ARDUA

Fidelity in EntailDB

How conversational analytics keeps answers grounded and inspectable

Executive summary

A conversational analytics tool that generates its own query and narrates its own result can be wrong in a way a fixed report cannot: confidently. The model can invent a number no query produced, describe a chart that shows something different from what it says, or recite a statistic that used to be true. EntailDB exists to close that gap — not by making the model more careful, but by building structural checks around it that do not depend on the model being careful at all. This paper describes four distinct mechanisms that make up EntailDB's fidelity discipline, states plainly which ones are proven by controlled measurement and which are evidenced only by the production incidents that motivated them, and shows the concrete engineering evidence behind each claim. Where the evidence is thin, this paper says so — the same discipline EntailDB asks of its own data has to apply to what is said about the product.

The problem: four ways a conversational answer goes wrong

SQL correctness is not the failure mode EntailDB's own incidents show most often. The pattern that recurred during this work looked different: the query ran, returned the right rows, and the prose describing those rows was wrong. That failure has four distinct shapes, and treating them as one problem — "the model hallucinates" — produces one broad, ineffective fix instead of four narrow, effective ones:

- The model states a number that no query ever returned, typically after the tool call that should have supplied it failed or came back empty.
- The model fabricates a citation or a download link that was never present in any tool result.
- The model's own sentence contradicts the data it just displayed — a stated count that doesn't match a chart's actual bars, a figure attributed to the wrong basis.
- The underlying data is structured in a way that produces a wrong number even when the model is not fabricating anything — a join that silently multiplies rows before they're summed, a rollup that looks current but isn't, a hierarchy that leaks values between levels.

EntailDB addresses each with a different mechanism, not one general-purpose instruction to "be accurate."

Pillar 1 — Runtime guards

The strongest measured result in EntailDB's fidelity work is also the simplest: an explicit instruction telling the model which source of information in its context is authoritative. In a controlled evaluation, a model reciting a stale, hand-pasted statistic from its prompt reproduced the failure 20 times out of 20 attempts on one model version and 9 times out of 20 on a newer one. Adding the instruction closed it to zero on both. That is the single largest fabrication-rate change measured anywhere in this project, and it is a one-paragraph fix — evidence that not every problem here needs a structural solution, only the right one.

Other runtime guards exist and are more narrowly proven. A duplicate-call refusal — declining to re-run a tool call identical to the one immediately before it — shipped, was refined within days against real usage, and had part of that refinement reverted the same day when it broke an unrelated feature. That is not a story of instability; it is what active measurement against production looks like when a team is willing to revert rather

than defend a change that turned out to be wrong. A separate guard withholds an answer entirely when a query phase collects zero results, rather than letting the following phase narrate from nothing. Its first, blunt version measured a genuinely divergent effect across two models: over 200 runs each, it roughly halved the fabrication rate on one model, but on the other it fired 40 times without preventing a single fabrication, replacing 40 accurate, specific refusals — for example, "the data warehouse is currently unavailable (no connections in the pool)" — with one generic sentence. Rather than ship that as a permanent per-model setting, a refined version was built that distinguishes an answer that asserts something from one that is declining to, and fires only on the former. Measured against the same two models and validated against 87 real suppressed answers from both ablations, the refinement cut the second model's false suppressions from 40 to 1, and on the first model raised its catch rate from 14 to 18 fabrications while cutting its own false suppressions from 33 to 9 — precision rising from 20% to 67%. The refined guard ships enabled by default.

Pillar 2 — The link allowlist

A model asked repeatedly, in four separate places in its instructions, not to fabricate download links kept doing it anyway — inventing javascript:void(0) placeholders, bare # anchors, and fully-formed URLs no tool result ever produced. Instruction did not work here the way it worked for pillar 1, because a fabricated link is generative: there is no single authoritative source in context to point the model back to. The fix instead makes the failure structurally unreachable — a streaming filter builds a per-turn allowlist from URLs actually present in tool results and strips anything else before it reaches the user.

In production, this catches roughly one confirmed fabrication per 200 conversational turns — a rate too low for a controlled evaluation of twenty runs to reliably observe at all, which is precisely why the honest record of this control lives in production telemetry rather than an eval score. A low catch rate is not evidence the control is unnecessary; it is evidence the failure it targets is rare enough that only real usage, not a lab test, can see it.

Pillar 3 — Post-hoc grounding checks

The first three pillars catch a model inventing something. This one catches a model misdescribing something it correctly retrieved. After an answer is generated, a grounding check compares any currency figure the model states against every value actually available to it — individual cells, and their sums, minimums, and maximums, including rounded or rescaled versions within half a percent — and flags a figure with no such basis. A second, narrower check compares a stated count of a plotted artifact — "the chart below shows three years" — against what was actually rendered, and flags the mismatch when they disagree. Both were built after a live session in which a four-bar chart was described in prose as "the most recent three years."

Both checks are deliberately narrow, and the scope was a design decision, not an oversight: they catch invention, not misstatement. A dollar figure computed correctly on a pre-tax basis and a different one computed on a tax-inclusive basis are both "grounded" in the data, even though they disagree by a meaningful margin — this mechanism does not adjudicate which basis is right, only whether a number was invented outright. Percentages, growth rates, and bare integer counts fall outside its scope entirely, for the same reason. It closes one specific gap cleanly rather than claiming a broader guarantee it cannot back.

Pillar 4 — The schema profiler and the traps it catches

The fourth pillar addresses the case where the model isn't fabricating or misdescribing anything — the data itself is set up to produce a wrong answer, and only a mechanism that actually understands the schema's structure can see it coming. EntailDB's schema profiler generates a live, refreshed profile of a connected database and derives a set of facts about it automatically, rather than depending on a hand-maintained document that inevitably goes stale. The facts it generates fall into three families:

Shape traps — a column that isn't what it claims to be: always null, effectively constant, silently enumerated to a small set of values, drifting from its declared format, or a table or column declared in the schema but absent from the data actually populating it.

Temporal traps — a number that is stale or incomplete without saying so: a rollup that looks current but hasn't refreshed, a period that is only partially elapsed and would understate a real trend if read as complete, a historical figure that varies enough across time that a single snapshot misrepresents it.

Structural traps — a join or hierarchy that distorts a result even though every individual row is correct. The most concrete example: a live conversation correctly named the right category in answer to "which category declined most," but reported a dollar figure inflated between 14 and 71 times, because the query summed an order-level subtotal across a join that fanned out to line-item grain — counting each order's total once per matching line item instead of once per order. A guard now checks this directly: it compares how many times a join actually visits a table against that table's real row count, and flags an aggregate computed on a fanned-out column regardless of what the query groups by. A related detector distinguishes a genuinely discovered relationship between two columns — found by measuring actual value overlap when column names don't match — from a guessed one, and reports its confidence accordingly rather than treating both as equally certain.

This is the part of the system with the deepest evidence base, because it has been run against real, differently-shaped production databases rather than only measured in a lab. Each new database engine tested — PostgreSQL, MySQL, SQL Server, and a document-oriented MongoDB deployment — surfaced a defect the previous ones could not: a SQL function that doesn't exist on one platform, a counting rule that silently double-counts null values on another, a join-type mismatch that produces a wrong answer rather than merely a missing one on a third. That pattern is itself the argument for why this work has to be tested against real, varied systems rather than a single reference schema: a profiler that only ever sees one kind of database will look correct for exactly as long as it stays untested against a second kind.

Measured against a real, hand-maintained internal prompt of over 800 lines, roughly half of its content turned out to be genuine data semantics rather than boilerplate — and about forty percent of that half was mechanically derivable by the profiler alone, without anyone having to notice, write down, or maintain it. What's left after automation is a small, genuinely irreducible residue: judgment calls no profiler can make, like which of two time columns is UTC and which is local wall-clock, or that a column named generically actually holds a specific business quantity.

The discipline behind the numbers

Every number in this paper has a defect history behind it, and that history is itself part of the evidence, not an embarrassment to omit. The standing rule is blunt: read the raw model output behind any reported number before trusting the number. That rule exists because scoring a fabrication rate is harder than it sounds — a refusal that names a figure only to decline repeating it can be scored as if it had repeated it; a test fixture that returns columns a query never asked for can make a model distrust the tool and have that distrust logged as restraint instead. Applying the rule caught eighteen separate measurement defects across seven evaluation runs — twelve in how a fabrication was scored, five in how a test case was constructed, and one in the evaluation harness itself — every one of them, without exception, inflating the apparent fabrication rate before it was caught and corrected. That is the discipline behind this paper's numbers, not a caveat on them.

That same discipline draws a line this paper holds to throughout: a control evidenced by a controlled evaluation (the anti-fabrication instruction, the empty-collection guard's model-dependent behavior) is described differently from a control evidenced only by the live production incident that motivated it (the duplicate-call refusal, the fanned-out join guard, most of the newer profiler fact types). Both kinds of evidence are real. They are not the same kind of proof, and this paper does not present them as if they were.

What this does not claim

None of these four pillars amounts to a formal guarantee of correctness, and none is described here as one. The runtime guards and the link allowlist target specific, previously-observed failure shapes; a new failure shape discovered tomorrow gets a new, narrow control, not automatic coverage from an existing one. The grounding checks in pillar 3 catch invented figures, not figures computed on a debatable basis — two honestly different numbers can both pass. The schema profiler's fact types are heuristic detectors tuned against real databases encountered so far; a fact type that would catch a trap in a database shape not yet tested may not exist yet. And fidelity, in every sense used in this paper, is evidence about whether an answer remains grounded in what the data actually shows — it says nothing about who is allowed to see that data in the first place. That is a separate, deliberately separate concern, governed by a different mechanism entirely.

Relationship to Private AI Analytics

EntailDB's fidelity discipline and Ardua AI's governance layer answer two different questions, and neither substitutes for the other. Fidelity asks: is this answer entailed by what the data actually shows? Governance asks: was this model, this provider, and this action ever authorized to see this data in the first place? A perfectly grounded, fully accurate answer can still be a governance failure if it reached a provider that should never have seen the underlying database's classification — and a perfectly governed request can still return a fabricated or misleading answer if nothing checks the prose against the rows. EntailDB owns the first question and is answerable for it on its own, independent of any governance layer, exactly as this paper has described. The mechanism connecting the two, and Ardua AI's own account of what it decides and enforces, is described in the companion paper, Private AI Analytics: EntailDB + Ardua AI.

Conclusion

Fidelity is not one property EntailDB has — it is four separate, independently-motivated mechanisms, each built in response to a real production incident, each proven to a different, honestly-stated degree, and each with an evidence trail that includes the mistakes made while measuring it. That is a deliberately narrower claim than "the model doesn't hallucinate," and it is the claim EntailDB can actually stand behind: a specific answer, checked against a specific mechanism, backed by a specific incident or a specific measured number. Organizations evaluating conversational analytics for their own data can ask to see any of these mechanisms run against their own schema before deciding whether to rely on it.

ARDUA, INC.

Appendix A — Fact-type and guard reference

Name	Trap family / pillar	What it checks	Evidence
`always_null` / `mostly_null`	Shape	A column that never or almost never carries a value	Live-verified, multiple database engines
`constant`	Shape	A column with effectively one distinct value	Live-verified, multiple database engines
`enumerated`	Shape	A column silently restricted to a small, fixed value set	Live-verified, multiple database engines
`format`	Shape	A column whose values drift from an inferred format pattern	Live-verified, multiple database engines
`empty_table`	Shape	A table declared in the schema but holding zero rows	Live-verified, multiple database engines
`declared_but_absent`	Shape	A schema object with no matching real data	Live production incident (a feature built on a table and column that never existed)
`stale_rollup`	Temporal	A precomputed aggregate that has stopped refreshing	Live-verified
`partial_period`	Temporal	A time period that is only partially elapsed, understating a trend if read as complete	Live-verified, refined against real usage
`history_variation`	Temporal	A figure whose value varies materially across time, misleading as a single snapshot	Live-verified
`join_miss`	Structural	A join key with low or no match rate, including keys discovered by value overlap rather than name	Live-verified across engines; one engine's join-probe budget produced a wrong fact, not just a missing one, under realistic table width
`leaky_hierarchy` / `ambiguous_hierarchy_value`	Structural	A hierarchy level that leaks values between categories, or a value with ambiguous placement in it	Live-verified, refined against real usage
`entaildb.sql_fanout` (app-level guard, not a profiler fact)	Structural	A GROUP BY aggregate summing a column from a table a join has fanned out, inflating the result regardless of grouping	Live production incident — a real query inflated a dollar figure 14x–71x; guard measured against that case
Anti-fabrication instruction	Pillar 1 (runtime)	Recitation of a stale, hand-pasted figure competing with a live source	Controlled evaluation: 20/20 and 9/20 → 0/20 across two model versions
Empty-collection guard	Pillar 1 (runtime)	Narration produced after a data-collection phase returns nothing	Controlled evaluation, two ablations plus a refinement validated against 87 real suppressed answers: blunt version net-harmful on one model (40 fired, 0 caught, 40 clean answers lost); refined version cuts that to 1 lost while raising the other model's catch rate 14→18 and precision 20%→67%; ships enabled by default
Duplicate-call refusal	Pillar 1 (runtime)	An identical tool call re-run back-to-back	Shipped; refined and partially reverted against real usage within days; not

Name	Trap family / pillar	What it checks	Evidence
			yet in the controlled eval harness
Link allowlist	Pillar 2	A markdown link to a URL never present in any tool result	Production telemetry: ~1 confirmed catch per 200 turns
`ungrounded_figures`	Pillar 3	A stated currency figure with no supporting cell, sum, min, or max in the collected data	Named incident (ADR 0017); scope explicitly excludes percentages, growth rates, and bare integers
`miscounted_artifact`	Pillar 3	A stated count of a chart or plotted artifact that contradicts what was actually rendered	Named incident: a 4-bar chart described as "the most recent three years"

Appendix B — Evidence and methodology

This paper cites numbers of two different kinds, and they are not equally strong. A controlled evaluation runs a fixed set of scripted cases against a model under repeatable conditions and counts outcomes directly — it can say precisely how often a failure reproduced and how often a fix closed it, but a small case count cannot rule out a rare failure it never happened to trigger. Production telemetry counts what actually happened across real, unscripted use — it can catch a failure too rare for any affordable evaluation to see, but it has no fixed denominator to report a rate against with the same precision. Where a number below has no version or date, it means this review could not independently corroborate it in the current EntailDB repository's committed history — stated plainly rather than presented with false precision.

Claim	Evidence type	Date / version	Sample size	Current status
Anti-fabrication instruction closes stale-fact recitation to zero	Controlled evaluation	2026-08-12 re-score, 800-run suite	20 attempts/model	Ships in the accuracy instruction; current
20/20 → 0/20 on one model, 9/20 → 0/20 on a second	Controlled evaluation	Same run	20/model: `claude-sonnet-4-5`, `claude-sonnet-5`	Historical baseline, reconfirmed 2026-08-12
Empty-collection guard (blunt): net-harmful on one model	Controlled ablation	Fidelity v0.1.30, 2026-08-12	200 runs/model: `claude-sonnet-5`, `qwen3.6`	Superseded, see next row
Empty-collection guard (refined): fewer false suppressions, higher catch rate	Controlled ablation + 87 real graded answers	Fidelity v0.1.32, 2026-08-12	Same 200-run ablations, re-scored	Ships enabled by default (`runner.py`)
Link allowlist: ~1 fabrication caught per ~200 turns	Production telemetry, not a controlled eval	EntailDB's own measurement record: "~1 in 201 turns"	Production volume, no fixed N	Live in the `FidelityRunner` request loop
Fanned-out join inflated an answer 14×–71×	Named production incident + reproducing fixture	2026-09-15	1 incident; 28 new tests	`entaildb.sql_fanout` guard live
800+-line prompt: ~half data semantics, ~40% of that automatable	Internal measurement	Not corroborated in this repository	Not stated	Directional, not yet a reproducible figure
18 measurement defects found across 7	The project's own defect log	Current as of 2026-09-19	12 grader, 5 fixture, 1 harness	Running count, not a ceiling

Claim	Evidence type	Date / version	Sample size	Current status
evaluation runs				

Three of these claims (the accuracy instruction, the empty-collection guard's two ablations, and the fanned-out join incident) were independently re-verified against the EntailDB repository's own source code and change history for this revision, not re-copied from an earlier draft of this paper.