runs, n=135 each, spread 34.8–37.8".
5. **Pin to revisions, not dates.** Codebase figures moved three times in three days. **Every line count in this document is physical lines — `wc -l` over the tracked source files of the named language at the named revision, blanks and comments included.** A SLOC tool such as `cloc` reports a materially smaller number for the same tree, and without the method stated a reader cannot tell whether the document is wrong or their tool counts differently.
6. **`source_rev`, prospectively.** No run produced from here on is published unless its meta carries `source_rev` and that rev is unmarked. Existing artifacts are grandfathered and labelled — some carry a `-dirty` rev, and that is disclosed rather than hidden.

The verifier, reproduced rather than cited

An earlier draft said the judge/verifier figures were not recomputable, on the theory that the runner lacked the telemetry. **That was wrong about the cause and wrong about the consequence.** The per-layer split came from replaying the verifier over a saved run, and replay needs no GPU — `scripts/run_verify_replay.py` still does it. `local-14b-verify.jsonl` is a genuine live sanity+14B-judge run, and its outcomes prove the judge engaged (EX 48.5%, facts 52.2%, defer 19.1%, wrong 32.4% on beacon v7, alongside §3); its null `judge_engaged` / `verify_layer` fields are a **recording** gap, not evidence the layer never ran. The earlier analysis was right about the artifact; only my inference that a re-run was therefore needed was wrong.

Re-run over `minidev-pg-14b-guided-fixed.regraded-2026-09-06.jsonl` (487 records). **The labels are not the ones the July figures used**: that run was graded on 2026-07-19 and the grading rule changed on 2026-08-06 — a 1% *relative* band replaced with beacon's bounds, and column order made to count for exact match. Re-executing every case's stored SQL against mini-dev

Postgres — 437 gradable cases, 0 execution failures, no model involved — moves five labels, and the denominators with them: **186 wrong / 237 correct becomes 190 / 232**, EX 48.7% → 47.6%, wrong 38.2% → 39.0%.

LAYER	WRONG CAUGHT	CORRECT LOST	EX	WRONG
sanity only	**31 / 190**	**2 / 232**	47.6% → 47.2%	39.0% → 32.6%
grounding only	28 / 190	16 / 232	47.6% → 44.4%	39.0% → 33.3%
sanity + grounding	49 / 190	18 / 232	47.6% → 43.9%	39.0% → 29.0%
judge @ 0.5	**94 / 190**	22 / 232	47.6% → 43.1%	39.0% → **19.7%**
judge @ 0.8	123 / 190	40 / 232	47.6% → 39.4%	39.0% → 13.8%
judge @ 0.9	144 / 190	53 / 232	47.6% → 36.8%	39.0% → 9.4%
sanity **+** judge @ 0.5	95 / 190	22 / 232	47.6% → 43.1%	39.0% → 19.5%

This ladder has not been replayed under beacon v7. Its denominators and its EX and wrong-rate anchors are the 2026-09-06 grading's, which is why they do not match §3's table; the catch counts are properties of the answers rather than of the labels, and are reported here unchanged. Replaying it under the current grader is outstanding.

Every catch count is unchanged by the re-grade. Sanity caught 31 and lost 2 on the old labels and on the new ones; grounding 28 and 16; the pair 49 and 18. The layers are deciding on the same answers either way — only the denominators they are quoted against moved, which is worth stating because "the numbers changed" and "the behaviour changed" are not the same claim and a corrected denominator invites the second reading.

The judge row is the one that overturns something. An earlier draft of this section said the deterministic layer was "the layer worth having". That was measured against a judge which, unknown at the time, had been switched off by our own token budget: `SemanticJudge` asked for 200 output tokens, a reasoning model spends its budget thinking before it emits, so the provider failed the request and `score` returned its fail-open **1.0 — the value approval returns**. 65% of that sweep's cached scores are exactly 1.0, against 7% once the budget was fixed. The published figure for this layer was 23/186 caught at "perfect precision"; the working judge catches **94** and loses 22, and the precision was never perfect — it was a judge that had stopped flagging.

The corrected reading is a cost ladder rather than a ranking:

- **Sanity is the cheap layer, not the best one.** 31 wrong answers for 2 correct ones, 0.4 points of EX. Nothing else comes close on price, and it needs no model.
- **The judge is the effective layer, and it is not free.** At the same 0.5 cutoff it catches three times as many — 94 — and cuts the wrong-answer rate from 39.0% to 19.7%, for 22 correct answers deferred and one LLM call per answer.
- **The layers do not add up, and the obvious move does almost nothing.** A ladder invites "then run both". Both, beside each other, catch **95** — one more than the judge alone, for no

additional correct answers lost. Thirty of the thirty-one answers sanity catches, the judge already had. What sanity earns in a judged deployment is not catches but calls: `Verifier.verify` returns on the first layer that produces a verdict, so the 83 cases sanity defers never reach the judge — **17% of answers skip the LLM call entirely**. It is a cost filter in front of the judge and a real verifier where no judge runs at all; it is not an accuracy layer stacked on one.

- **Grounding remains the poor trade** it was: 28 caught for 16 lost, which is why it is opt-in.

The judge sweep is certified rather than cited: 487 calls, **0 endpoint errors, 0 unrecovered**, `scoring_complete: true`, artifacts committed beside the run. It took five attempts to get one — the first four each ended at least one case short, and the error records for all of them are in the repo.

One caveat that belongs with the judge row and not the others. These figures are one sweep. The same judge, re-swept over a different corpus at threshold 0.9, produced 43, 45, 48 and 44 catches across four runs — a five-case spread from nondeterminism alone. Read the judge column as the shape of a trade, not as a constant to four significant figures. The deterministic rows carry no such caveat: they are functions of the data and reproduce exactly.